



FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA – UNIFOR

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

Matheus Ciríaco Cerqueira Cunha

FORTALEZA

2012

Matheus Ciríaco Cerqueira Cunha

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

Dissertação apresentada ao Programa de Pós-Graduação em Informática Aplicada (PPGIA) da Universidade de Fortaleza como parte dos requisitos necessários para a obtenção do grau de Mestre em Informática Aplicada.

Orientador:

Prof. Dr. Nabor das Chagas Mendonça

Co-orientador:

Prof. Dr. Américo Tadeu Falcone Sampaio

UNIVERSIDADE DE FORTALEZA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA APLICADA (PPGIA)

FORTALEZA

2012

C972a Cunha, Matheus Círiaco Cerqueira.
Um ambiente programável para avaliar o desempenho de aplicações em
nuvens de infraestrutura / Matheus Círiaco Cerqueira Cunha. - 2012.
84 f.

Dissertação (mestrado) – Universidade de Fortaleza, 2012.
“Orientação: Prof. Dr. Nabor das Chagas Mendonça.”
“Coorientação: Prof. Dr. Américo Tadeu Falcone Sampaio.”

1. Computação em nuvem. 2. Internet. 3. Arquitetura de computadores.
I. Título.

CDU 681.3:004.451.624

Matheus Ciríaco Cerqueira Cunha

**Um Ambiente Programável para Avaliar o Desempenho
de Aplicações em Nuvens de Infraestrutura**

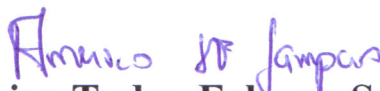
Data de Aprovação: 21/09/2012

Banca Examinadora:



Prof. Nabor das Chagas Mendonça, Ph.D.

(Orientador – UNIFOR)




Prof. Americo Tadeu Falcone Sampaio, Ph.D.

(Co-orientador – UNIFOR)



Prof. Nelson Luis Saldanha da Fonseca, Ph.D.

(Membro – UNICAMP)



Prof. Raimir Holanda Filho, Doctor.

(Membro – UNIFOR)

DEDICATÓRIA

Para minha amada esposa, **Marlene**,
meu carneirinho, **Nina**, e
meu coelhinho, **Sara**.
:-)

AGRADECIMENTOS

Agradeço por meus pais terem me ensinado e dado condições para entender que pobre tem que estudar. Meus pais que, quando mais precisaram, contaram com o apoio incondicional de Tio Jairo, Tio Rege e Tia Sandra. Cada um deles teve uma enorme contribuição na minha jornada, por isso eu sou grato a todos! Pronto, esses cinco primeiros merecem destaque. Mas muitos familiares contribuíram, não quero ser ingrato e deixar de citar nomes. Por isso só irei nomear mais dois familiares, minhas madrinhas Sanara e Ivana. Realmente sou uma pessoa de sorte.

Agradeço por Nabor e Américo terem conduzido esta pesquisa. No início eu tive muita dificuldade, pois eu sentia bastante pressão de Nabor, mas Américo estava sempre atento e disposto a ouvir minhas lamentações. Só na metade do mestrado que consegui entender o que estava se passando, foi então que o trabalho começou a fluir, culminando com a semana de orientação na sala de Nabor. Realmente eu aprendi muito, portanto sou grato a vocês dois.

Fechando a trindade, este parágrafo é dedicado aos amigos. Agradeço os companheiros da CMDS, pelas conversas sem fim que não levam a lugar nenhum e assim consigo pensar só amenidades. Agradeço também os companheiros do meu happy hour sagrado de toda sexta, sem qual o qual a vida fica muito sóbria. Finalmente, agradeço os companheiros de pesquisa, mais fortemente Henrique Teófilo, bons tempos do laboratório OW2, e Marcelo e Júlio, por serem os felizes primeiros usuários do Cloud Crawler.

Saravá!

Resumo da Dissertação apresentada ao Programa de Pós-Graduação em Informática Aplicada (PPGIA)/UNIFOR como parte dos requisitos necessários para a obtenção do grau de Mestre em Informática Aplicada.

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

Matheus Ciríaco Cerqueira Cunha

Março/2012

Orientador: Prof. Dr. Nabor das Chagas Mendonça

Co-Orientador: Prof. Dr. Américo Tadeu Falcone Sampaio

Programa: Informática Aplicada

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 (em particular, máquinas virtuais) necessários às suas aplicações. Embora a maioria das plataformas de nuvem IaaS atuais ofereça formas rápidas de adquirir ou liberar recursos, é importante que os clientes tenham um entendimento prévio do desempenho esperado das aplicações quando executadas utilizando diferentes perfis de recursos. Este trabalho apresenta um ambiente programável, chamado *Cloud Crawler*, que automatiza a tarefa de executar e avaliar o desempenho de aplicações em nuvens IaaS, considerando diferentes perfis de recursos virtuais e diferentes níveis de demanda. O ambiente inclui uma nova linguagem declarativa, chamada *Crawl*, para especificação dos cenários de avaliação da aplicação na nuvem, e um motor de execução, chamado *Crawler*, que executa os cenários especificados em *Crawl*, gerando, como resultado, um mapa do desempenho da aplicação para os diferentes perfis de recursos e diferentes níveis de demanda avaliados. O uso do ambiente proposto é ilustrado através da avaliação do desempenho de uma aplicação de rede social em duas plataformas de nuvem comerciais.

Abstract of the Dissertation presented to PPGIA/UNIFOR as a partial fulfillment of the requirements for the degree of Master of Applied Informatics.

A Programmable Environment for Evaluating Application Performance in IaaS Clouds

Matheus Ciríaco Cerqueira Cunha

March/2012

Adviser: Prof. Dr. Nabor das Chagas Mendonça

Co-Adviser: Prof. Dr. Américo Tadeu Falcone Sampaio

Program: Applied Informatics

One of the main challenges faced by current users of infrastructure-as-a-service (IaaS) clouds are the difficulties to estimate cloud resources (particularly virtual machines) according to their application needs. Even though most IaaS cloud platforms provide fast ways to acquire or release resources, it is important to have a prior understanding of the expected application performance when executed using different cloud resource profiles. This work presents a programmable environment, called *Cloud Crawler*, which automates the task of executing and evaluating the performance of IaaS cloud applications considering different resource profiles and different demand levels. The environment includes a new declarative language, called *Crawl*, to specify the cloud application evaluation scenarios, and an execution engine, called *Crawler*, which executes the scenarios specified in *Crawl* and generates, as a result, a map of the application performance for the different resource profiles and demand levels evaluated. The use of the proposed environment is illustrated through an evaluation of the performance of a social network application in two commercial clouds.

Sumário

Lista de Figuras.....	
------------------------------	--

Lista de Tabelas	
-------------------------------	--

1	Introdução.....	1
1.1	Motivação	1
1.2	Objetivos.....	5
1.2.1	Objetivo geral	5
1.2.2	Objetivos específicos	5
1.3	Contribuições	6
1.4	Estrutura da dissertação.....	7
2	Computação em Nuvem.....	8
2.1	Histórico	8
2.2	Conceitos Básicos.....	10
2.2.1	Definições	10
2.2.2	Características Essenciais.....	11
2.2.3	Modelos de Implantação	12
2.2.4	Tipos de Serviço de Nuvem	13
2.3	Virtualização	15
2.3.1	Tecnologias de Virtualização	17
2.3.1.1	Xen	17
2.3.1.2	KVM.....	18

2.3.1.3	VMware	18
2.3.2	Gerenciadores de Infraestrutura de Virtualização	18
2.3.2.1	OpenNebula	19
2.3.2.2	OpenStack	20
2.4	Provedores de Nuvens IaaS Comerciais	21
2.4.1	<i>Amazon</i>	21
2.4.2	<i>Rackspace</i>	23
2.5	Resumo do Capítulo	24
3	Avaliação de Desempenho na Nuvem	25
3.1	Avaliação de Desempenho	25
3.2	Avaliação de Desempenho em Nuvens IaaS	26
3.2.1	Avaliação de Serviços e Recursos da Nuvem	27
3.2.1.1	bitcurrent	27
3.2.1.2	CloudSleuth	28
3.2.1.3	CloudHarmony	28
3.2.2	Avaliação de Aplicações na Nuvem	29
3.2.2.1	Jayasinghe <i>et al.</i>	29
3.2.2.2	Ostermann <i>et al.</i>	30
3.2.2.3	<i>Cloudstone</i>	30
3.2.2.4	<i>CloudXplor</i>	31
3.2.3	Ferramentas de Apoio às Avaliações	31
3.2.3.1	<i>CloudCmp</i>	32
3.2.3.2	<i>Expertus</i>	32
3.3	Resumo do Capítulo	33
4	O Ambiente <i>Cloud Crawler</i>	34

4.1	Visão Geral do Ambiente	34
4.2	A Linguagem <i>Crawl</i>	36
4.2.1	Modelo Conceitual	38
4.2.2	Ilustrando as Entidades do Modelo	40
4.2.3	Representando Entidades de <i>Crawl</i> em <i>Yaml</i>	43
4.2.4	Estendendo <i>Yaml</i>	46
4.2.4.1	Inclusão de Arquivos Externos	47
4.2.4.2	Definição de Laços de Iteração	49
4.3	O Motor <i>Crawler</i>	50
4.3.1	Arquitetura	51
4.3.1.1	Módulo <i>Engine</i>	51
4.3.1.2	Módulo <i>Parser</i>	52
4.3.1.3	Módulo <i>Builder</i>	53
4.3.1.4	Módulo <i>Executer</i>	54
4.3.2	Implementação de um Protótipo	55
4.3.2.1	Implementação do Módulo <i>Engine</i>	56
4.3.2.2	Implementação do Módulo <i>Parser</i>	56
4.3.2.3	Implementação do Módulo <i>Builder</i>	58
4.3.2.4	Implementação do Módulo <i>Executer</i>	59
4.4	Resumo do Capítulo	60
5	Experimentos	61
5.1	Aplicação Alvo	61
5.1.1	Arquitetura da Aplicação.....	62
5.1.2	Gerador de Carga	63
5.2	Metodologia	63
5.2.1	Objetivos e Hipóteses	63

5.2.2	Provedores de Nuvem Escolhidos	64
5.2.3	Descrição dos Cenários de Avaliação	64
5.2.4	Representação dos Cenários em <i>Crawl</i>	67
5.3	Resultados.....	69
5.3.1	Resultados Obtidos na Nuvem <i>Amazon EC2</i>	70
5.3.2	Resultados Obtidos na Nuvem <i>Rackspace</i>	72
5.3.3	Discussão	74
5.4	Resumo do Capítulo	76
6	Conclusão.....	78
6.1	Contribuições e Resultados.....	78
6.2	Sugestões para Trabalhos Futuros	79
	Referências	81

Lista de Figuras

2.1	Modelos de instalação da camada de virtualização.	16
2.2	Ilustração da arquitetura do OpenNebula (extraído de (OPENNEBULA, 2012)).	19
2.3	Ilustração da arquitetura do OpenStack (extraído de (OPENSTACK, 2012a)).	21
4.1	Ilustração de um exemplo de utilização do ambiente <i>Cloud Crawler</i>	35
4.2	Modelo conceitual da linguagem <i>Crawl</i>	38
4.3	Componentes da aplicação wiki mapeados para entidades da linguagem <i>Crawl</i>	40
4.4	Representação em <i>Crawl</i> dos componentes da aplicação wiki.	41
4.5	Representação em <i>Crawl</i> de um cenário de avaliação para a aplicação wiki.	42
4.6	Representação textual de duas entidades do tipo <i>Provider</i>	45
4.7	Representação textual de uma entidade do tipo <i>Benchmark</i>	46
4.8	Representação textual do cenário de avaliação definido para a aplicação wiki.	47
4.9	Representação textual do cenário de avaliação definido para a aplicação wiki, com a utilização do recurso de inclusão de arquivos externos.	48
4.10	Especificação dos perfis de recurso de um provedor de nuvem utilizando a sintaxe original de <i>Yaml</i>	49
4.11	Especificação dos perfis de recurso de um provedor de nuvem utilizando um laço de iteração.	50
4.12	Arquitetura do motor <i>Crawler</i>	52
4.13	Fluxo de inicialização das máquinas virtuais executado pelo módulo <i>Builder</i> .	54
4.14	Arquitetura do protótipo implementado para o motor <i>Crawler</i>	55
4.15	Diagrama de classe representando as entidades de avaliação de desempenho utilizadas internamente pelo motor <i>Crawler</i>	57

4.16	Diagrama de sequência ilustrando o procedimento de preparação dos recursos da nuvem implementado pelo módulo <i>Builder</i>	59
5.1	Arquitetura da aplicação <i>Olio</i> implementada na plataforma <i>Ruby on Rails</i> (extraído de (SOBEL <i>et al.</i> , 2008)).	62
5.2	Exemplo de implantação dos componente da aplicação <i>Olio</i> na nuvem.	66
5.3	Especificação em <i>Crawl</i> descrevendo os cenários de avaliação da aplicação <i>Olio</i> na nuvem.	68
5.4	Comparação dos custos mínimos por demanda atendida observados para a aplicação <i>Olio</i> nas nuvens <i>Amazon EC2</i> e <i>Rackspace</i>	76

Lista de Tabelas

1.1	Zonas de disponibilidade e quantidades de perfis de recursos computacionais oferecidos por sete provedores de nuvem IaaS comerciais.	3
2.1	Tipos de instâncias oferecidos pela Amazon EC2 na região <i>US East</i> e suas respectivas configurações e preços por hora de uso.	23
2.2	Tipos de máquinas virtuais oferecidos pelo Rackspace <i>Cloud Servers</i> e suas respectivas configurações e preços por hora de uso.	24
5.1	Resultados da avaliação de desempenho da aplicação <i>Olio</i> na nuvem <i>Amazon EC2</i>	70
5.2	Resultados da avaliação de desempenho da aplicação <i>Olio</i> na nuvem <i>Rackspace</i>	72
5.3	Configurações recomendadas para os provedores <i>Amazon EC2</i> e <i>Rackspace</i> capazes de hospedar os componentes das camadas web e de aplicação da <i>Olio</i> em cada nível de demanda avaliado.	75

Capítulo 1

Introdução

Este capítulo apresenta as principais questões que motivaram a realização deste trabalho, seus objetivos e contribuições, e sua organização.

1.1 Motivação

A computação em nuvem (do Inglês *cloud computing*) vem cada vez mais sendo considerada como uma alternativa viável à aquisição e à manutenção de uma infraestrutura computacional própria por parte das organizações (ARMBRUST *et al.*, 2009, 2010). Uma das razões da sua crescente popularidade é o seu atrativo modelo de negócios, baseado na oferta de recursos computacionais (como máquinas virtuais, espaço de armazenamento, plataformas de desenvolvimento e aplicações) na forma de serviços que podem ser utilizados e, no caso de provedores de nuvem comerciais, pagos sob demanda (ARMBRUST *et al.*, 2009; RHOTON, 2009; REESE, 2009).

Atualmente, já há diversas soluções de computação em nuvem disponíveis. Estas variam desde soluções de código aberto, como OpenNebula (OPENNEBULA, 2012), Eucalyptus (EUCALYPTUS, 2012) e OpenStack (OPENSTACK, 2012b), que permitem configurar e gerenciar um ambiente de nuvem dentro da infraestrutura computacional da própria organização, até provedores de nuvem públicos, como Amazon EC2 (EC2, 2012), Rackspace (RACKSPACE, 2012), Google AppEngine (APPENGINE, 2012), Microsoft Windows Azure (AZURE, 2012) e Salesforce (SALESFORCE, 2012), que comercializam uma ampla gama de recursos computacionais virtualizados acessíveis através da Internet.

Dependendo do modelo de serviço oferecido, as soluções de computação em nuvem podem ser classificadas em três tipos (ARMBRUST *et al.*, 2009): infraestrutura como serviço (*infrastructure as a service – IaaS*), que oferecem recursos de infraestrutura como máquinas

virtuais, armazenamento e rede; plataforma como serviço (*platform as a service – PaaS*), que oferecem uma plataforma completa para o desenvolvimento e execução de aplicações na nuvem; e software como serviço (*software as a service – SaaS*), que oferecem aplicações completas que podem ser facilmente configuradas e utilizadas pelos clientes da nuvem.

Nuvens do tipo IaaS (também referenciadas neste trabalho como nuvens de infraestrutura), em particular, têm sido as mais utilizadas até o momento, o que tem elevado o número de provedores oferecendo esse tipo específico de serviço de nuvem.¹ Essa preferência por nuvens IaaS se dá principalmente pelo fato desse tipo de serviço facilitar a aquisição e o gerenciamento de infraestrutura computacional por parte do cliente da nuvem, sem praticamente nenhuma restrição de natureza tecnológica quanto ao tipo de aplicação que o cliente pode disponibilizar nessa infraestrutura. Dessa forma, o cliente da nuvem não precisa investir muito tempo nem esforço na manutenção de recursos físicos, podendo, assim, se dedicar mais àquilo que lhe é mais importante, ou seja, o seu negócio e as suas aplicações.

Por outro lado, a grande oferta de serviços e de provedores de nuvem IaaS traz à tona um novo desafio aos seus potenciais clientes. Como cada provedor de nuvem IaaS disponibiliza uma variedade de recursos computacionais, com diversas opções de configuração e modelos de precificação, selecionar os tipos de recursos ou mesmo os provedores de nuvem que ofereçam a melhor relação custo/benefício para uma determinada aplicação de uma dada organização passa a ser uma tarefa não trivial (LI *et al.*, 2010). Para dar uma ideia das opções oferecidas pelo mercado, a Tabela 1.1 lista a quantidade de zonas de disponibilidade² e de tipos (ou perfis) de recursos computacionais oferecidos por sete provedores de nuvens IaaS comerciais. No caso do provedor Amazon EC2, por exemplo, são oferecidos mais de 10 perfis diferentes de máquinas virtuais, as quais podem estar fisicamente alocadas em uma de sete zonas de disponibilidade espalhadas por quatro continentes (incluindo uma no Brasil). Se for considerada a possibilidade de uma mesma aplicação ser implantada parcialmente em múltiplos provedores (por exemplo, o servidor web em um provedor e o servidor de banco de dados em outro), o processo de seleção dos recursos da nuvem por parte da organização seria ainda mais complexo.

A escolha dos perfis de recursos e do provedor de nuvem mais vantajosos para um determinado cliente requer que o cliente conheça bem não apenas as necessidades e demandas específicas de sua aplicação, como também as características (em termos de desempenho,

¹O sítio cloudorado.com, que oferece um serviço de busca e comparação de provedores de nuvem IaaS, lista mais de 15 provedores comerciais espalhados por diversas regiões do planeta.

²Zonas de disponibilidade correspondem às regiões geográficas (normalmente indicadas pelo nome de um país ou cidade) onde os recursos da nuvem são fisicamente alocados.

Provedor	Zonas de Disponibilidade	# Perfis de Recursos
Amazon EC2	US East (Northern Virginia), US West (Oregon), US West (Northern California), EU (Ireland), Asia Pacific (Singapore), Asia Pacific (Tokyo), South America (São Paulo), AWS GovCloud	11
BitRefinery	US Central (Colorado)	Configurável
OpSource	US East (Virginia), US West (California), EU (Netherlands), Asia Pacific (Australia)	Configurável
Rackspace	US (Grapevine), US (Richardson), US (Chicago), US (Herndon), US (Ashburn), EU (London), EU (Slough), Asia Pacific (Hong Kong)	8
Softlayer	US (Dallas), US (Seattle), US (Washington), US (Houston), US (San Jose), EU (Amsterdam), Asia Pacific (Singapore)	13
GoGrid	US West (California), US East (Virginia)	7
Hosting.com	US (Dallas), US (Denver), US (Irvine), US (Louisville), US (Newark), US (San Francisco)	Configurável

Tabela 1.1: Zonas de disponibilidade e quantidades de perfis de recursos computacionais oferecidos por sete provedores de nuvem IaaS comerciais.

preço e localização geográfica) dos recursos oferecidos por cada provedor. Por exemplo, recursos disponibilizados em regiões geograficamente distantes da localização física dos clientes (usuários da nuvem) podem incorrer em tempos de resposta maiores devido à maior latência da rede.

A falta de um planejamento adequado dos recursos a serem adquiridos da nuvem pode acarretar duas situações de riscos indesejáveis para o cliente, nas quais a sua aplicação poderia ser implantada na nuvem com recursos além ou aquém do necessário. Ambas situações podem resultar em prejuízos para o cliente da nuvem, embora por motivos distintos. Na primeira, o cliente pagará por recursos que não irá utilizar; já na segunda, a aplicação terá menos recursos do que de fato necessita, o que poderá afetar a qualidade de seus serviços e, conseqüentemente, reduzir a satisfação de seus usuários (ARMBRUST *et al.*, 2009).

As dificuldades de provisionar adequadamente os recursos da nuvem não estão restritas ao momento da implantação da aplicação, e podem ocorrer durante a sua execução. Por exemplo, diante da variação da demanda pela aplicação, o cliente pode ter que alterar o perfil dos recursos atualmente alocados, seja aumentando ou diminuindo o seu poder computacional. Nesse caso, também é importante que o cliente conheça bem a relação entre os diferentes perfis de recursos oferecidos pelo provedor da nuvem e o seu potencial

impacto (negativo ou positivo) no desempenho da aplicação.

O problema do planejamento da capacidade dos recursos da nuvem de acordo com as necessidades e demandas específicas da aplicação, em virtude do grau de dificuldade envolvido e a sua importância estratégica para as organizações, tem sido alvo de diversos trabalhos de pesquisa nos últimos anos (SOBEL *et al.*, 2008; OSTERMANN *et al.*, 2010; MALKOWSKI *et al.*, 2010; LI *et al.*, 2010; JAYASINGHE *et al.*, 2011, 2012; CLOUDHARMONY, 2012). Porém, a maioria desses trabalhos tem focado na execução, através de procedimentos predominantemente manuais, de *benchmarks* conhecidos da área de sistemas distribuídos em diferentes ambientes de computação em nuvem, visando entender o impacto de diferentes tipos de recursos de nuvem no desempenho das aplicações que fazem parte desses *benchmarks* (SOBEL *et al.*, 2008; OSTERMANN *et al.*, 2010; MALKOWSKI *et al.*, 2010; LI *et al.*, 2010; JAYASINGHE *et al.*, 2011; CLOUDHARMONY, 2012). Pouca atenção tem sido dada até o momento ao problema de como facilitar a definição e automatização de diferentes cenários de avaliação de desempenho de aplicações na nuvem, com destaque para o trabalho descrito em (JAYASINGHE *et al.*, 2012), que propõe um ambiente para execução automática de testes de desempenho na nuvem. No entanto, a execução de testes de desempenho nesse ambiente é baseada na geração e customização de *scripts shell*, uma linguagem de comandos comumente utilizada para a execução de programas e procedimentos de configuração em computadores remotos, e que requer um alto grau de conhecimento sobre recursos básicos de infraestrutura para ser adequadamente utilizada.

Diante desse contexto, uma solução automatizada para avaliar o desempenho de aplicações na nuvem, que ofereça uma notação de alto nível para a definição e execução de diferentes cenários de avaliação, seria uma importante contribuição para apoiar os clientes de nuvens IaaS no planejamento e alocação dos recursos da nuvem às suas aplicações. Tal solução poderia ser de grande valia, por exemplo, para ajudar os clientes da nuvem a responderem de forma mais precisa perguntas tipicamente relacionadas à implantação de aplicações na nuvem, como:

- Qual provedor de nuvem ofereceria o melhor custo/desempenho para uma determinada aplicação?
- Quais perfis de recursos oferecidos por um determinado provedor de nuvem ofereceriam o melhor custo/desempenho para a aplicação diante de uma demanda específica (por exemplo, até 100 usuários concorrentes)?
- Até que ponto as variações entre os preços dos recursos oferecidos pelo provedor de

nuvem seriam proporcionais às variações nos níveis de desempenho que cada um desses recursos ofereceria à aplicação (por exemplo, uma recurso com o dobro do preço de outro ofereceria um desempenho duas vezes melhor)?

- No caso de recursos da mesma faixa de preço mas com perfis diferentes (por exemplo, em termos de processamento e/ou memória), qual dos recursos ofereceria o melhor desempenho à aplicação?
- Qual seria a melhor estratégia de alocação de recursos da nuvem para a aplicação: implantar cada componente em uma única máquina virtual de maior porte (e, portanto, mais cara) ou distribuí-los em múltiplas máquinas virtuais de menor capacidade (e, portanto, mais baratas)?

1.2 Objetivos

1.2.1 Objetivo geral

Este trabalho tem como principal objetivo investigar, propor e validar empiricamente soluções automatizadas para apoiar a avaliação do desempenho de aplicações em nuvens de infraestrutura. Tais soluções devem permitir a definição de cenários de avaliação utilizando uma linguagem declarativa de alto nível, com cada cenário podendo considerar diferentes provedores, perfis de recursos e níveis de demanda.

1.2.2 Objetivos específicos

Os objetivos específicos do trabalho são:

- Propor uma linguagem declarativa para a definição de cenários de avaliação do desempenho de aplicações em nuvens de infraestrutura;
- Propor um ambiente para a execução automática dos cenários de avaliação definidos na linguagem proposta no item anterior;
- Implementar uma prova de conceito, na forma de protótipos, da linguagem e do ambiente de execução propostos;
- Ilustrar a utilidade dos protótipos implementados através de exemplos concretos envolvendo a avaliação do desempenho de uma aplicação real em diferentes provedores de nuvem IaaS;

1.3 Contribuições

Para atingir os objetivos descritos acima, inicialmente foi feito um levantamento detalhado de ferramentas para avaliação do desempenho de aplicações na nuvem. Uma dessas ferramentas (SOBEL *et al.*, 2008) foi selecionada para a realização de diversos experimentos, visando obter um entendimento inicial do impacto de diferentes perfis de recursos de nuvem no desempenho de uma aplicação de rede social de código aberto (CUNHA *et al.*, 2011a, 2011b).

As dificuldades encontradas durante a realização desses experimentos, executados manualmente, motivaram a investigação de uma solução automatizada e de fácil utilização para a execução de testes de desempenho de aplicações na nuvem, culminando com as seguintes contribuições:

Crawl — uma linguagem declarativa de alto nível, através da qual os clientes da nuvem podem especificar os cenários de avaliação do desempenho de suas aplicações utilizando recursos de nuvens de infraestrutura. *Crawl* permite a especificação de cenários de forma modular, promovendo e facilitando o reuso de especificações comuns a diferentes cenários, aplicações e provedores;

Crawler — um motor que automatiza a execução dos cenários de avaliação especificados em *Crawl*. O *Crawler* é responsável por todas as atividades necessárias para a realização das avaliações de desempenho, incluindo a configuração dos recursos da nuvem, o gerenciamento das execuções dos cenários, e a geração e coleta dos resultados das avaliações.

Juntos, a linguagem *Crawl* e o motor *Crawler* — coletivamente chamados de ambiente *Cloud Crawler* — constituem uma poderosa ferramenta com a qual é possível especificar e executar, de forma totalmente automatizada, uma grande variedade de cenários de avaliação, considerando diferentes tipos de recursos, provedores de nuvem e parâmetros de configuração das aplicações. Dessa forma, o ambiente proposto permite ao cliente da nuvem expressar e avaliar diversas estratégias de implantação de aplicações na nuvem, facilitando a identificação daquelas de melhor custo/benefício. Além disso, abordagem declarativa utilizada para definição de cenários no ambiente tem como importante benefício a redução dos esforços de programação e configuração de recursos por parte do cliente da nuvem, especialmente quando comparados aos esforços tipicamente exigidos quando experimentos dessa natureza são executados manualmente ou através da customização de linguagens de comando de baixo nível de abstração.

Como prova de conceito do ambiente proposto, a linguagem *Crawl* e o motor *Crawler* foram inicialmente implementados na forma de um protótipo. Este protótipo foi então utilizado para automatizar a execução de diversos testes de desempenho em dois provedores de nuvem IaaS comerciais, com o objetivo de investigar a relação entre o custo e o desempenho oferecidos por esse dois provedores no contexto específico de uma aplicação de rede social de código aberto (SOBEL *et al.*, 2008). Os resultados obtidos indicam que há diferenças significativas (tanto em termos de custo quanto de desempenho) não apenas entre os dois provedores, mas também entre os diferentes perfis de recursos oferecidos por cada provedor.

Esses resultados reforçam a importância de se oferecer soluções automatizadas e facilmente customizáveis, como é o caso do ambiente *Cloud Crawler*, que permitam apoiar os clientes da nuvem no planejamento e na avaliação dos recursos computacionais a serem alocados às suas aplicações.

1.4 Estrutura da dissertação

Além desta Introdução, esta dissertação está organizada em mais cinco capítulos, descritos brevemente abaixo.

O Capítulo 2 revisa alguns conceitos e definições relevantes para o entendimento do paradigma da computação em nuvem.

O Capítulo 3 cobre alguns trabalhos relacionados à avaliação de desempenho na nuvem, discutindo as suas principais características e limitações.

O Capítulo 4 descreve o ambiente *Cloud Crawler* em mais detalhes, incluindo sua arquitetura, funcionalidades e aspectos de sua implementação na forma de um protótipo.

O Capítulo 5 apresenta a metodologia e os resultados de alguns experimentos realizados com o objetivo de ilustrar o uso e o funcionamento do ambiente *Cloud Crawler* em diversos cenários de avaliação de desempenho, utilizando provedores de nuvem comerciais.

Finalmente, o Capítulo 6 conclui o trabalho com uma discussão de suas principais contribuições e algumas sugestões para pesquisas futuras.

Capítulo 2

Computação em Nuvem

Este capítulo apresenta uma visão geral do paradigma da computação em nuvem, iniciando com um breve histórico e em seguida abordando algumas definições, conceitos e tecnologias relevantes para o seu entendimento.

2.1 Histórico

Segundo Rhoton (2009), não é possível apontar um evento ou tecnologia isolada que tenha contribuído para a viabilização da computação em nuvem. Na verdade, seu surgimento é decorrente de uma série de melhorias incrementais em diversas áreas, tais como pagamento flexível (por exemplo, via cartão de crédito), plataformas de virtualização, acesso à Internet de banda larga e arquiteturas orientadas a serviços. Combinadas, essas tecnologias viabilizaram o novo modelo de negócio que constitui a base do sucesso da computação em nuvem (RHOTON, 2009).

Do ponto de vista comercial, a computação em nuvem teve início em 2006, quando a Amazon lança o serviço *Amazon Web Services* (AWS), tornando-se o primeiro provedor massivo de serviços baseados no conceito de nuvem computacional (RHOTON, 2009).

A Amazon iniciou suas atividades em 1995, como uma livraria virtual. Posteriormente, a empresa diversificou o seu negócio com a comercialização de outros produtos como CDs, DVDs, computadores, periféricos, jóias e até automóveis. Seu serviço de vendas online se expandiu rapidamente, o que levou a empresa a alterar seu modelo de negócio, passando a anunciar produtos de empresas parceiras e a intermediar as transações entre essas empresas e seus clientes.

Enquanto o volume de transações no seu portal online crescia, a Amazon investia na modernização da sua infraestrutura computacional, focando na alta disponibilidade dos seus

serviços e na segurança dos dados e das transações. Esse investimento foi feito através da aquisição de novos equipamentos, na distribuição geográfica dos seus centros de processamento de dados (*data centers*), e na melhoria do nível de segurança de suas transações financeiras e dos dados de seus parceiros e clientes. Essa infraestrutura computacional era robusta o suficiente para atender a demanda atípica de períodos de altíssimo pico, como no Natal e em outras datas festivas, garantindo a qualidade do serviço aos usuários do portal. No entanto, durante o restante do ano, grande parte dessa infraestrutura computacional ficava ociosa, uma vez que a demanda era muito inferior a dos períodos de pico.

Nesse contexto, quando a Amazon lança o seu serviço de computação em nuvem ela está de fato disponibilizando ao mercado parte da sua capacidade computacional ociosa. Tal modelo de negócio mostrou-se uma alternativa atraente para um número grande de clientes, que tinham à sua disposição uma das infraestruturas computacionais mais seguras e robustas do mercado, a um custo significativamente reduzido comparado à aquisição de uma infraestrutura própria de capacidade equivalente. Nesse sentido, a computação em nuvem pode ser vista como a concretização de um sonho antigo de oferecer TI como serviço de utilidade (PARKHILL, 1966), permitindo que desenvolvedores ou empresas disponibilizem seus sistemas ao grande público com baixo custo inicial e sem precisar se preocupar com a administração de servidores físicos.

A perspectiva de crescimento do mercado de computação em nuvem, principalmente a partir da crise econômica mundial iniciada em 2009, que afetou os investimento em tecnologia da informação (TI) na maioria das organizações, atraiu a atenção de outros gigantes da indústria, como Google e Microsoft, que logo lançaram seus serviços de nuvem.

Projeções realizadas pelas empresas de consultoria Merrill Lynch, IDC e Gartner indicam que o mercado da computação em nuvem crescerá expressivamente nos próximos anos. Em números, a Gartner aponta que o mercado atual de computação em nuvem representa U\$46.4 bilhões e até 2013 atingirá U\$150.1 bilhões (PRING *et al.*, 2009). Já a IDC estima que o número de servidores disponíveis na nuvem crescerá de 600 mil unidades em 2010 para 1,3 milhão de unidades em 2014, somente em nuvens privadas¹ (BRODERICK *et al.*, 2010).

¹Nuvs privadas são nuvs criadas utilizando a infraestrutura computacional própria das empresas, não sendo compartilhadas com terceiros. Ver seção 2.2.3.

2.2 Conceitos Básicos

2.2.1 Definições

Até o momento, não há uma definição consensual sobre o paradigma da computação em nuvem, e não é objeto deste trabalho propô-la. Assim, nesta seção serão apresentadas e discutidas algumas das definições mais citadas na literatura.

Inicialmente, são apresentadas três definições que compartilham alguns conceitos em comum:

Computação em nuvem é “um paradigma de computação distribuída em larga escala [...], no qual um *pool* de recursos abstratos, virtualizados, escaláveis e gerenciados, na forma de poder computacional, armazenamento, plataformas ou serviços, são entregues sob demanda a consumidores externos através da Internet.” (FOSTER *et al.*, 2009).

“Computação em nuvem é um estilo de computação no qual recursos de TI escaláveis e elásticos são providos ‘como serviço’ para consumidores usando tecnologias de Internet.” (CEARLEY; PHIFER, 2010)

“A computação em nuvem é um modelo computacional que permite o acesso ubíquo, conveniente, sob demanda e através da rede a um *pool* de recursos computacionais configuráveis (redes, servidores virtuais ou físicos, armazenamento, aplicações e serviços) que podem ser rapidamente provisionados e liberados...” (MELL; GRANCE, 2011)

Nas duas primeiras citações é possível observar três palavras chave: **escaláveis** — relacionada à percepção de recursos ilimitados; **serviço** — similar ao fornecimento de serviços utilitários como energia elétrica ou gás, que são pagos apenas quando utilizados; e **Internet** — relativa ao meio através do qual o consumidor terá acesso ao serviço. Já a terceira citação menciona o rápido provisionamento e a liberação de recursos, como forma de garantir escalabilidade, e o acesso aos recursos através da rede, não necessariamente a Internet.

Em (VAQUERO *et al.*, 2008), os autores discutem o paradigma da computação em nuvem através da identificação e estudo de 20 definições disponíveis na literatura, para então extrair uma definição mais abrangente, contendo as características comuns a várias das definições analisadas:

A computação em nuvem “consiste em um grande *pool* de recursos virtualizados (tais como hardware, plataformas de desenvolvimento e/ou serviços) facilmente utilizáveis e acessíveis. Esses recursos podem ser dinamicamente reconfigurados para se ajustarem a uma carga variável, permitindo também a sua utilização ótima. Esse *pool* de recursos é tipicamente explorado em um modelo de pagar-pelo-uso, no qual o provedor de infraestrutura oferece garantias por meio de acordos de nível de serviço (SLAs) customizados.”

Essa última definição menciona o termo **acordo de nível de serviço**, uma característica particularmente importante para aplicações sensíveis a flutuações de desempenho. Outra característica mencionada é o modelo de **pagar-pelo-uso**, o que permite que uma aplicação se adapte a variações na demanda com a adição de novos recursos ou a liberação de recursos existentes, restringindo seus gastos com a nuvem apenas a recursos que de fato necessita e utiliza.

2.2.2 Características Essenciais

Uma das definições discutidas na subseção anterior foi elaborada pelo NIST² (acrônimo para *National Institute of Standards and Technology*), uma agência federal de tecnologia dos EUA. Além dessa definição, o NIST também identifica cinco características consideradas essenciais dos serviços de computação em nuvem (MELL; GRANCE, 2011). São elas:

Autoatendimento sob demanda — um cliente pode provisionar unilateralmente recursos computacionais, como tempo de processamento no servidor e armazenamento na rede, à medida em que os necessite e sem precisar de interação humana com os provedores de cada serviço.

Amplo acesso através da rede — os recursos estão disponíveis através da rede e são acessados por meio de mecanismos padronizados que promovam o uso de plataformas clientes heterogêneas (por exemplo, telefones celulares, laptops e PDAs).

Pool de recursos — os recursos computacionais do provedor são consolidados para servir múltiplos clientes, com diferentes recursos físicos e virtuais dinamicamente atribuídos e reatribuídos de acordo com a demanda. Há um senso de independência de localização no qual o cliente não tem controle ou conhecimento do local exato onde se encontra o recurso, sendo apenas possível um certo nível de abstração da

²<http://www.nist.gov/>

sua localização, como país, estado ou região. Exemplos de recursos incluem o armazenamento, processamento, memória e largura de banda de rede.

Elasticidade rápida — recursos podem ser elasticamente provisionados e liberados, em alguns casos automaticamente, de modo que as aplicações passam a se adaptar à demanda imposta pelo usuário. Para o cliente da nuvem, os recursos disponíveis geralmente aparentam ser ilimitados, podendo ser solicitados em qualquer quantidade e a qualquer tempo.

Serviço medido — sistemas de gerenciamento da nuvem automaticamente controlam e otimizam o uso dos recursos, estabelecendo uma capacidade de medição em algum nível de abstração adequado para o tipo de serviço (por exemplo, armazenamento, processamento, largura de banda, e contas de clientes). O uso dos recursos pode ser monitorado, controlado e reportado de forma transparente para o provedor e os clientes.

Um aspecto importante a se discutir sobre a computação em nuvem diz respeito à maneira como a infraestrutura de nuvem é implantada. Os principais modelos de implantação associados à computação em nuvem são descritos a seguir.

2.2.3 Modelos de Implantação

Dependendo da sua forma de implantação, as nuvens podem ser classificadas em três modelos: nuvem privada, nuvem pública e nuvem híbrida (ARMBRUST *et al.*, 2009; ZHANG *et al.*, 2010).³

A nuvem privada é construída para ser utilizada por uma única organização e geralmente a alocação de seus recursos se dá por uma rede interna. Esse modelo de nuvem oferece todos os benefícios normalmente atribuídos à computação em nuvem, como recursos virtualizados e escalabilidade das aplicações. A exceção é o seu alto custo de manutenção, já que a própria organização é responsável pelo gerenciamento da infraestrutura física. Atualmente há diversas soluções que permitem a criação e o gerenciamento de uma nuvem privada dentro de uma organização, com destaque para várias soluções de código aberto como OpenStack (OPENSTACK, 2012b), Eucalyptus (EUCALYPTUS, 2012) e OpenNebula (OPENNEBULA, 2012).

³A noção de nuvem privada não é consenso na literatura. Por exemplo, alguns autores, como (REISE, 2009), consideram que o conceito de computação em nuvem deve necessariamente envolver uma infraestrutura de nuvem pública.

Nuvens públicas constituem o modelo de nuvem mais conhecido, e disponibilizam recursos de nuvem para o público em geral. Esses recursos normalmente estão disponíveis através da Internet e são gerenciados por uma organização (provedor da nuvem), que pode cobrar pela sua utilização.

Por fim, no modelo de nuvem híbrida, recursos de uma nuvem pública são alocados em conjunto com os de uma nuvem privada. Um exemplo de utilização é no caso dos recursos da nuvem privada terem se exaurido (ex.: em momentos de pico). Neste caso, novos recursos podem ser alocados a partir de uma nuvem pública. O modelo de nuvem híbrida é implementado em algumas soluções como OpenNebula (OPENNEBULA, 2012), que permite a integração de recursos de uma nuvem privada com recursos da nuvem da Amazon, e Righscale (RIGHTSCALE, 2012), que oferece um serviço de nuvem híbrida comercial.

2.2.4 Tipos de Serviço de Nuvem

Outra forma de classificar as soluções de computação em nuvem é quanto aos tipos dos serviços oferecidos. Os três tipos de serviços de nuvem mais comumente citados são: infraestrutura como serviço (IaaS), plataforma como serviço (PaaS), e software como serviço (SaaS) (ARMBRUST *et al.*, 2010; ZHANG *et al.*, 2010).

Serviços do tipo SaaS caracterizam-se por ter como cliente o próprio usuário final do serviço publicado, que normalmente consiste em uma aplicação hospedada na nuvem. Nas nuvens SaaS, é o provedor da nuvem quem se responsabiliza pelo desenvolvimento e atualização dos serviços (aplicações) fornecidos aos usuários.

Serviços do tipo SaaS estão cada vez mais presentes no dia a dia dos usuários de computador. Os exemplos mais comuns são serviços de email, discos virtuais, CRM e editores online, que muitas vezes são ofertados sem custo inicial para seus usuários. No caso do email, um dos serviços mais conhecidos é o GMail, que oferece uma determinada capacidade de armazenamento gratuitamente, com o usuário tendo a opção de pagar para estender essa capacidade inicial caso necessite de mais espaço. Serviços de discos virtuais, como Dropbox e GoogleDrive, também seguem uma política de preços similar. Já serviços de CRM, como [Salesforce.com](https://www.salesforce.com) e o [Sugarcrm.com](https://www.sugarcrm.com), oferecem um período de teste gratuito para o usuário, após o qual o serviço passa a ser pago.

Diferentemente das nuvens SaaS, nas nuvens PaaS o desenvolvimento das aplicações fica sob responsabilidade dos clientes, cabendo ao provedor da nuvem oferecer uma plataforma

de desenvolvimento apropriada ao cliente. Nesse tipo de nuvem, o provedor é responsável por manter o ambiente operacional, garantindo a disponibilidade dos serviços e realizando os ajustes necessários para que as aplicações desenvolvidas pelos clientes atendam a demanda de seus usuários. Dessa forma, os clientes precisam apenas se concentrar na implementação de suas aplicações, sem terem que se preocupar com detalhes de gerenciamento da infraestrutura física subjacente. Essa característica torna as nuvens PaaS bastante interessantes para empresas ou desenvolvedores sem experiências em administração de recursos físicos. Por outro lado, como cada provedor de nuvem PaaS disponibiliza uma plataforma específica, não padronizada, na prática uma aplicação desenvolvida para uma plataforma não pode ser facilmente migrada para outra. Esse problema é conhecido como *vendor lock-in*, uma vez que os clientes dessas plataformas ficam “presos” a um único provedor de nuvem.

Dois dos provedores de nuvem PaaS mais conhecidos são Microsoft e Google, que oferecem os serviços Windows Azure (AZURE, 2012) e Google AppEngine (APPENGINE, 2012), respectivamente. No Windows Azure, o cliente da nuvem está restrito às plataformas .NET ou em Node.js. Já no Google App Engine o desenvolvedor pode escolher entre as linguagens Java, Python e Go (essa última em fase experimental). No entanto, uma aplicação Java ou Python já existente dificilmente poderia ser portada para o Google AppEngine sem sofrer modificações, pois ambas as plataformas possuem restrições em suas APIs quando comparadas às plataformas tradicionais.

Por fim, nuvens do tipo IaaS oferecem aos clientes recursos básicos de infraestrutura, como processamento, armazenamento e rede. Comparado aos outros dois tipos de serviço de nuvem, os serviços do tipo IaaS têm muito mais possibilidades de configuração pelos clientes, que assim podem implantar e executar na nuvem as aplicações de sua escolha, inclusive os próprios sistemas operacionais. Alguns dos provedores de nuvens IaaS mais conhecidos são Amazon (EC2, 2012) e Rackspace (RACKSPACE, 2012).

Este trabalho foca em soluções para apoiar a avaliação do desempenho de aplicações especificamente no contexto de nuvens do tipo IaaS. Por essa razão, o restante do capítulo abordará algumas tecnologias fundamentais para a construção de nuvens que seguem esse modelo, quais sejam, virtualização, gerenciadores de infraestrutura virtual e provedores de nuvens IaaS comerciais.

2.3 Virtualização

O termo virtualização “descreve a separação entre a requisição de um serviço da sua implementação física em hardware” (VMWARE, 2007). A técnica de virtualização é utilizada desde a década de 1960, como forma de particionar os recursos físicos de computadores *mainframes*, visando uma melhor utilização do poder de processamento dessas máquinas (VMWARE, 2007). Na época, os computadores executavam programas que operavam unicamente em modo *batch* (lote), mas que através de técnicas de *time-sharing* podiam ser executados no mesmo computador. No entanto, com o *time-sharing* um programa defeituoso poderia comprometer o desempenho e até mesmo a execução dos demais. Esse problema foi resolvido com a introdução do conceito de partições lógicas, que consistiam em um subconjunto dos recursos computacionais da máquina, cada uma representando uma máquina virtual. Isso permitiu a execução de múltiplas tarefas em partições separadas, com cada partição contendo uma reserva própria dos recursos físicos da máquina.

Com o surgimento e a popularização dos microcomputadores de arquitetura x86, a virtualização perdeu espaço, pois era possível comprar vários desses equipamentos (a preços muito mais baratos que os dos *mainframes*) e executar diferentes programas separadamente em cada um deles. Essa arquitetura acabou se tornando padrão de mercado, fazendo com que hoje grandes corporações tenham que gerenciar centenas, até milhares de servidores x86.

Esse novo panorama trouxe novos desafios para os administradores de sistemas, tais como: baixa utilização da capacidade dos servidores, alto custo de manutenção e gerenciamento, e desperdício de energia. Diante desses desafios, técnicas de virtualização para a arquitetura x86 começaram a ser desenvolvidas, culminando com o lançamento da primeira solução de virtualização comercial para essa arquitetura, em 1998, pela empresa VMware (VMWARE, 2007):

“a solução foi uma combinação de tradução de código binário (*binary translation*) e execução direta no processador hospedeiro, o que permitiu a múltiplos sistemas operacionais convidados serem executados completamente isolados no mesmo computador com baixo *overhead* de virtualização.” (VMWARE, 2007).

Para viabilizar esse tipo de solução de virtualização, uma camada de software deve ser instalada entre o sistema operacional e a máquina física, de modo que o sistema operacional convidado é executado em uma máquina virtual com recursos computacionais próprios,

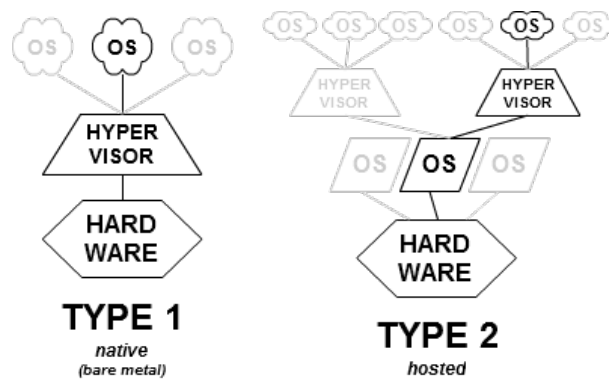


Figura 2.1: Modelos de instalação da camada de virtualização.

que nada mais são do que uma abstração dos recursos da máquina física. A Figura 2.1 ilustra a utilização dessa camada, conhecida como *hypervisor*, nos seus dois modelos de instalação: nativo e hospedado.

No modelo nativo, o *hypervisor* é instalado diretamente na máquina física (*bare metal*), enquanto que no modelo hospedado o *hypervisor* é instalado como uma aplicação do sistema operacional.

Na arquitetura x86, a virtualização pode ser implementada através de três técnicas: **virtualização total**, **paravirtualização** e **virtualização assistida por hardware** (VMWARE, 2007). Na virtualização total, o *hypervisor* intercepta todas as chamadas do sistema operacional convidado ao meio físico, de modo que o sistema convidado não sabe que está sendo virtualizado. Nesse caso, o *hypervisor* precisa oferecer suporte a diversos tipos de dispositivos, o que geralmente implica em disponibilizar *drivers* genéricos, que podem não implementar as funcionalidades mais especializadas de cada tipo de dispositivo. Na paravirtualização, o sistema operacional convidado precisa ser modificado para executar as funções do *hypervisor*, sendo que os *drivers* da própria máquina virtual fazem as chamadas ao meio físico. A principal desvantagem dessa técnica é que ela requer acesso ao código fonte do sistema operacional, o que nem sempre é possível em sistemas operacionais comerciais, como o Windows. Por fim, na virtualização assistida por hardware, o suporte à virtualização é disponibilizado diretamente no processador. Essa técnica é implementada em processadores produzidos por fabricantes como Intel e AMD, que oferecem suporte nativo à virtualização, possibilitando que a execução de determinadas funcionalidades seja automaticamente direcionada para o *hypervisor*, eliminando a necessidade de tradução binária.

Com a virtualização, os recursos de um mesmo servidor físico podem ser compartilhados entre várias máquinas virtuais, permitindo um melhor aproveitamento desses recursos.

Essa consolidação de recursos implica em uma menor necessidade de aquisição de servidores físicos, o que também significa economia na manutenção e no espaço físico para aloca-los. A virtualização também permite facilmente migrar uma máquina virtual de um servidor hospedeiro para outro, sem necessidade de reconfiguração da máquina virtual, o que aumenta a disponibilidade das aplicações que executam na máquina virtual em caso de falha do hospedeiro. Essa migração pode acontecer inclusive com o sistema operacional convidado em funcionamento, técnica conhecida como *live migration* (CLARK *et al.*, 2005).

Outra forma de virtualização é a de **armazenamento** (*storage*) (CLARK, 2005). Nessa forma de virtualização, os dados dos servidores ficam armazenados em um repositório central, que é acessado através de redes de alta velocidade, usando padrões como *fibre channel* (CLARK, 1999). Nesse modelo, os servidores vêm com uma capacidade de armazenamento mínima, suficiente para iniciar o sistema operacional, que fica então responsável por montar os volumes que são armazenados no repositório central. Essa forma de virtualização permite que servidores compartilhem a mesma área de armazenamento, a qual, uma vez próxima do seu limite de capacidade, pode ser expandida de modo transparente para os servidores que a utilizam.

2.3.1 Tecnologias de Virtualização

Atualmente, há várias tecnologias de virtualização disponíveis no mercado. Três das tecnologias mais utilizadas são Xen, KVM e VMware, descritas brevemente a seguir.

2.3.1.1 Xen

O *Xen* (BARHAM *et al.*, 2003) é uma solução de virtualização de código aberto para a arquitetura x86, e que utiliza técnicas de paravirtualização de servidores. O Xen consiste em um *hypervisor* instalado no núcleo de um sistema operacional Linux, conhecido como *Domain 0*, e que tem acesso direto aos recursos do meio físico, ficando responsável por interagir com os sistemas operacionais convidados. Todas as funcionalidades de virtualização do Xen dependem da inicialização do *Domain 0*. Os demais sistemas operacionais visitantes, conhecidos como *Domain U*, não têm acesso direto aos recursos físicos da máquina hospedeira, e também precisam ser modificados — a menos que o processador do computador virtualizado tenha suporte nativo à virtualização.

2.3.1.2 KVM

O KVM (KIVITY *et al.*, 2007) é uma solução de virtualização total para Linux, que funciona em processadores da arquitetura x86 com suporte nativo à virtualização (Intel ou AMD) (KVM, 2012). A solução consiste em um módulo (*kvm.ko*) que provê a infraestrutura principal para virtualização e em módulos específicos para os processadores Intel (*kvm-intel.ko*) ou AMD (*kvm-amd.ko*). Por fazer uso da virtualização assistida por hardware, o KVM não necessita que os sistemas operacionais convidados sejam modificados.

2.3.1.3 VMware

A empresa VMware oferece diversas soluções de virtualização total, que vão desde o *VMware Player*, que é executado como uma aplicação do sistema operacional, até o *VMware vSphere*, que pode ser utilizado para gerenciar todo um centro de processamento de dados (*data center*). As soluções mais simples, como o *VMware Player*, gerenciam apenas os recursos de uma única máquina, e possuem *hipervisor* no modelo hospedado. Já soluções como o *VMware vSphere* podem gerenciar os diversos recursos computacionais de centenas de servidores e equipamentos de armazenamento, e possuem *hipervisor* no modelo nativo, o que significa que sua solução de virtualização é instalada no computador que será virtualizado, sem a necessidade de um sistema operacional hospedeiro. Assim como o KVM, as soluções da VMware fazem uso da virtualização assistida por hardware quando instaladas em equipamentos com suporte a esse tipo de tecnologia. As soluções da VMware possuem *drivers* ou serviços tanto para Windows quanto para Linux, o que permite o acesso dos sistemas operacionais convidados aos recursos físicos mais específicos da máquina hospedeira.

2.3.2 Gerenciadores de Infraestrutura de Virtualização

À medida em que as organizações se familiarizam com as tecnologias de virtualização, e passam a usufruir de seus benefícios, há um aumento natural da demanda pelos serviços de virtualização. Esse aumento se reflete em um maior número de recursos virtualizados na empresa, o que torna a sua administração mais complexa.

Nesse cenário, é importante a utilização de um gerenciador de infraestrutura virtual, responsável pelo ciclo de vida das máquinas virtuais e pelo monitoramento dos recursos computacionais disponíveis. Outra vantagem da utilização de um gerenciador de infraestrutura virtual é que este oferece uma visão homogênea dos recursos virtuais, uma vez

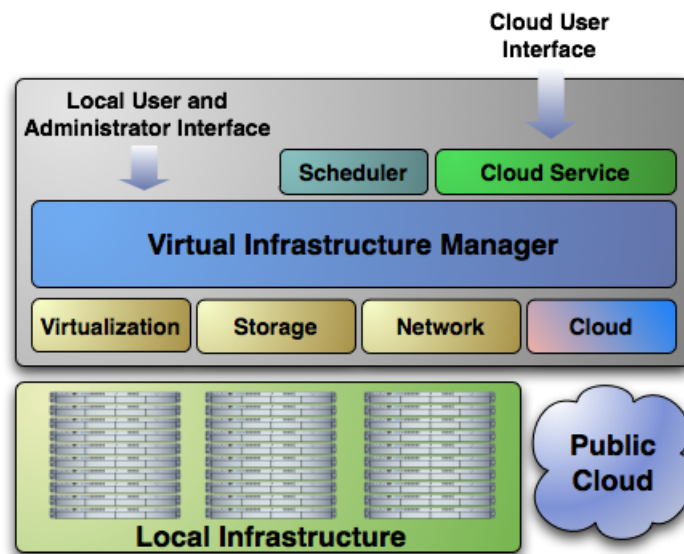


Figura 2.2: Ilustração da arquitetura do OpenNebula (extraído de (OPENNEBULA, 2012)).

que as peculiaridades dos diferentes *hipervisores* ficam transparentes aos usuários. Nesse aspecto, o gerenciador disponibiliza aos usuários um ponto de controle central, através do qual novos recursos são adicionados, máquinas virtuais são criadas, redes virtuais são configuradas e novas imagens são disponibilizadas.

A utilização de um gerenciador de infraestrutura virtual é a base para a construção de uma nuvem do tipo IaaS. Existem vários desses gerenciadores com código aberto, como é o caso de *Eucalyptus* (EUCALYPTUS, 2012), *OpenNebula* (OPENNEBULA, 2012) e *OpenStack* (OPENSTACK, 2012b). Os dois últimos, por serem os mais utilizados atualmente, serão apresentados em mais detalhes nas subseções seguintes.

2.3.2.1 OpenNebula

O OpenNebula é um gerenciador de infraestrutura virtual de código aberto que gerencia os recursos computacionais de cada servidor (ou nó) físico de forma transparente, sendo possível utilizá-lo com diferentes *hipervisores* como Xen, KVM e VMWare. Ele é responsável por controlar a distribuição das máquinas virtuais e suas imagens, bem como por gerenciar a migração destas em caso de falha do nó subjacente ou até mesmo para atender a uma determinada necessidade (por exemplo, para evitar a sobrecarga do nó atual). O OpenNebula também permite separar o tráfego de informações entre as diferentes aplicações hospedadas através da criação de redes virtuais, de modo que, quando uma nova máquina virtual está sendo implantada, é possível configurar a qual rede virtual essa nova máquina estará conectada.

A Figura 2.2 revela detalhes da arquitetura do OpenNebula, a qual é constituída de quatro módulos principais. O módulo de virtualização (*Virtualization*) é responsável por invocar as funções dos *hipervisores*; o módulo de armazenamento (*Storage*) é encarregado da distribuição das imagens das máquinas virtuais; o módulo de rede (*Network*) é responsável pela configuração das redes virtuais; por fim, o módulo de nuvem (*Cloud*) trata da interação com outras plataformas de nuvem. Como pode ser visto na Figura 2.2, a arquitetura do OpenNebula prevê dois tipos distintos de interface de acesso ao gerenciador: uma interface local e uma interface usuário. A interface local permite administrar os recursos computacionais da infraestrutura de servidores, esses recursos consistem nos nós físicos que compõe a nuvem. Além disso, através da interface local é possível gerenciar o ciclo de vida das máquinas virtuais, a configuração das redes virtuais e o gerenciamento das imagens das máquinas virtuais. Estas três funcionalidades são as que estão disponíveis para a interface de usuário, que pode ser acessada remotamente pelos clientes da nuvem, através da Internet. A interface de usuário é baseada em várias tecnologias e padrões de acesso como *EC2 API* (OPENNEBULA, 2012), *OGF OCCI* (OCCI, 2012) e *Java API* (OPENNEBULA, 2012).

2.3.2.2 OpenStack

O OpenStack, assim como o OpenNebula, implementa um gerenciador de infraestrutura virtual de código aberto que permite a criação de nuvens IaaS públicas e privadas, com suporte para diversos *hipervisores* (OPENSTACK, 2012c). O sistema está sendo desenvolvido no âmbito de um projeto iniciado em 2010, fruto de uma parceria entre a NASA e o provedor de nuvem pública Rackspace, e que tem como missão a criação de nuvens simples de implementar e massivamente escaláveis (OPENSTACK, 2012d).

A Figura 2.3 ilustra a arquitetura do *OpenStack*, a qual é constituída de cinco componentes principais:⁴:

- *Dashboard* (“*Horizon*”) — disponibiliza uma interface web modular para todos os serviços do *OpenStack*, através da qual o cliente da nuvem gerencia seus recursos;
- *Object Store* (“*Swift*”) — provê um serviço de armazenamento de objetos, através de qual os clientes podem armazenar ou buscar seus arquivos;
- *Image* (“*Glance*”) — disponibiliza um catálogo e repositório de imagens de discos

⁴<http://docs.openstack.org/trunk/openstack-compute/admin/content/openstack-architecture-overview.html>

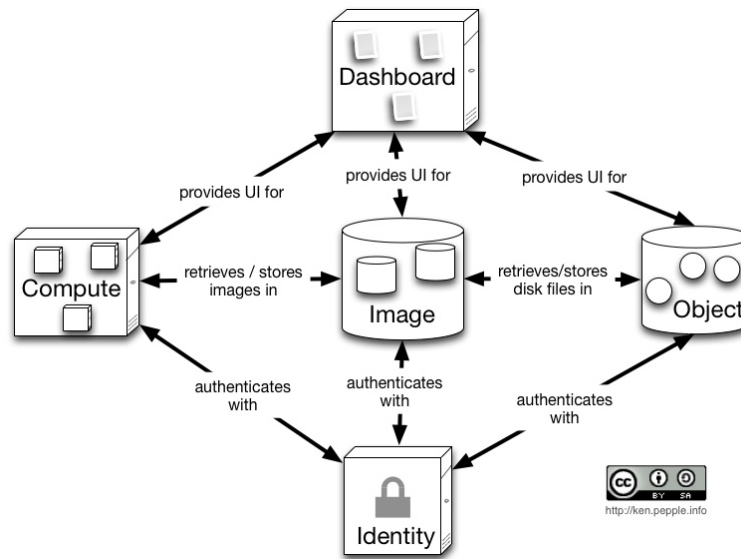


Figura 2.3: Ilustração da arquitetura do OpenStack (extraído de (OPENSTACK, 2012a)).

virtuais, as quais são utilizadas pelas máquinas virtuais executadas na nuvem;

- *Compute* (“*Nova*”) – responsável por disponibilizar máquinas virtuais à medida em que são demandas pelos clientes;
- *Identity* (“*Keystone*”) – provê autenticação e autorização para todos os serviços oferecidos pelo *OpenStack*.

2.4 Provedores de Nuvens IaaS Comerciais

Uma vez expostos os conceitos de virtualização e de gerenciadores de infraestrutura virtual, que constituem a base para a construção de nuvens do modelo IaaS, serão agora apresentadas amostras do leque de recursos e serviços oferecidos por dois provedores de nuvens IaaS comerciais, *Amazon* e *Rackspace*. Estes dois provedores foram escolhidos por estarem entre os mais utilizados atualmente.

2.4.1 *Amazon*

No que diz respeito à variedade de serviços, a *Amazon* é o provedor que mais se destaca, com um amplo catálogo de serviços acessíveis através da web (utilizando interfaces *SOAP* ou *REST*) agrupado em 9 áreas distintas, dentre elas armazenamento, banco de dados, processamento, rede e até aplicações para o usuário final. Alguns desses serviços são descritos brevemente a seguir.

O principal serviço de nuvem da Amazon é o *Elastic Compute Cloud* (EC2), que permite ao cliente obter e configurar recursos computacionais como processamento, memória e armazenamento, na forma de máquinas virtuais que são executadas em máquinas físicas virtualizadas utilizando a tecnologia Xen. Essas máquinas virtuais, referenciadas pela Amazon como “instâncias”, podem executar sistemas operacionais pré-configurados em um formato de imagem proprietário, conhecido como *Amazon Machine Image* (AMI), ou executar sistemas totalmente instalados e configurados pelo cliente da nuvem. Uma vez em execução, o cliente pode acessar suas máquinas virtuais através da Internet usando um cliente SSH de sua preferência.

O cliente do EC2 pode escolher entre oito regiões de disponibilidade para hospedar suas instâncias, as quais podem ser criadas a partir de diversos tipos de instâncias pré-definidos pela Amazon. Cada um desses tipos oferece uma configuração diferente em termos de capacidade de processamento, memória RAM, espaço de armazenamento em disco e desempenho de entrada e saída de dados.

Para armazenar de forma persistente os dados mantidos nos discos que são montados nas máquinas virtuais, o cliente pode fazer uso do serviço *Elastic Block Store* (EBS) (EBS, 2012). O EBS provê um meio de armazenamento de alta disponibilidade, e permite a criação de volumes (discos virtuais) entre 1 GB e 1 TB de tamanho, os quais podem ser livremente montados e desmontados em diferentes instâncias do EC2.

Outro serviço de armazenamento oferecido pela Amazon é o *Simple Storage Service* (S3) (S3, 2012), que permite ao usuário da nuvem criar, acessar e alterar uma quantidade ilimitada de objetos contendo desde 1 *byte* até 5 TB de dados. A segurança no uso do serviço é garantida pelo uso de criptografia tanto na transferência dos dados (via HTTPS), quando no seu armazenamento na nuvem. A tarifação sobre o uso do serviço é feita com base na quantidade de dados armazenados por mês, no número de requisições externas ao serviço e pelo volume de dados transferidos.

A Tabela 2.1 apresenta uma amostra dos tipos de instâncias oferecidos pela Amazon, juntamente com suas configurações e preços por hora de uso para a região *US East* (a mais barata atualmente dentre todas as regiões da Amazon).

Além da tarifação por hora de uso, a Amazon também cobra sobre o volume de dados transferidos entre as instâncias e endereços localizados fora da nuvem, a quantidade de dados armazenados (via EBS), a utilização de endereços IP's fixos (conhecidos como *Elastic IP Addresses*), o monitoramento das instâncias (através do serviço *Amazon CloudWatch*), a utilização de redes privadas virtuais (VPN's) conectando as instâncias

Tipo de Instância	CPU	RAM (GB)	Armazenamento (GB)	Preço
m1.small	1	1,7	160	U\$ 0,085
m1.large	4	7,5	850	U\$ 0,34
m1.xlarge	8	15	1.690	U\$ 0,68
t1.micro	até 2	0,615	–	U\$ 0,02
m2.xlarge	6,5	17,1	420	U\$ 0,50
m2.2xlarge	13	34,2	850	U\$ 1,0
m2.4xlarge	26	68,4	1690	U\$ 2,0
c1.medium	5	1,7	350	U\$ 0,17
c1.xlarge	20	7	1690	U\$ 0,68

Tabela 2.1: Tipos de instâncias oferecidos pela Amazon EC2 na região *US East* e suas respectivas configurações e preços por hora de uso.

na nuvem a máquinas externas (através do serviço *Amazon Virtual Private Cloud*), entre outros serviços, conforme são habilitados.

2.4.2 *Rackspace*

O provedor Rackspace foi apontado pelo *Gartner* como líder no serviço de hospedagem de servidores (CHAMBERLIN; LEONG, 2012), possuindo um portfólio com mais de 10 tipos de serviços de computação em nuvem, a maioria no modelo de infraestrutura como serviço. Dois desses serviços são descritos brevemente a seguir.

Cloud Servers é a solução oferecida pelo Rackspace para disponibilizar máquinas virtuais na nuvem. O serviço provê um catálogo com 8 tipos de máquinas virtuais, chamadas de *flavors*, sendo que cada tipo apresenta uma configuração diferente em termos de processamento, memória RAM e espaço de armazenamento em disco. O acesso às máquinas virtuais pode ser feito via Internet, utilizando um cliente *SSH* ou através de uma console de monitoramento disponibilizada pelo próprio provedor.

A Tabela 2.2 lista as configurações de cada *flavor* do Rackspace, juntamente com seus respectivos preços por hora de uso.

Observa-se, na Tabela 2.2, que cada *flavor* do Rackspace tem o dobro da capacidade de memória e armazenamento bem como do preço do *flavor* imediatamente anterior. A documentação disponibilizada no sítio do Rackspace não especifica a capacidade de processamento de cada *flavor*. Nesse aspecto, a documentação apenas menciona que “todos os *Cloud Servers* vêm com uma certa quantidade de processamento, que é baseada

Flavor	RAM (GB)	Armazenamento (GB)	Preço
1	256	10	U\$ 0,015
2	0,5	20	U\$ 0,03
3	1	40	U\$ 0,06
4	2	80	U\$ 0,12
5	4	160	U\$ 0,24
6	8	320	U\$ 0,48
7	15,5	620	U\$ 0,96
8	30	1.200	U\$ 1,80

Tabela 2.2: Tipos de máquinas virtuais oferecidos pelo Rackspace *Cloud Servers* e suas respectivas configurações e preços por hora de uso.

no tamanho do servidor criado⁵. Ainda segundo a documentação do Rackspace, caso haja processamento disponível no hardware hospedeiro, este é direcionado para os servidores virtuais que estejam em execução.

Como solução de armazenamento, o Rackspace oferece o *Cloud Files*, serviço construído sobre o componente de armazenamento (*Object Store*) do projeto OpenStack. O *Cloud Files* permite ao cliente armazenar e gerenciar uma quantidade ilimitada de arquivos. Esses arquivos são protegidos através do uso de mecanismos de criptografia e de controle de acesso. O modelo de tarifação pelo uso do *Cloud Files* é baseado na quantidade de dados armazenados e no volume de dados transferidos para fora da nuvem.

2.5 Resumo do Capítulo

Este capítulo apresentou o paradigma da computação em nuvem, destacando seu histórico e principais conceitos, modelos e tecnologias.

O próximo capítulo foca na questão da avaliação do desempenho de recursos e aplicações em nuvens IaaS, oferecendo uma revisão de vários trabalhos existentes sobre o tema.

⁵http://www.rackspace.com/cloud/cloud_hosting_products/servers/technology/

Capítulo 3

Avaliação de Desempenho na Nuvem

Este capítulo apresenta uma análise crítica de diversos trabalhos recentes que tratam da avaliação de desempenho de recursos e aplicações na nuvem. Inicialmente, a questão da avaliação de desempenho de sistemas de computação é coberta de forma mais abrangente. Em seguida, trabalhos mais específicos sobre a avaliação de desempenho na nuvem são agrupados e analisados de acordo com a semelhança entre as suas abordagens.

3.1 Avaliação de Desempenho

A avaliação de desempenho auxilia os profissionais da computação a realizar comparações entre diferentes sistemas computacionais, incluindo hardware, software (aplicações) e firmware (JAIN, 2008). No que diz respeito a aplicações, além de possibilitar a realização de comparações, a avaliação de desempenho também auxilia na identificação de alternativas de arquitetura, na identificação de gargalos de desempenho e na determinação dos parâmetros de configuração ideais para a execução do software.

A importância da avaliação de desempenho fica mais evidente quando os recursos avaliados possuem especificações complexas e não padronizadas. Para citar um exemplo, a especificação de um servidor físico *IBM System x3850 X5* possui 15 itens,¹ curiosamente, a mesma quantidade itens da especificação de um equipamento equivalente, o *Dell PowerEdge R815 server*.² Apesar de possuírem a mesma quantidade de itens, as duas especificações são compostas por categorias distintas. Essa falta de padronização na especificação faz com que a comparação entre os recursos, baseada apenas nas especificações, seja uma tarefa complicada. Nesse contexto, a avaliação de desempenho é útil pois permite a realização de comparações entre equipamentos, independentemente das

¹<http://www-03.ibm.com/systems/x/hardware/enterprise/x3850x5/specs.html>

²<http://www.dell.com/us/enterprise/p/powerededge-r815/pd>

suas especificações.

Para realizar uma avaliação de desempenho justa, é preciso garantir que os recursos avaliados estejam sendo tratados sob as mesmas condições. Nesse sentido, organizações como a SPEC (SPEC, 2012b) e a EEMBC (EEMBC, 2012), ambas sem fins lucrativos, foram criadas para definir processos e ferramentas padronizadas para a execução de avaliações de desempenho nos mais diversos ambientes computacionais, envolvendo tanto recursos físicos quanto sistemas de software. A SPEC, por exemplo, disponibiliza *benchmarks* em 11 áreas bem definidas: *CPU*, *Graphics/Workstations*, *MPI/OMP*, Java Cliente/Servidor, servidores de e-mail, sistema de arquivos em rede, *Power*, *SIP*, *SOA*, virtualização e servidores web. Qualquer empresa pode utilizar esses *benchmarks* para avaliar o desempenho do/no seu produto. Após a realização da avaliação, os resultados devem ser submetidos à organização responsável pelo *benchmark*, para que sejam publicados e, no caso da SPEC, disponibilizados na Internet. Dessa forma, é possível comparar o desempenho dos diversos produtos avaliados.

O *benchmark CPU2006* (SPEC, 2012a), por exemplo, permite a comparação entre recursos computacionais físicos. Os servidores das empresas Dell e IBM, mencionados acima, foram ambos submetidos ao CPU2006, e os resultados mostraram que o servidor com o melhor desempenho entre os dois é o da *Dell*.

3.2 Avaliação de Desempenho em Nuvens IaaS

Em uma infraestrutura de computação em nuvem, o escopo da avaliação de desempenho é ampliado para entender o impacto dos serviços da nuvem no desempenho da aplicação. Para isso, a avaliação deve investigar tanto os recursos computacionais da nuvem, quanto o comportamento das aplicações implantadas utilizando esses recursos. Os recursos da nuvem são investigados através da medição de sua capacidade de processamento, memória, largura de banda, latência, armazenamento, e disponibilidade. Enquanto que o comportamento das aplicações que utilizam os recursos da nuvem é avaliado através da medição e da análise de métricas de desempenho relativas aos domínios dessas aplicações. A análise dos resultados da avaliações permite a identificação dos parâmetros ideais de configuração dos componentes das aplicações, a identificação de alternativas de implantação e o planejamento da capacidade dos recursos da nuvem necessários às aplicações.

Os trabalhos citados neste capítulo utilizam diferentes abordagens de avaliação de desempenho em nuvens do modelo IaaS. As principais delas são: medição da disponibili-

dade e latência dos serviços e recursos oferecidos pelos provedores de nuvem; avaliação e comparação do comportamento das aplicações ao serem executadas utilizando tanto recursos da nuvem quanto recursos físicos tradicionais; e avaliação do comportamento das aplicações utilizando diferentes perfis de recursos da nuvem.

3.2.1 Avaliação de Serviços e Recursos da Nuvem

Esta subseção apresenta três trabalhos que avaliam o desempenho de serviços e recursos computacionais oferecidos pelos próprios provedores de nuvem. São eles: *bitcurrent* (CROLL, 2011), *CloudSleuth* (CLOUDSLEUTH, 2012) e *CloudHarmony* (CLOUDHARMONY, 2012).

3.2.1.1 bitcurrent

Este trabalho apresenta os resultados da medição do desempenho e da disponibilidade dos serviços oferecidos por nove provedores de nuvem (CROLL, 2011). A coleta dos resultados foi realizada por um cliente *Javascript* que era executado ao final do carregamento de uma página web, aproveitando o tempo ocioso em que o usuário da página estava lendo o seu conteúdo. Esse *script* recebia uma lista de provedores de nuvem e de, forma randômica, escolhia o provedor que seria avaliado. Para cada provedor, eram enviados dois tipos de requisição: uma com 50 *bytes* e outra com 100 *Kbytes*. A requisição de menor tamanho era enviada duas vezes, com cada envio passando um parâmetro diferente, cujo conteúdo era definido randomicamente, de modo a evitar a utilização de *caches*. Essas requisições eram respondidas por uma aplicação hospedada em cada um dos provedores avaliados, permitindo a coleta de quatro métricas pelo *script* cliente: disponibilidade (o serviço está no ar), *HTTP Connect Time* (tempo necessário para estabelecer a conexão), *HTTP Response Time* (tempo necessário para receber a resposta da requisição) e *Throughput* (taxa de transferência da requisição).

Esse trabalho tem o mérito de conseguir mapear o desempenho de vários provedores de nuvem sob o ponto de vista de usuários espalhados por todo o mundo. Através dele, é possível escolher o melhor provedor para hospedar uma determinada aplicação, em termos das quatro métricas coletadas na época da realização dos experimentos, com base apenas na localização física dos clientes. Por outro, o trabalho não disponibiliza publicamente nenhum *benchmark* ou ferramenta de apoio, de modo que não é possível replicar os testes realizados em outros cenários ou com outros provedores.

3.2.1.2 CloudSleuth

O projeto *CloudSleuth* (CLOUDSLEUTH, 2012) foi motivado pela necessidade de se entender a disponibilidade e o tempo de resposta de diferentes serviços de provedores de nuvens *IaaS* e *PaaS*. Para atingir esse objetivo, o projeto implantou uma aplicação “alvo” em cada provedor de nuvem de interesse. Essa aplicação consiste em um sítio web composto por duas páginas: a primeira possui uma lista com mais de 40 itens e suas respectivas imagens, cada uma com 4 *Kbytes*, enquanto a segunda contém uma imagem com 1,75 MB. Esse sítio web é acessado por agentes implantados em 38 pontos de acesso, com alta capacidade computacional, largura de banda, e geograficamente distribuídos em 22 países participantes. Cada ponto envia continuamente centenas de requisições às aplicações alvo, as quais estão implantadas em aproximadamente 60 provedores de nuvens (provedores com diferentes zonas de disponibilidade foram contabilizados como se cada zona de disponibilidade fosse um provedor distinto).

A grande relevância desse trabalho está no fato dos seus testes estarem sendo executados continuamente, o que permite oferecer uma visão quase em tempo real do desempenho e da disponibilidade dos provedores. Por outro lado, os testes não podem ser modificados nem customizados por terceiros, o que impede a realização de avaliações diferentes daquelas previamente definidas pelos membros do projeto.

3.2.1.3 CloudHarmony

O projeto *CloudHarmony*, cujo objetivo é “tornar-se a principal fonte independente, imparcial e útil de métricas de desempenho de provedores de nuvem” (CLOUDHARMONY, 2012), agrega dados de testes de desempenho realizados desde 2009 em mais de 60 provedores de nuvem. Além do histórico das avaliações, o *CloudHarmony* disponibiliza uma ferramenta para executar novas avaliações de desempenho a qualquer momento, denominada *Cloud Speed Test*,³, a qual permite realizar quatro tipos de teste:

Download a few large files — objetiva determinar o melhor provedor para descarregar arquivos grandes, sendo útil para aplicações como *video streaming*;

Download many small files — objetiva determinar o melhor provedor para descarregar arquivos pequenos, podendo ser útil para hospedar uma página web, por exemplo;

³<http://cloudharmony.com/speedtest>

Upload — útil para avaliar serviços que serão utilizados para envio de arquivos;

Test network latency — a latência afeta o tempo de resposta da aplicação e geralmente está relacionada com a região de onde o teste está partindo.

Os resultados disponibilizados pelo *CloudHarmony* têm como pontos fortes a grande quantidade de dados de testes de desempenho disponíveis, além da possibilidade do cliente da nuvem poder executar novos testes a qualquer tempo. Por outro lado, os testes estão limitados àqueles implementados pela ferramenta de teste, não podendo ser facilmente modificados para contemplar novas métricas ou cenários de avaliação.

3.2.2 Avaliação de Aplicações na Nuvem

Os trabalhos apresentados nesta subseção têm como objetivo entender tanto o comportamento dos recursos e serviços da nuvem (OSTERMANN *et al.*, 2010; JAYASINGHE *et al.*, 2011), quanto o comportamento de aplicações implantadas na nuvem (SOBEL *et al.*, 2008; MALKOWSKI *et al.*, 2010).

3.2.2.1 Jayasinghe *et al.*

Jayasinghe *et al.* (2011) realizaram um estudo das variações no desempenho e na escalabilidade da aplicação RUBBoS⁴ ao ser migrada de um modelo tradicional de *datacenter* para nuvens do modelo IaaS. Os experimentos coletaram dados de desempenho da aplicação — como utilização de CPU e tráfego de rede — nas nuvens *Amazon EC2* e *Open Cirrus*,⁵ bem como no ambiente distribuído *Emulab*.⁶

Durante os experimentos, a RUBBoS foi avaliada em diversos cenários, variando-se a quantidade de máquinas virtuais de uma determinada camada (escalabilidade horizontal) e, no caso das execuções na *Amazon EC2*, variando-se também o perfil de recurso da máquinas virtuais (escalabilidade vertical). Os dados de desempenho coletados durante as execuções no *Emulab* foram utilizados como valores de referência, os quais foram comparados com os dados obtidos nas nuvens *Open Cirrus* e *Amazon EC2*.

Logo no primeiro experimento, os autores tiveram uma surpresa, pois, apesar de estarem usando os melhores recursos na *Amazon EC2*, com relação aos demais serviços avaliados,

⁴RUBBoS implementa um sítio de comércio eletrônico, desenvolvido em camadas e amplamente utilizado em experimentos de avaliação de desempenho de larga escala (OW2, 2012).

⁵<https://opencirrus.org/>

⁶<http://www.emulab.net/>

a quantidade de requisições atendidas por segundo era menor do que no *Emulab* e no *Open Cirrus*. Os autores concluem que o cenário de avaliação que oferece o melhor desempenho no *Emulab* pode oferecer o pior desempenho na *Amazon EC2*, e apontam que para migrar uma aplicação para a nuvem é importante a realização de diversos experimentos que revelem o seu comportamento à medida em que é exposta a variações na demanda e na sua configuração.

3.2.2.2 Ostermann *et al.*

Ostermann *et al.* (2010) realizaram um estudo onde foi avaliado o desempenho de recursos da nuvem *Amazon EC2* para executar aplicações do domínio da computação científica. As avaliações foram realizadas comparando-se os resultados das execuções de *benchmarks* na nuvem, com as execuções desses mesmos *benchmarks* em ambientes físicos tradicionais. Os *benchmarks* utilizados foram: *lmbench*,⁷ *Bonnie*,⁸ *CacheBench*,⁹ e *HPCC*.¹⁰

Os autores concluem que o desempenho dos recursos da nuvem utilizados para os experimentos era baixa, desencorajando que aplicações científicas fossem executadas na nuvem avaliada. Segundo os autores, a razão foi que havia muitos problemas de desempenho de E/S de dados, de memória, além de problemas na execução dos *benchmarks*, que nem sempre conseguiam ser finalizados.

3.2.2.3 Cloudstone

Sobel *et al.* (2008) apresentam o projeto *Cloudstone*, o qual disponibiliza um conjunto de ferramentas para a execução de avaliações de desempenho em nuvens IaaS. O projeto é composto pela aplicação web 2.0, *Olio*, que implementa uma rede social, e pelo *Faban*, ferramenta responsável pela geração de carga e medição de desempenho. A utilização dessas ferramentas permite que o cliente da nuvem, manualmente, execute diversas avaliações de desempenho da aplicação. O projeto relata algumas dessas avaliações, realizadas na nuvem *Amazon EC2*, com as quais foi possível entender o comportamento da aplicação à medida em que é submetida a diferentes níveis de demanda, além da relação custo-demanda para diferentes configurações da aplicação e diferentes perfis de recurso da nuvem (SOBEL *et al.*, 2008) .

⁷<http://www.bitmover.com/lmbench/>

⁸<http://www.textuality.com/bonnie/>

⁹<http://icl.cs.utk.edu/projects/llcbench/cachebench.html>

¹⁰<http://icl.cs.utk.edu/hpcc/>

3.2.2.4 *CloudXplor*

CloudXplor (MALKOWSKI *et al.*, 2010) é uma ferramenta para planejamento de configuração de recursos da nuvem baseada em dados empíricos. A ferramenta foi desenvolvida tomando como base um modelo de planejamento de configuração de recursos de Tecnologia da Informação (TI), com foco explícito em aspectos econômicos. Por essa razão, a ferramenta se utiliza de acordos de nível de serviço baseados na relação do custo da infraestrutura de TI com o valor dos recursos do provedor do serviço. Esse valor será maior quando o tempo de resposta da aplicação for plenamente atendido pelo provedor do serviço, e vai diminuindo à medida em que esse tempo de resposta não é alcançado.

Os dados empíricos precisam ser coletados, previamente, através da execução de diversos experimentos de avaliação de desempenho. Esses dados são compostos por métricas de sistema (uso de CPU, memória utilizada, tráfego na rede e E/S de disco) e métricas de mais alto nível (tempo de resposta e *throughput*). Após a coleta, os dados dos experimentos são submetidos e analisados pela ferramenta, utilizando um de seus quatro módulos: análise de tempo de resposta, análise de *throughput*, análise do valor agregado e do custo, e análise do lucro. Cada um desses módulos filtra os dados, fazendo uso apenas das informações necessárias para a execução da sua análise. Após a análise dos dados, a ferramenta pode ser utilizada para produzir gráficos que ilustram o comportamento da aplicação ao se variar parâmetros como carga de trabalho e configuração dos componentes da aplicação.

Nenhum dos trabalhos descritos nesta subseção propõe nem muito menos disponibiliza ferramentas voltadas especificamente para apoiar a definição e a execução de novos cenários de avaliação de desempenho de aplicações na nuvem, limitando-se a relatar os resultados de avaliações previamente realizadas pelos autores (OSTERMANN *et al.*, 2010; JAYASINGHE *et al.*, 2011; SOBEL *et al.*, 2008), ou a apresentar ferramentas para facilitar a análise e a visualização dos dados coletados nos experimentos *após* a sua realização (MALKOWSKI *et al.*, 2010).

3.2.3 Ferramentas de Apoio às Avaliações

Esta subseção apresenta dois trabalhos que, diferentemente dos trabalhos descritos anteriormente, têm como foco a disponibilização de ferramentas para apoiar, com maior ou menor grau de automação e precisão, avaliações de desempenho de aplicações na nuvem. Essas ferramentas são *CloudCmp* (LI *et al.*, 2010) e *Expertus* (JAYASINGHE *et al.*, 2012), descritas a seguir.

3.2.3.1 *CloudCmp*

Li *et al.* (2010) apresentam uma ferramenta para apoiar a avaliação e a comparação do desempenho e do custo dos recursos e serviços de diversos provedores de nuvem pública, de modo a auxiliar o cliente da nuvem a escolher o provedor mais adequado para a sua aplicação. Essa ferramenta, denominada *CloudCmp*, analisa os serviços de elasticidade, persistência de dados e rede oferecidos pelos provedores de interesse, com base em resultados previamente coletados a partir da execução de diversos *benchmarks*: uma versão modificada do *SPECjvm2008* (SPEC, 2008) para avaliar a característica de elasticidade do provedor; um cliente Java para avaliar os serviços de armazenamento e persistência de dados; e as ferramentas *iperf*¹¹ e *ping* para avaliar os serviços de rede. Após a fase inicial da coleta dos dados, a ferramenta pode ser utilizada para gerar gráficos que auxiliem o cliente da nuvem a comparar o desempenho dos recursos de cada um dos provedores nos quais as avaliações foram realizadas, que assim poderá escolher o provedor e os recursos mais apropriados para as necessidades e demandas específicas de suas aplicações.

Segundo Li *et al.* (2010), até a época do trabalho não houve nenhum provedor de nuvem que se destacasse com relação aos demais. Outra constatação foi de que os resultados obtidos a partir da execução dos *benchmarks* em cada provedor apenas refletiam o momento em que foram coletados, uma vez que a estrutura utilizada pelos provedores para hospedar seus serviços sofre frequentes modificações e a demanda por seus recursos computacionais é bastante variável.

Como essa solução compara serviços e recursos da nuvem através da análise de dados previamente coletados a partir da execução de diferentes *benchmarks*, a escolha dos provedores e recursos mais apropriados para uma determinada aplicação só será eficaz se a aplicação utilizar os recursos da nuvem de forma semelhante à dos *benchmarks* avaliados. Além disso, a ferramenta não oferece suporte à execução de novos cenários de avaliação na nuvem, estando limitada àqueles previamente definidos para os respectivos *benchmarks*.

3.2.3.2 *Expertus*

Jayasinghe *et al.* (2012) apresentam *Expertus*, um *framework* para avaliação de desempenho de aplicações distribuídas implantadas em nuvens IaaS. No *Expertus*, a definição do fluxo de execução das avaliações é feita através da geração automática de artefatos de código em linguagem de comando (*shell scripts*) baseada em *templates*. O uso de

¹¹<http://iperf.sourceforge.net>.

templates permite a fácil incorporação de diferentes provedores e diferentes aplicações no código gerado, evitando, assim, que os artefatos de código tenham que ser modificados diretamente.

Para a realização dos experimentos, é preciso configurar um arquivo *XML*, onde ficarão especificados os parâmetros de avaliação. Esse arquivo passa por vários estágios de transformação, cada um responsável por uma determinada configuração. A quantidade de estágios vai depender do número de componentes da aplicação. Por exemplo, caso a configuração contenha os parâmetros de avaliação de um servidor de aplicação *Apache*, estes serão processadas por um estágio de transformação específico que será responsável por gerar os *scripts* necessários à configuração desse servidor.

De acordo com os relatos e resultados apresentados em (JAYASINGHE *et al.*, 2012), o *Expertus* se mostrou bastante flexível na definição de novos cenários de avaliação, permitindo a reutilização de um grande volume de código em cenários que envolviam um mesmo provedor ou uma mesma aplicação já utilizados em outros cenários previamente definidos. Além disso, o uso de *templates* na geração dos artefatos de código responsáveis pela execução dos cenários ajuda a tornar transparente para o usuário muitos detalhes referentes à configuração e à implantação dos componentes da aplicação na nuvem. Por outro lado, o fato da definição dos cenários de avaliação ser feita na ferramenta através da modificação e customização de *shell scripts*, uma linguagem de comando muito usada por administradores de sistemas, pode dificultar o uso do ambiente por parte de usuários não especialistas.

3.3 Resumo do Capítulo

Este capítulo apresentou uma breve revisão do estado da arte na área de avaliação de desempenho na nuvem. Ao longo do capítulo, diversas abordagens que tratam da avaliação do desempenho de serviços e aplicações em nuvens de infraestrutura foram descritas e analisadas criticamente, em termos de suas vantagens e limitações.

O próximo capítulo apresenta um novo ambiente para avaliação de desempenho de aplicações na nuvem, proposto no âmbito deste trabalho, e que visa resolver algumas das principais limitações identificadas nos trabalhos atualmente existentes sobre o tema.

Capítulo 4

O Ambiente *Cloud Crawler*

Este capítulo apresenta o ambiente *Cloud Crawler*, proposto neste trabalho com o objetivo de apoiar a realização de avaliações de desempenho de aplicações em nuvens de infraestrutura.

4.1 Visão Geral do Ambiente

O ambiente *Cloud Crawler* fornece um conjunto de tecnologias que permitem automatizar a tarefa de executar e avaliar o desempenho de aplicações em nuvens de infraestrutura, considerando diferentes provedores e perfis de recursos computacionais. Fazem parte do ambiente a linguagem declarativa *Crawl* e o motor de execução *Crawler*: a linguagem é utilizada para definir os cenários de avaliação das aplicações na nuvem, enquanto que o motor é responsável por automatizar a execução dos cenários descritos em *Crawl* e pela coleta dos resultados das avaliações.

Crawl oferece um conjunto de elementos sintáticos que permitem ao usuário do ambiente declarar as diversas entidades (componentes de software, parâmetros de configuração, níveis de demanda, métricas de desempenho, tipos de recursos da nuvem, etc.) necessários à execução e avaliação do desempenho da sua aplicação na nuvem. As declarações feitas em *Crawl* servem de insumo para o motor *Crawler*, que configura os componentes da aplicação na nuvem e executa os cenários de avaliação conforme declarado pelo usuário.

A Figura 4.1 ilustra um exemplo de um possível cenário de avaliação de desempenho de uma aplicação em uma nuvem IaaS utilizando o ambiente *Cloud Crawler*. A aplicação considerada no exemplo, aqui chamada de “wiki”, representa uma típica aplicação web desenvolvida segundo o modelo de arquitetura em camadas, onde os componentes da camada web (servidor Apache) e da camada de persistência (servidor de banco de da-

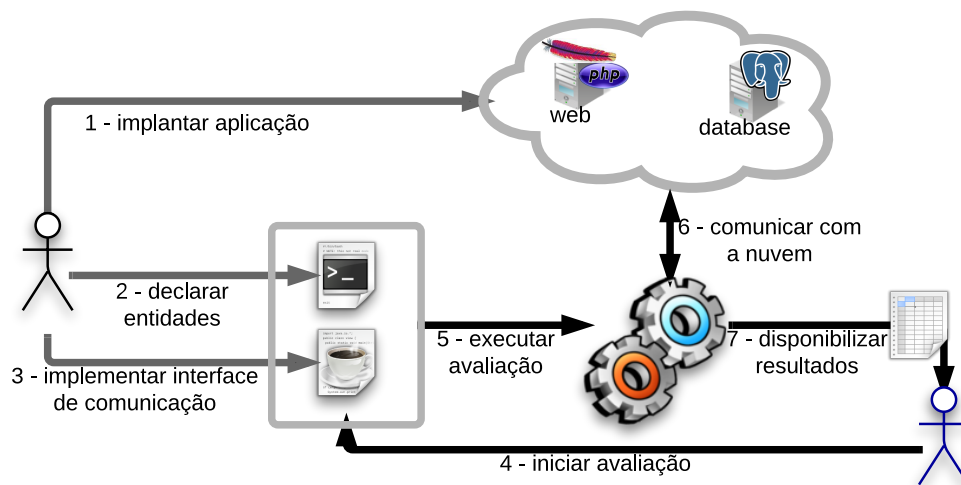


Figura 4.1: Ilustração de um exemplo de utilização do ambiente *Cloud Crawler*.

dos Postgres) são executados em diferentes máquinas virtuais da nuvem. Esse exemplo será utilizado ao longo do capítulo para ilustrar o uso e as funcionalidades do ambiente proposto.

Como pode ser visto na Figura 4.1, a utilização do ambiente *Cloud Crawler* envolve a realização de 7 etapas. São elas:

- 1. Implantar aplicação.** Esta etapa é executada pelo cliente da nuvem e consiste na implantação dos componentes da aplicação na infraestrutura computacional disponibilizada pelo provedor da nuvem. Como, neste primeiro momento, não se sabe ao certo quais os melhores perfis de recursos da nuvem e nem a quantidade certa dos mesmos para atender a demanda da aplicação, aloca-se a quantidade de recursos mínima requerida para a execução dos componentes da aplicação — no caso do exemplo da Figura 4.1, os servidores Apache e Postgres;
- 2. Declarar entidades.** Esta etapa é realizada pelo cliente da nuvem e consiste na declaração das entidades sintáticas da linguagem *Crawl* que representam os componentes da aplicação e demais entidades de apoio que fazem parte dos cenários de avaliações. No caso da aplicação wiki, essas entidades são o provedor da nuvem, as duas máquinas virtuais, o servidor web (Apache) e o servidor de banco de dados (Postgres). Uma vez declaradas, as entidades sintáticas em *Crawl* poderão ser reutilizadas, total ou parcialmente, em diversos outros cenários de avaliação;

3. **Implementar interface para comunicação com aplicação.** Esta etapa é realizada pelo cliente da nuvem e consiste na implementação da interface responsável pela comunicação entre o motor *Crawler* e aplicação que será avaliada. Esta interface será responsável por iniciar e verificar a correta execução dos cenários de avaliação;
4. **Iniciar avaliação.** Nesta etapa, o cliente da nuvem dá início ao processo de avaliação, fazendo uso dos artefatos gerados nas duas etapas anteriores. Esta etapa pode ser realizada repetidas vezes, a critério do usuário;
5. **Executar avaliação.** Esta etapa é realizada pelo motor *Crawler* e consiste na interpretação das entidades declaradas em *Crawl* e na preparação dos recursos da nuvem para a execução dos cenários de avaliação;
6. **Comunicar com a nuvem.** Esta etapa é realizada pelo motor *Crawler* e representa a comunicação entre o ambiente e o provedor da nuvem, tanto para iniciar a execução de um cenário de avaliação, quanto para coletar os resultados das métricas de desempenho ao final da execução de cada cenário;
7. **Disponibilizar resultados.** Esta etapa é realizada pelo motor *Crawler* e representa a disponibilização dos resultados das avaliações ao usuário do ambiente. Essa funcionalidade permite que o cliente da nuvem mantenha uma base de dados com os resultados de diversas avaliações, facilitando, assim, sua análise e visualização.

Cada uma dessas etapas será descrita em mais detalhes nas próximas seções, quando serão apresentados aspectos relevantes de projeto e implementação das ferramentas que compõem o ambiente.

4.2 A Linguagem *Crawl*

A linguagem declarativa *Crawl*¹ foi criada com o objetivo de permitir que as informações necessárias à avaliação de desempenho de uma aplicação na nuvem — tais como os componentes da aplicação e seus parâmetros de configuração, o provedor da nuvem e a quantidade e os perfis dos recursos computacionais alocados a cada componente — possam ser declaradas através de um modelo sintático de alto nível, modular e de fácil entendimento por parte do cliente da nuvem (usuário do ambiente *Cloud Crawler*).

¹Acrônimo para *Cloud resource application and workload language*.

Algumas características inerentes às nuvens de infraestrutura tornam a especificação de cenários de avaliação para esse modelo de nuvem particularmente desafiadora. Uma delas é possibilidade de alocar recursos computacionais à medida em que forem necessários. Essa possibilidade traz consigo novos desafios, uma vez que informações críticas desses recursos só serão conhecidas em tempo de execução. Por exemplo, para configurar uma aplicação que utiliza um banco de dados é preciso informar aos componentes da camada de negócio o endereço de rede (*IP*) do servidor de banco de dados. No entanto, em muitos provedores de nuvem o *IP* de uma máquina virtual só é conhecido após a máquina ter sido efetivamente criada e inicializada, o que impede que os parâmetros de configuração dos componentes da aplicação possam ser totalmente definidos *a priori*.

Além da dinamicidade inerente à alocação de novos recursos, há a dinamicidade relacionada ao perfil do recurso alocado. Isso porque um mesmo componente de uma aplicação pode ter que ser configurado de forma diferente, dependendo do perfil do recurso da nuvem alocado a ele. Por exemplo, um importante parâmetro de configuração do servidor Apache é a quantidade de *workers* (ou *threads*) que irão atender concorrentemente as requisições enviadas ao servidor. A definição desse parâmetro, por sua vez, depende fortemente das características de hardware (como quantidade de CPUs e memória RAM) da máquina onde o servidor será executado. Dessa forma, sempre que novos recursos da nuvem forem alocados, ou houver alguma alteração no perfil dos recursos atualmente em uso, a configuração dos componentes da aplicação poderá ter que ser alterada para melhor adequá-los aos novos perfis.

Para dar suporte a esse caráter dinâmico da nuvem, a linguagem *Crawl* oferece recursos que permitem mudar a configuração dos componentes da aplicação com informações conhecidas tanto em tempo de definição dos cenários de avaliação, quanto em tempo de execução.

As especificações de cenários de avaliação de desempenho na nuvem tendem a compartilhar muitas informações em comum, como, por exemplo, as credenciais de acesso ao provedor da nuvem e os perfis de recursos computacionais disponibilizados pelo provedor. Por esse motivo, *Crawl* foi projetada de modo a permitir e facilitar o reuso de seus elementos sintáticos através de diferentes cenários. Além desses recursos, a linguagem também permite a declaração de diversos cenários de avaliação a partir de variações pontuais nos elementos de um mesmo cenário modelo, o que aumenta o nível de abstração das especificações e, conseqüentemente, reduz o esforço de programação por parte do usuário do ambiente.

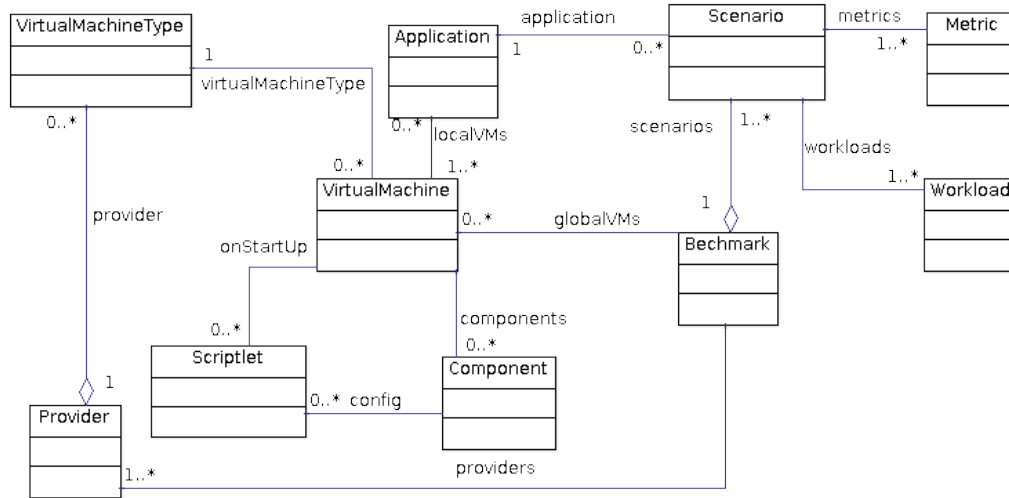


Figura 4.2: Modelo conceitual da linguagem *Crawl*.

Os recursos para especificação de cenários de avaliação de desempenho na nuvem disponíveis em *Crawl* serão detalhados nas próximas subseções, iniciando com uma descrição dos elementos de domínio que compõem o modelo conceitual da linguagem.

4.2.1 Modelo Conceitual

Para permitir a declaração das informações necessárias à avaliação de desempenho de aplicações na nuvem, *Crawl* define um conjunto de entidades do universo específico do domínio de avaliação de desempenho de aplicações. Esse conjunto constitui o *modelo conceitual* da linguagem, é composto por dez entidades, as quais são representadas graficamente no diagrama de classe da Figura 4.2.

Segue uma breve descrição de cada uma das entidades do modelo:

Benchmark — entidade maior do modelo, representa o contexto global de uma avaliação de desempenho especificada na linguagem *Crawl*. Nela, são definidos os provedores de nuvem, os cenários de avaliação e demais atributos de escopo global, comuns aos vários cenários da avaliação;

Provider — representa um provedor de nuvem. Possui como atributos as credenciais de acesso ao provedor e uma lista de perfis de recursos computacionais;

VirtualMachineType — representa um perfil de recurso computacional disponibilizado pelo provedor de nuvem. Por exemplo, pode representar um tipo de instância da

nuvem Amazon EC2 ou um *flavor* do Rackspace, conforme descrito na seção 2.4;

VirtualMachine — representa o recurso (máquina virtual) criado na nuvem de infraestrutura de um determinado provedor. A associação entre a entidade declarada em *Crawl* e a máquina virtual do provedor é feita através do ID do recurso, tal como fornecido pelo provedor. O relacionamento dessa entidade com a entidade *VirtualMachineType* pode ser alterado durante a avaliação, o que significa que o perfil de recurso da máquina virtual pode ser modificado em tempo de execução mantendo o mesmo ID atribuído pelo provedor. Outra característica importante desse entidade é que ela pode ser declarada tanto com escopo global, associada à entidade *Benchmark*, quanto com escopo local, associada à entidade *Application*;

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 *shell script* que deverá ser executado para configurar um componente ou iniciar qualquer serviço da máquina virtual;

Application — representa a aplicação que será implantada na nuvem. Através dessa entidade é possível mapear todos os componentes necessários para a execução da aplicação na nuvem. Uma vez definida, a declaração dessa entidade poderá ser reutilizada em diversos cenários que envolvam a mesma aplicação;

Metric — representa uma métrica de desempenho que será utilizada em um ou mais cenários de avaliação. São exemplos de métricas de desempenho o tempo médio de resposta por requisição, o tempo total de execução da aplicação, e o custo financeiro da aplicação por usuário atendido, dentre outros;

Workload — representa a carga de trabalho (demanda) a ser submetida à aplicação pelo gerador de carga (por exemplo, quantidade de total de usuários simulados pelo gerador, quantidade de requisições enviadas por segundo, etc.);

Scenario — entidade que representam um específico cenário de avaliação que será executado na nuvem pelo motor *Crawler*. Cada cenário envolve uma única aplicação e possui um conjunto de atributos, como os níveis de carga de trabalho e as métricas de desempenho, cujos valores definem como a aplicação será avaliada.

4.2.2 Ilustrando as Entidades do Modelo

Nesta subseção, os componentes do cenário de avaliação da aplicação wiki, exibido como exemplo na Figura 4.1, serão mapeados para elementos da linguagem *Crawl*. O mapeamento será feito na forma de diagramas de objeto da UML (FOWLER; SCOTT, 2000), com o objetivo de ilustrar como as entidades que fazem parte do modelo conceitual da linguagem podem ser instanciadas para representar um típico cenário de avaliação de desempenho na nuvem.

A Figura 4.3 mostra um mapeamento inicial dos componentes da aplicação wiki para entidades de *Crawl*. Os componentes da aplicação estão representados graficamente no lado esquerdo da figura, enquanto as entidades em *Crawl* que correspondem a esses componentes estão representadas no lado direito. Ao todo, foram definidas cinco entidades em *Crawl* para este exemplo: **rackspace**, do tipo *Provider*, representando o provedor da nuvem; **web** e **database**, ambos do tipo *VirtualMachine*, representando as máquinas virtuais da camada web e da camada de persistência, respectivamente; e **apache** e **postgres**, ambos do tipo *Component*, representando o servidor web Apache e o servidor de banco de dados Postgres, respectivamente.

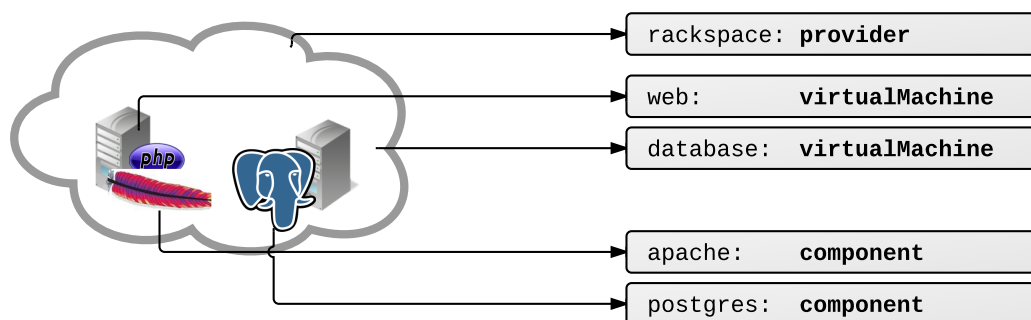


Figura 4.3: Componentes da aplicação wiki mapeados para entidades da linguagem *Crawl*.

A Figura 4.4 mostra uma representação mais detalhada, na forma de um diagrama de objeto, dos componentes da aplicação wiki mapeados em *Crawl*. Note que esse diagrama inclui três novas entidades em relação ao mapeamento inicial ilustrado na Figura 4.3: **bench_1**, do tipo *Benchmark*, representando o contexto geral da avaliação; **flavor_2**, do tipo *VirtualMachineType*, representando um perfil de recurso oferecidos pelo provedor da nuvem; e **apacheConfig**, do tipo *Scriptlet*, representando um *script* de configuração a ser

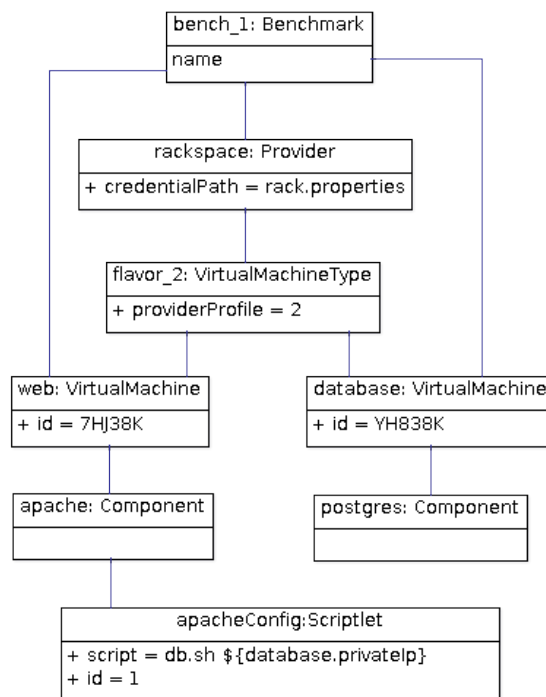


Figura 4.4: Representação em *Crawl* dos componentes da aplicação wiki.

executado em uma das máquinas virtuais. Note, também, que a entidade **bench_1** está associada à entidade **rackspace**, do tipo *Provider*, representando o provedor da nuvem, que por sua vez está associada à entidade **flavor_2**. Esta última associação representa o fato de que o provedor **rackspace** oferece um perfil de recurso computacional de nome **flavor_2**. Mais abaixo no diagrama, é possível observar também a definição de duas máquinas virtuais de escopo global, **web** e **database**, ambas do perfil **flavor_2**. Cada máquina virtual está associada a um componente diferente da aplicação, **apache** e **postgres**, respectivamente, sendo que apenas o componente **apache** está associado ao *script* **apacheConfig**. Esse *script* é responsável por configurar o servidor web com o *IP* da máquina virtual onde será executado o servidor de banco de dados.

Uma vez mapeados os componentes da aplicação, o próximo passo é mapear os elementos que comporão os cenários nos quais a aplicação será avaliada. A Figura 4.5 mostra a especificação completa de um cenário de avaliação para a aplicação wiki representado em *Crawl*. Note que foi definido um único cenário, representado pela entidade de nome **scenario_3_vm**, do tipo *Scenario*, que tem como alvo a aplicação **wiki_3_vm**. A aplicação em questão será executada em três máquinas virtuais, disponibilizadas pelo provedor **rackspace**, todas com o perfil **flavor_2** e alocadas da seguinte forma: duas máquinas virtuais, de nomes **web** e **web2**, para o componente referente ao servidor web, de nome **apache**, e uma máquina virtual, de nome **database**, para o componente referente ao servidor de banco da-

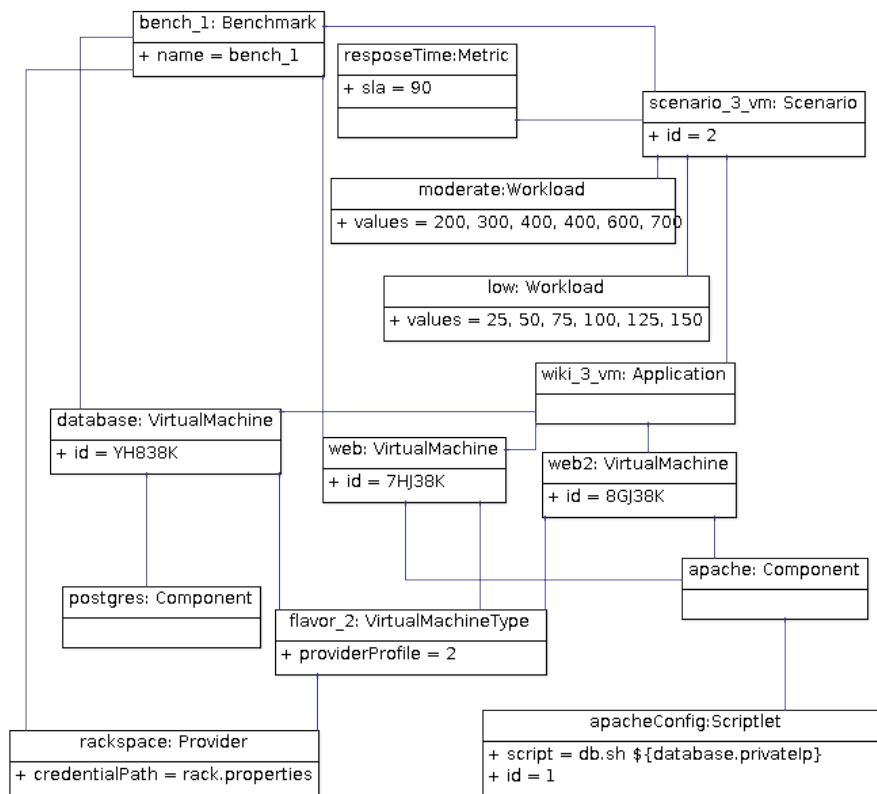


Figura 4.5: Representação em *Crawl* de um cenário de avaliação para a aplicação wiki.

dos, de nome **postgres**. O componente **apache** está associado a um *script* de configuração, de nome **apacheConfig**, que deverá ser executado na máquina virtual alocada ao componente, como parte do seu procedimento de inicialização. Também foram definidas duas entidades do tipo *Workload*, de nomes **low** e **moderate**, representando os diferentes níveis de demanda aos quais a aplicação será submetida. Note que os valores correspondentes a cada nível de demanda (que, no caso da aplicação wiki, representam as diferentes quantidades de requisições simultâneas enviadas ao servidor web da aplicação pelo gerador de carga a cada rodada de execução do cenário) são registrados como atributos das suas respectivas entidades. Por fim, foi definida uma métrica de desempenho, representada pela entidade de nome **responseTime**, do tipo *Metric*, que sinaliza que a aplicação será avaliada nesse cenário com base no seu tempo de resposta, tal como medido pelo gerador de carga.

Conforme pôde ser visto nos exemplos acima, a linguagem *Crawl* suporta um modelo conceitual ao mesmo tempo simples e flexível, oferecendo diversas possibilidades de mapeamento das informações necessárias à avaliação de desempenho de uma aplicação na nuvem. Por exemplo, ao invés de criar uma nova máquina virtual com um novo perfil para

hospedar um componente da aplicação, o cliente da nuvem, responsável pela definição dos cenários de avaliação em *Crawl*, pode optar por simplesmente alterar o perfil da máquina virtual previamente alocada a esse componente.

Outro recurso bastante útil da linguagem, e que pode ajudar a reduzir consideravelmente o custo de execução dos cenários de avaliação junto aos provedores de nuvem, é a possibilidade de definir separadamente máquinas virtuais com escopo global e local. Isso porque máquinas virtuais com escopo global só são finalizadas após a execução de todos os cenários, podendo ser (re)utilizadas em qualquer cenário, enquanto que máquinas de escopo local são iniciadas e finalizadas no contexto de cada cenário em que são definidas, não podendo ser reutilizadas em outros cenários. Dessa forma, ao exercer um maior controle sobre o ciclo de vida das máquinas virtuais que irá utilizar, o cliente da nuvem (usuário do ambiente *Cloud Crawler*) tem muito mais liberdade para ajustar os seus gastos com a contratação de máquinas de virtuais de acordo com o modelo de precificação adotado pelo provedores de nuvem atuais, que cobram o mesmo preço por hora ou fração de hora utilizada de cada máquina. Para dar um exemplo, se uma máquina virtual de escopo global vai ser utilizada em três cenários de uma avaliação com duração total de uma hora, o valor cobrado pelo provedor de nuvem referente a essa máquina será proporcional a apenas uma hora de uso. Por outro lado, se essa mesma máquina for declarada com escopo local, como parte da definição de cada um dos três cenários, ao final da avaliação o valor cobrado pelo provedor referente a essa máquina será equivalente a três horas de uso, uma vez que a máquina terá sido criada e finalizada três vezes (uma vez em cada cenário) durante a avaliação.

Crawl também permite definir diversos cenários de avaliação para a mesma aplicação, variando-se apenas os níveis de demanda e/ou as métricas de desempenho de cada cenário. Esse recurso não apenas propicia uma maior flexibilidade ao usuário da linguagem, como também reduz a complexidade das especificações geradas com ela.

4.2.3 Representando Entidades de *Crawl* em *Yaml*

O modelo conceitual adotado em *Crawl* permite múltiplas representações de suas entidades e seus atributos e relacionamentos. Os diagramas de objeto ilustrados nas Figuras 4.4 e 4.5 são exemplos de uma possível notação para representar visualmente as entidades do modelo. No entanto, uma notação puramente visual sofre de limitações conhecidas, como baixa escalabilidade espacial e possibilidade de sobrecarga de informações, que podem dificultar a compreensão de seu conteúdo por parte do usuário, particularmente no caso

de especificações mais complexas ou envolvendo um grande número de entidades.

Nesse sentido, uma representação textual aparece como uma alternativa mais atraente, não apenas por ser mais compacta e de fácil compreensão por parte do usuário, mas também por ser facilmente manipulável através de ferramentas automatizadas.

Em princípio, qualquer notação textual com um mínimo de estrutura, suficiente para representar entidades, atributos e seus relacionamentos, poderia ser adotada como modelo sintático para a linguagem *Crawl*. A linguagem de marcação *XML* (BRAY *et al.*, 1997), por exemplo, é uma candidata natural, haja vista sua simplicidade e universalidade, além do grande número de ferramentas atualmente disponíveis para seu processamento.

Mais recentemente, porém, *XML* tem perdido espaço como padrão de representação de dados para outras notações textuais mais compactas, como é o caso de *JSON* (JSON, 2011) e *Yaml* (BEN-KIKI *et al.*, 2001), cada vez mais utilizadas como formato de serialização e transmissão de dados no ambiente da web (FONSECA; SIMOES, 2007). Dentre as duas últimas, *Yaml* vem se destacando pela sua sintaxe extremamente simples e flexível, além de ser um superconjunto funcional de *JSON* (FONSECA; SIMOES, 2007). Por essas razões, *Yaml* foi escolhida como base para a representação textual dos elementos da linguagem *Crawl*.

Yaml usa indicadores estruturais limpos e minimalistas, baseados fortemente na identificação de elementos encadeados. Essa característica permite que *Yaml* represente dados de forma muito mais compacta que *XML*, o que tende a tornar ainda mais fácil a compreensão de seu conteúdo:

“Embora não seja menos genérico que o *XML*, o *Yaml* é em grande parte mais simples de ler, editar, modificar e produzir que o *XML*. Ou seja, quase tudo o que é possível de representar em *XML* pode ser representado em *Yaml*, e ao mesmo tempo, de uma forma mais compacta. (FONSECA; SIMOES, 2007).

A lista abaixo oferece um breve resumo dos principais elementos sintáticos de *Yaml*:

‘#’ — sinaliza o início de um comentário;

‘!’ — utilizado para instanciar uma nova entidade;

‘-’ — utilizado para a declaração de listas;

‘&’ — utilizado para definir entidades que serão referenciado em outras partes da representação;

‘*’ — utilizado para referenciar entidades previamente definidas.

```
1 #Definição de dois provedores de nuvem em Crawl
2
3 !provider
4   name: rackspace
5   credentialPath: Rackspace.properties
6 - - -
7 !provider
8   name: ec2
9   credentialPath: EC2.properties
```

Figura 4.6: Representação textual de duas entidades do tipo *Provider*.

A Figura 4.6 mostra como duas entidades de *Crawl*, do tipo *Provider*, podem ser representadas textualmente utilizando a sintaxe de *Yaml*. Nesse exemplo, a linha 1 corresponde a um comentário, sinalizado pelo caractere ‘#’. As linhas 3-5 contêm a definição do primeiro provedor, sinalizada pelo uso do caractere ! na linha 3. Essa entidade possui dois atributos, **name** (linha 4) e **credentialPath** (linha 5), cujos valores são definidos com as *strings* “rackspace” e “Rackspace.properties”, respectivamente. Os três hifens, na linha 6, demarcam o fim da declaração da primeira entidade. A definição do segundo provedor, nas linhas 7-9, é feita de forma análoga.

Um outro exemplo, um pouco mais complexo, é dado na Figura 4.7, que ilustra a definição de uma entidade do tipo *Benchmark* com duas máquinas virtuais. Essa entidade (cuja definição inicia na linha 3) possui cinco atributos, **name**, **rounds**, **interval**, **providers** e **virtualMachines**, sendo que os dois últimos são definidos na forma de uma lista de entidades (dos tipos *Provider* e *VirtualMachine*, respectivamente). Note que as declarações das duas entidades da lista de provedores, **rackspace** (linha 8) e **ec2** (linha 11), são precedidas do caractere ‘&’, que em *Yaml* indica que essas entidades poderão ser referenciadas de outros locais da especificação. Referências a entidades previamente definidas são sinalizadas em *Yaml* através do caractere ‘*’, como ocorre nas linhas 19 e 23 da Figura 4.7, onde as entidades **rackspace** e **flavor_2** (definida na linha 17), respectivamente, são referenciadas.

Como pode-se observar nesse último exemplo, *Yaml* é bastante flexível quanto ao escopo das definições, permitindo que novas entidades sejam definidas a qualquer momento e referenciadas sempre que necessário. Tal flexibilidade se reflete 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. Esse poder da linguagem *Crawl* pode ser melhor apreciado na Figura 4.8, que exibe a representação textual completa para o cenário de avaliação da aplicação wiki ilustrado na Figura 4.5.

```
1 #Definição de um benchmark com duas máquinas virtuais em Crawl
2
3 !benchmark
4 name: bench_1
5 rounds: 3
6 interval: 5000
7 providers:
8   - &rackspace !provider
9     credentialPath: Rackspace.properties
10
11   - &ec2 !provider
12     credentialPath: EC2.properties
13
14 virtualMachines:
15   - !virtualMachine
16     id: YH838K
17     type: &flavor_2 !virtualMachineType
18     providerProfile: 2
19     provider: *rackspace
20
21   - !virtualMachine
22     id: 7HJ38K
23     type: *flavor_2
```

Figura 4.7: Representação textual de uma entidade do tipo *Benchmark*.

Note que, na representação da Figura 4.8, as definições pertinentes aos componentes da aplicação wiki (linhas 7-32) bem como ao cenário de avaliação escolhido (linhas 34-58) estão claramente visíveis. Além disso, essa representação inclui algumas informações adicionais em relação ao diagrama de objetos da Figura 4.5. Por exemplo, a entidade do tipo *Benchmark* possui dois novos atributos, **rounds** (linha 3) e **interval** (linha 4), que representam, respectivamente, o número de vezes que o *benchmark* será executado e o intervalo de tempo, em milisegundos, a ser aguardado antes de cada nova execução. Outra novidade é a definição completa do *script* de configuração a ser executado durante a inicialização da máquina virtual alocada ao servidor web Apache (linhas 21-25).

4.2.4 Estendendo *Yaml*

Apesar de sua simplicidade e flexibilidade, o *Yaml*, na sua notação original, apresentou algumas restrições práticas que, dependendo da complexidade do cenário de avaliação de desempenho sendo especificado em *Crawl*, podem resultar em representações mais longas e, conseqüentemente, mais difíceis de compreender e manipular.

Uma dessas restrições é ausência de elementos sintáticos que permitam a importação de entidades declaradas externamente. Tal restrição obriga o usuário da linguagem *Crawl* a sempre (re)definir todas as entidades presentes na especificação de um determinado *benchmark*, mesmo quando parte dessas entidades já tenha sido definida no contexto de outros *benchmarks*.

1	!benchmark	30	components:
2	name: bench_1	31	- &postgres !component
3	rounds: 3	32	id: 2
4	interval: 5000	33	
5		34	scenarios:
6	#Definição dos provedores de nuvem	35	- !scenario
7	providers:	36	id: 1
8	- &rackspace !provider	37	workloads:
9	credentialPath: Rackspace.properties	38	- &low !workload
10		39	id: 1
11	#Definição das máquinas virtuais	40	values: 25,50,75,100,125,150
12	virtualMachines:	41	
13	- &web !virtualMachine	42	- &moderate !workload
14	id: 7HJ38K	43	id: 2
15	type: &flavor_2 !virtualMachineType	44	values: 200,300,400,500,600,700
16	providerProfile: 2	45	
17	provider: *rackspace	46	metrics:
18	components:	47	- &responseTime !metric
19	- &apache !component	48	sla: 90
20	id: 1	49	
21	onStartup:	50	application: &wiki_3_vm !br.Wiki
22	- &databaseConfig !scriptlet	51	database: *database
23	id: 1	52	web:
24	scripts:	53	- *web
25	db.sh \${database.privateIP}	54	- &web2 !instance
26		55	id: 8GJ38K
27	- &database !virtualMachine	56	type: *flavor_2
28	id: YH838K	57	components:
29	type: *flavor_2	58	- *apache

Figura 4.8: Representação textual do cenário de avaliação definido para a aplicação wiki.

Outra restrição diz respeito à forma como os cenários de avaliação são especificados em *Crawl*, com vários cenários diferentes podendo ser definidos com pequenas variações em relação uns aos outros (por exemplo, pode-se definir um conjunto de cenários onde a única diferença de um para outro são os perfis dos recursos da nuvem alocados aos componentes da aplicação). Como cada novo cenário representa uma nova entidade da linguagem, na sintaxe original de *Yaml* cada variação de cenário tem que ser definida separadamente, o que obriga o usuário a repetir a parte comum das definições para cada cenário.

Para eliminar as restrições discutidas acima, duas extensões à sintaxe original de *Yaml* foram propostas neste trabalho. A primeira extensão permite a inclusão de entidades definidas em arquivos externos como parte das especificações feitas em *Crawl*. Já a segunda extensão permite a definição de laços de interação, através dos quais é possível definir, de um forma simples e compacta, um conjunto de entidades similares que compartilham vários elementos em comum. Essas duas extensões são descritas em mais detalhes a seguir.

4.2.4.1 Inclusão de Arquivos Externos

Especificações de cenários de avaliação de desempenho de aplicações na nuvem tendem a compartilhar muitas informações em comum. Por exemplo, definições referentes aos provedores de nuvem e aos tipos de recursos computacionais que eles disponibilizam são

1	!benchmark	20	values: 25,50,75,100,125,150
2	name: bench_1	21	
3	rounds: 3	22	- &moderate !workload
4	interval: 5000	23	id: 2
5		24	values: 200,300,400,500,600,700
6	#Definição dos provedores de nuvem	25	
7	!include	26	metrics:
8	file: /home/providers.yml	27	- &responseTime !metric
9		28	sla: 90
10	#Definição das máquinas virtuais	29	
11	!include	30	application: &wiki_3_vm !br.Wiki
12	file: /home/machines-wiki.yml	31	database: *database
13		32	web:
14	scenarios:	33	- *web
15	- !scenario	34	- &web2 !instance
16	id: 1	35	id: 8GJ38K
17	workloads:	36	type: *flavor_2
18	- &low !workload	37	components:
19	id: 1	38	- *apache

Figura 4.9: Representação textual do cenário de avaliação definido para a aplicação wiki, com a utilização do recurso de inclusão de arquivos externos.

comuns à especificação de quaisquer cenários de avaliação que utilizem algum desses provedores, podendo corresponder uma parte significativa da especificação final. Para dar uma ideia, só os provedores Amazon EC2 e Rackspace juntos somam mais de 20 tipos de recursos computacionais. Da mesma forma, definições referentes aos componentes de uma aplicação são comuns à especificação de quaisquer cenários envolvendo essa mesma aplicação.

Para permitir o reuso de entidades externamente definidas na linguagem *Crawl*, e assim facilitar o compartilhamento de entidades comuns por diferentes especificações, a sintaxe original de *Yaml* foi estendida com um novo recurso de importação de arquivos externos. Esse recurso é representado pela entidade especial `!include`, que contém um único atributo de nome `file`. O valor desse atributo corresponde ao caminho do arquivo externo cujo conteúdo deve ser importado para ser incorporado à especificação atual. Dessa forma, todas as entidades definidas no arquivo externo poderão ser referenciadas como parte da nova especificação.

A Figura 4.9 mostra uma nova representação textual para o mesmo cenário de avaliação representado na Figura 4.8, agora utilizando o recurso de inclusão de arquivos externos. Note que, nessa nova representação, tanto a definição dos provedores de nuvem (linhas 7-8) quanto a definição das máquinas virtuais utilizadas pela aplicação wiki (linhas 11-12) são importadas de dois arquivos externos, de nomes “providers.yml” e “machines-wiki.yml”, respectivamente, o que contribui para uma redução significativa no tamanho da especificação do *benchmark* em relação ao seu tamanho original. Nesse exemplo, fica evidente as vantagens da utilização do recurso de importação de arquivos, que, além de

1	#Definição dos perfis de recurso	16	providerProfile: 5
2	virtualMachineTypes:	17	provider: *rackspace
3	– &1_flavor !virtualMachineType	18	– &6_flavor !virtualMachineType
4	providerProfile: 1	19	providerProfile: 6
5	provider: *rackspace	20	provider: *rackspace
6	– &2_flavor !virtualMachineType	21	– &7_flavor !virtualMachineType
7	providerProfile: 2	22	providerProfile: 7
8	provider: *rackspace	23	provider: *rackspace
9	– &3_flavor !virtualMachineType	24	– &8_flavor !virtualMachineType
10	providerProfile: 3	25	providerProfile: 8
11	provider: *rackspace	26	provider: *rackspace
12	– &4_flavor !virtualMachineType	27	– &9_flavor !virtualMachineType
13	providerProfile: 4	28	providerProfile: 9
14	provider: *rackspace	29	provider: *rackspace
15	– &5_flavor !virtualMachineType		

Figura 4.10: Especificação dos perfis de recurso de um provedor de nuvem utilizando a sintaxe original de *Yaml*.

promover o reuso de código entre as especificações, ao permitir a separação explícita entre entidades de contexto mais específico e aquelas de contexto mais global, ajuda a tornar as especificações em *Crawl* ainda mais legíveis e modulares.

4.2.4.2 Definição de Laços de Iteração

Conforme mencionado anteriormente, é comum a especificação de entidades de *Crawl* com pequenas variações em relação umas às outras. A Figura 4.10 mostra um trecho da definição de nove perfis de recurso computacional oferecidos pelo provedor de nuvem Rackspace, utilizando a sintaxe original de *Yaml*. Observe que a diferença entre as declarações de cada uma das entidades do tipo *VirtualMachineType* definidas nesse exemplo é apenas o número que identifica o seu respectivo perfil, causando uma incômoda e, na verdade, desnecessária redundância de código na representação resultante.

Para diminuir o volume de código gerado — e, consequentemente, o esforço de programação necessário por parte do usuário da linguagem *Crawl* — em situações em que trechos de uma especificação podem ser reusados com pequenas modificações, a sintaxe original de *Yaml* foi estendida com um novo recurso que permite a definição de laços de iteração. Esse recurso é representado pela entidade especial **!foreach**, que possui quatro atributos. O primeiro atributo, **list**, contém uma lista com os valores a serem iterados no laço. Os elementos dessa lista devem ser todos do tipo texto e separados por vírgula. O segundo atributo, **count**, corresponde ao contador de iterações do laço, cujo valor inicial será definido pelo usuário. O contador será incrementado de uma unidade cada nova iteração. O terceiro atributo, **var**, corresponde ao nome da variável que receberá cada item da lista de valores do laço. Finalmente, o quarto atributo, **statement**, contém a definição dos elementos que estão dentro do escopo do laço. Dentro desse escopo, é possível usar

```
1 #Definição dos perfis de recurso
2 virtualMachineTypes:
3   - !foreach
4     list: 1, 2, 3, 4, 5, 6, 7, 8, 9
5     count: 0
6     var: flavor
7     statement:
8   - &${flavor}_flavor !virtualMachineType
9     providerProfile: ${flavor}
10    provider: *rackspace
```

Figura 4.11: Especificação dos perfis de recurso de um provedor de nuvem utilizando um laço de iteração.

um segundo delimitador de variável, `$$`, através do qual pode-se referenciar os atributos `count` e `var`, os quais serão substituídos por seus respectivos valores em 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 da anterior apenas nos trechos que referenciam os valores dos atributos `count` e `var`.

A Figura 4.11 mostra uma nova representação, equivalente à representação da Figura 4.10, mas agora utilizando um laço de interação para definir os diferentes perfis de recurso do Rackspace. Note, na Figura 4.11, que a lista de valores do laço (linha 4) corresponde à sequência de nove números que identificam cada um dos perfis de recurso daquele provedor. Note, também, que apenas uma entidade do tipo *VirtualMachineType* está sendo explicitamente definida dentro do escopo do laço (linhas 7-10), e que tanto o nome (linha 8) quanto o valor do atributo `providerProfile` dessa entidade (linha 9) fazem referência à variável do laço, no caso, `flavor`, a qual será substituída por cada um dos valores da lista a cada iteração.

Em termos de linhas de código, a nova representação tem cerca de um terço do tamanho da representação original, o que ilustra bem o poder desse recurso da linguagem *Crawl* não apenas para reduzir a o tamanho das especificações, mas também para aumentar a sua legibilidade.

4.3 O Motor *Crawler*

O motor *Crawler* é responsável por automatizar a preparação e a execução dos cenários de avaliação de desempenho declarados na linguagem *Crawl*. Visando torná-lo independente de uma representação específica, o *Crawler* também define um conjunto próprio de entidades para avaliação de desempenho. Esse conjunto é aderente ao modelo conceitual suportado pela linguagem *Crawl* (descrito na seção 4.2.1), e pode ser especializado pelo

usuário do ambiente para definir entidades mais específicas da aplicação a ser avaliada.

A preparação dos cenários de avaliação consiste, basicamente, em inicializar as máquinas virtuais na nuvem e nelas configurar os componentes da aplicação, de acordo com os recursos definidos para cada cenário. Para inicializar as máquinas virtuais, o *Crawler* precisa acessar os serviços disponibilizados pelos seus respectivos provedores de nuvem. Como atualmente não há um padrão de acesso a serviços de nuvem, com cada provedor oferecendo uma interface própria — muitas vezes proprietária —, o *Crawler* foi projetado de modo a suportar diferentes provedores com diferentes interfaces de acesso.

Para oferecer maior liberdade de utilização do ambiente por parte do usuário, o *Crawler* também oferece uma interface genérica para comunicação com os componentes da aplicação na nuvem. É através dessa interface que o *Crawler* executa ações como iniciar a execução de uma determinada carga de trabalho e coletar os resultados das avaliações. Essa interface deve ser implementada pelo usuário do ambiente, que assim pode customizar o comportamento do *Crawler* de acordo com as especificidades dos cenários de avaliação definidos para a sua aplicação.

4.3.1 Arquitetura

A arquitetura do *Crawler* é composta de cinco módulos, conforme ilustrado na Figura 4.12. O módulo ***Engine*** é o principal componente da arquitetura, responsável por coordenar as ações entre os demais módulos. O módulo ***Parser*** é responsável por validar as especificações na linguagem *Crawl*, recebidas como dados de entrada do motor, e instanciar as entidades correspondentes do *Crawler*. O módulo ***Builder*** é encarregado de configurar os recursos da nuvem necessários a cada cenário, verificando também o seu estado antes de cada execução. Por fim, o módulo ***Executer*** é responsável pela execução dos cenários na nuvem e pela coleta dos resultados das avaliações.

Cada um desses módulos será descrito em mais detalhes nas próximas subseções.

4.3.1.1 Módulo ***Engine***

O módulo *Engine* é responsável por iniciar e coordenar a execução dos demais módulos do motor. O módulo recebe como entrada as especificações dos cenários de avaliação, descritas na linguagem *Crawl*. Essas especificações são então repassadas ao módulo *Parser*, que instancia um conjunto de entidades do *Crawler* correspondentes às entidades especificadas em *Crawl*. De posse dessas entidades, o *Engine* solicita ao módulo *Builder* a preparação e

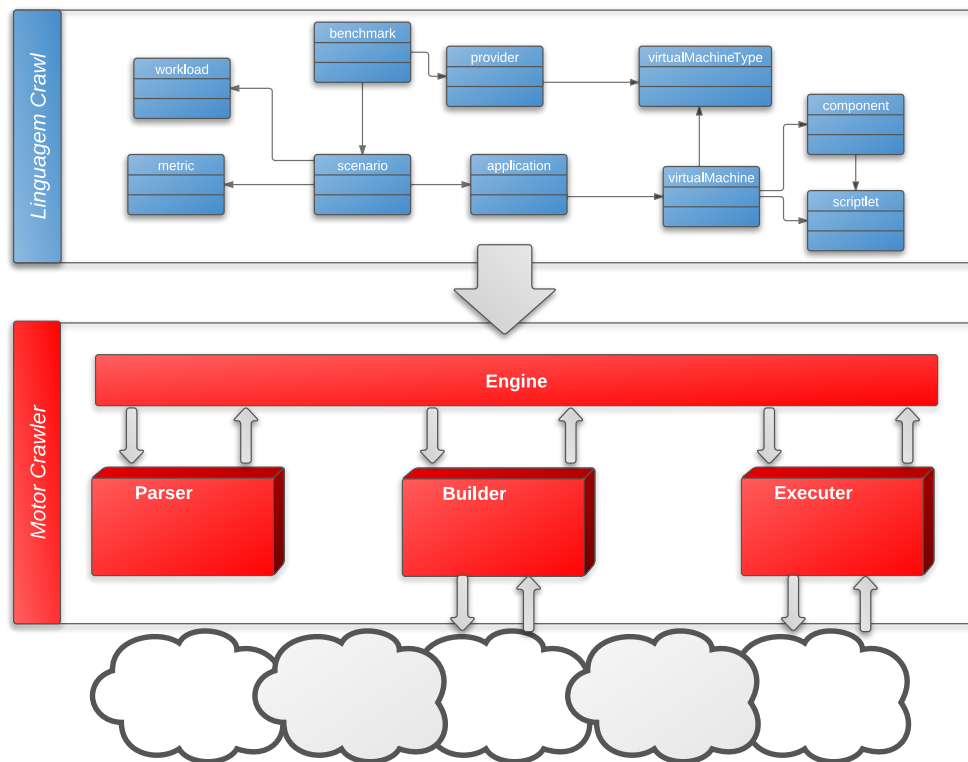


Figura 4.12: Arquitetura do motor *Crawler*.

configuração dos recursos da nuvem. Ao final da preparação, o *Engine* solicita ao *Executer* que dê início à execução dos cenários na nuvem. Finalizada a execução de um determinado cenário, o *Executer* encaminha ao *Engine* os dados referentes aos resultados da avaliação, e segue para a execução do próximo cenário, caso haja.

4.3.1.2 Módulo *Parser*

O módulo *Parser* é responsável por validar e mapear as definições dos cenários de avaliação a serem executados na nuvem. Ele recebe como entrada as entidades definidas na linguagem *Crawl* pelo usuário do ambiente, e fornece como saída para o *Engine* as definições mapeadas para entidades do *Crawler*.

Além de realizar esse mapeamento inicial, o módulo é responsável por interpretar e definir os valores das variáveis definidas em *Crawl*. Um exemplo da definição de uma variável encontra-se na linha 25 da Figura 4.8, onde o script de configuração do componente *apache* recebe como parâmetro o endereço de rede (*IP*) do banco de dados. Como, em cada cenário, a aplicação pode ser executada em diferentes perfis de recurso, então o *Engine* deverá consultar o *Parser* antes de solicitar a execução de cada um dos cenários. Uma

vez que a modificação do perfil de recurso, pode significar a modificação na configuração de um componente da aplicação.

4.3.1.3 Módulo *Builder*

O módulo *Builder* é responsável por preparar os recursos dos provedores de nuvem e por configurar os componentes da aplicação implantada nesses recursos. Para isso, o *Builder* requisita os serviço de gerenciamento e configuração de recursos oferecidos pelos seus próprios provedores. O acesso a esses serviços é realizado através da utilização de credenciais de acesso, que devem ser informadas pelo usuário do ambiente no momento da definição de cada provedor.

Para a preparação dos recursos da nuvem, o *Builder* segue o fluxo de trabalho especificado na Figura 4.13. A primeira tarefa do fluxo é verificar se existem máquinas virtuais definidas com escopo global na especificação *Crawl*. Em seguida, o módulo verifica se o perfil do recurso de cada máquina com escopo global corresponde ao perfil definido para o cenário a ser executado. Caso o perfil declarado no cenário seja igual ao da máquina virtual, então ela será iniciada. No caso dos perfis serem diferentes, o *Builder* alterará o perfil da máquina virtual para que fique em acordo com o perfil esperado para o cenário, para só então iniciá-la. Após iniciar todas as máquinas virtuais de escopo global, o *Builder* prossegue para a inicialização das máquinas virtuais de escopo local, ou seja, específicas do cenário a ser executado.

Essa modificação que é realizada pelo *Builder* no perfil de uma máquina virtual, em tempo de execução dos cenários, permite que uma mesma máquina seja reutilizada com diferentes configurações, sem que seja necessária a criação de novas máquinas virtuais junto ao provedor. Essa estratégia tem como principal vantagem a diminuição do tempo e do custo de execução do cenários na nuvem. Conforme discutido na seção 4.2.2, os provedores de nuvem atuais adotam um modelo de precificação no qual é cobrado o mesmo valor por hora ou fração de hora utilizada de cada máquina virtual criada. Portanto, dependendo do tempo de utilização da máquina virtual, reutilizar uma máquina existente pode sair muito mais barato do que solicitar a criação de uma nova ao provedor.

Outro papel do *Builder* é o de verificar o estado dos recursos da nuvem antes da execução de cada cenário. Tal verificação é necessária para evitar situações comuns de perda de conectividade com uma ou mais máquinas virtuais de um determinado provedor. Essas situações normalmente acontecem quando o provedor considera uma máquina virtual no estado “pronta”, mas o cliente da nuvem não consegue acessá-la através da Internet. Nesse

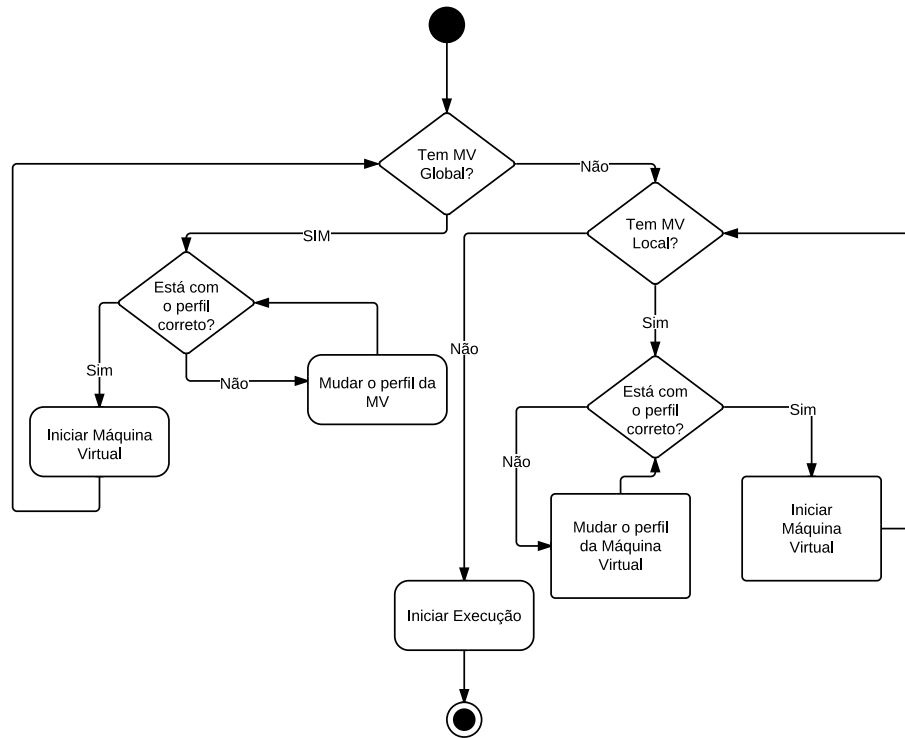


Figura 4.13: Fluxo de inicialização das máquinas virtuais executado pelo módulo *Builder*.

caso, é responsabilidade do *Builder* verificar se a máquina virtual está acessível para só então prosseguir com a configuração de seus componentes. Na ocorrência de falha na verificação, o *Builder* reinicia a máquina virtual e tenta se conectar novamente. Até que a conectividade seja confirmada, o *Builder* continuará reiniciando a máquina virtual.

4.3.1.4 Módulo *Executer*

O módulo *Executer* é responsável por iniciar e controlar a execução dos cenários de avaliação da aplicação no ambiente de nuvem previamente configurado pelo *Builder*. Esse módulo também é responsável por definir as entidades do motor *Crawler* que representarão as entidades especificadas na linguagem *Crawl*, e por definir e utilizar a interface genérica para comunicação com os componentes da aplicação na nuvem. Conforme mencionado anteriormente, essa interface é o mecanismo que permite, aos usuários do ambiente, customizar o comportamento do *Crawler* para atender as peculiaridades da sua aplicação. Com efeitos, é através dessa interface que o *Executer* verifica se há alguma avaliação de desempenho em andamento, e também coleta e devolve para o módulo *Engine* os resultados das avaliações já realizadas referentes a cada uma das cargas de trabalho definidas

para cada cenário.

Vale ressaltar que a customização do *Crawler* precisa ser realizada apenas uma vez para cada aplicação, através da implementação da interface genérica definida pelo módulo *Executer*. Essa customização é restrita a atividades específicas de interação com os componentes da aplicação, não envolvendo nenhuma atividade relativa à configuração desses componentes ou dos recursos da nuvem utilizados por eles. Dessa forma, o modelo conceitual suportado pela linguagem *Crawl* é preservado, já que todas as informações necessárias à configuração das aplicação e dos cenários de avaliação são providas pelo usuário exclusivamente através das especificações em *Crawl*.

4.3.2 Implementação de um Protótipo

Como prova de conceito para o ambiente *Cloud Crawler*, um protótipo do motor *Crawler* foi implementado na linguagem Java. A Figura 4.14 mostra a arquitetura do protótipo.

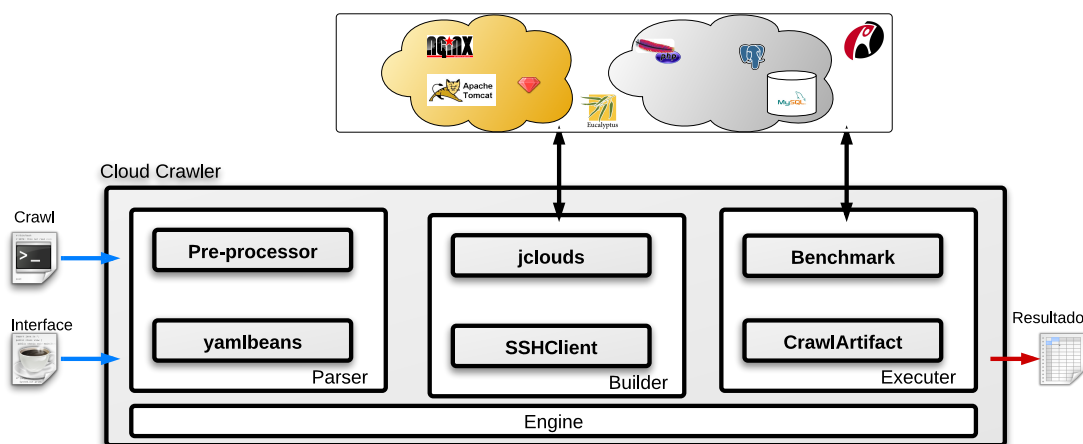


Figura 4.14: Arquitetura do protótipo implementado para o motor *Crawler*.

Como pode ser observado nessa figura, o motor recebe como entrada uma especificação em *Crawl* e uma classe escrita em Java, que implementa a interface de comunicação utilizada pelo módulo *Executer* para interagir com os componentes da aplicação avaliada na nuvem, e produz como saída os dados referentes aos resultados das avaliações. Tanto o módulo *Builder* quanto o *Executer* acessam os recursos e serviços da nuvem através de invocações remotas. Dessa forma, o motor *Crawler* pode ser executado tanto dentro quanto fora da infraestrutura da nuvem onde a aplicação será avaliada, ficando essa decisão a critério do usuário do ambiente.

As próximas subseções dão mais detalhes sobre a implementação de cada um dos quatro módulos que compõem o protótipo do motor *Crawler*.

4.3.2.1 Implementação do Módulo *Engine*

O módulo *Engine* tem como classe principal a *EngineImpl*, que controla a execução dos demais módulos através de chamadas diretas às classes de cada um deles. Essa classe lê os dados dos arquivos contendo as especificações em *Crawl*, para em seguida encaminhá-los ao *Parser*, para que sejam validados e mapeados para entidades do *Crawler*. Essas entidades são então repassadas ao *Builder* e ao *Executer*, nessa ordem, para que se encarreguem de configurar o ambiente de nuvem e realizar a execução dos cenários de avaliação da aplicação, respectivamente. Ao final, a classe *EngineImpl* disponibiliza os resultados das avaliações ao usuário na forma de planilhas, para que possam ser facilmente analisados e visualizados.

4.3.2.2 Implementação do Módulo *Parser*

Inicialmente, o *Parser* interpreta as extensões do Yaml que tratam da inclusão de arquivos e da definição de laços de iteração, o que é feito através da classe *CrawlParserYml*, responsável pela análise léxica e sintática das extensões feitas ao Yaml. Essa classe processa os dados completos das especificações em *Crawl*, encaminhados pelo módulo *Engine*, instanciando uma classe do tipo *StringStatement* para cada linha processada. No caso da linha conter uma declaração de inclusão de arquivo ou a definição de um laço de iteração, esta será tratada pela classe *IncludeStatement* ou *ForStatement*, conforme o caso. Todas as três classes, *StringStatement*, *IncludeStatement* e *ForStatement*, implementam a interface *Statement*. Além disso, as classes *IncludeStatement* e *ForStatement* podem conter ambas uma lista de outros elementos do tipo *Statement*, permitindo, assim, que, por exemplo, um arquivo incluído tenha dentro dele a definição de um laço de iteração ou a inclusão de um outro arquivo. Ao final dessa fase, as especificações em *Crawl* são convertidas para especificações puramente em *Yaml*.

Uma vez convertidos para *Yaml*, os dados das especificações em *Crawl* podem ser automaticamente mapeados para classes de Java com o auxílio de diversas ferramentas desenvolvidas para este fim. Uma dessas ferramentas é o *yamlbeans*,² que foi a ferramenta escolhida para ser utilizada na implementação do protótipo. Essa ferramenta disponibiliza a classe *YamlLoader*, utilizada pelo *Parser* para mapear os elementos de *Crawl*, especi-

²<http://code.google.com/p/yamlbeans/>

ficados em *Yaml*, para o conjunto de classes de Java utilizado pelo *Crawler*, conforme definido pelo módulo *Executer*.

Dentre as classes utilizadas pelo *Crawler*, ilustradas no diagrama de classe da Figura 4.15, apenas duas merecem destaque, uma vez que as demais são equivalentes às entidades que fazem parte do modelo conceitual da linguagem *Crawl*, descrito na seção 4.2.1. São elas:

CrawlArtifact — classe abstrata utilizada como superclasse para todas outras classes.

Contém dois atributos: um identificador e uma lista de propriedades. Este último é implementado como um mapa (tipo *Map* de Java), permitindo que qualquer valor, não previamente mapeado, possa ser associado a uma chave e armazenado junto com os demais atributos das subclasses, sem a necessidade de criação de um atributo específico na subclasse;

MetricEval — classe associada a uma métrica de desempenho, utilizada para armazenar o resultado da avaliação da aplicação em um determinado cenário (por exemplo, se a aplicação atendeu ou não ao critério de qualidade definido pela métrica).

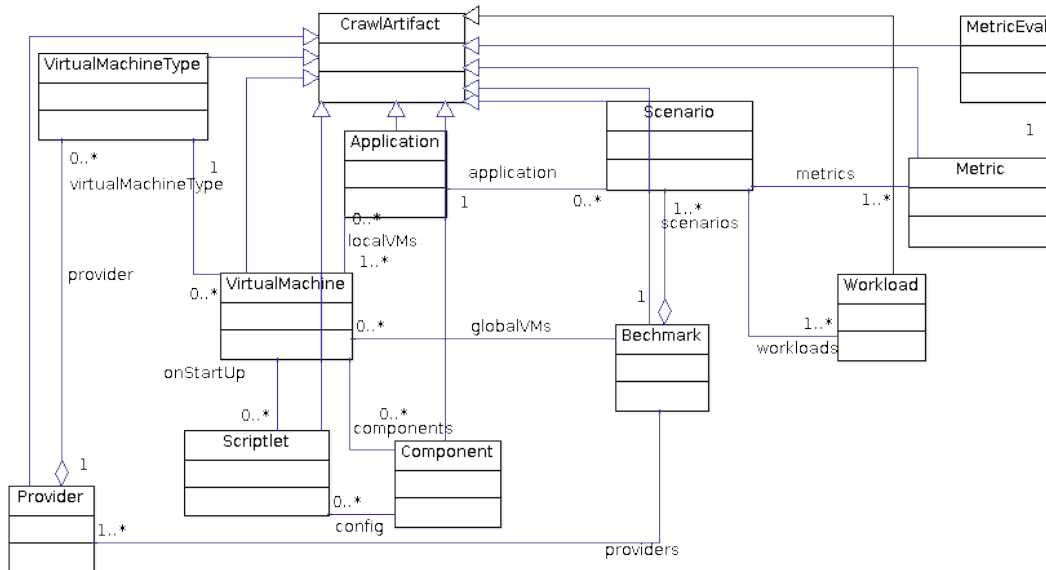


Figura 4.15: Diagrama de classe representando as entidades de avaliação de desempenho utilizadas internamente pelo motor *Crawler*.

4.3.2.3 Implementação do Módulo *Builder*

Para a configuração dos recursos da nuvem, realizada pelo módulo *Builder*, foi implementada a classe *ComputeProvider*. Essa classe utiliza a biblioteca de código aberto *jclouds*,³ que oferece uma interface comum, em Java, para acessar os serviços de diferentes provedores de nuvem. A grande vantagem decorrente da utilização do *jclouds* é que o código de acesso aos serviços da nuvem fica, em grande parte, transparente quanto ao provedor escolhido.

Por outro lado, por ser um produto em constante desenvolvimento, o *jclouds* ainda não oferece suporte para algumas funcionalidades específicas de alguns provedores, como a possibilidade de mudança do perfil dos recursos computacionais criados na nuvem. Nesses casos, o desenvolvedor precisa invocar diretamente os serviços específicos desses provedores, como ocorreu na implementação do *Builder*.

O diagrama de sequência da Figura 4.16 ilustra a implementação do procedimento de preparação dos recursos da nuvem executado pelo módulo *Builder*. O primeiro passo é realizado pela classe *EngineImpl*, que solicita à classe *ComputeProvider* do *Builder* a inicialização de uma máquina virtual. Os parâmetros de entrada dessa solicitação são, a máquina virtual, a aplicação, e a referência ao módulo *Parser*. Em seguida, através da classe *ComputeServiceContext*, a máquina virtual é finalmente inicializada. Após o retorno dessa solicitação, a classe *ComputeProvider* verifica se a máquina possui conectividade com a Internet, para só então solicitar que as variáveis dos *scripts* de configuração tenham seus valores atualizados pela classe *ScriptParser*. São realizadas três chamadas a essa classe, uma vez que podem ser configurados tantos os componentes, quanto a aplicação. Só depois da execução das configurações especificadas nos *scripts* é que a máquina virtual é considerada preparada para a execução.

Para a verificação da existência de conectividade com uma máquina virtual, o *Builder* utiliza a ferramenta *JSch*,⁴ que oferece uma implementação totalmente em Java do protocolo *SSH2* (BARRETT *et al.*, 2005). Essa ferramenta foi utilizada como uma alternativa à chamada de clientes específicos do sistema operacional para acesso a máquinas remotas via protocolo *SSH*, de modo a tornar a implementação do módulo transparente quanto ao sistema operacional utilizado pelo usuário do ambiente.

³<http://www.jclouds.org/>

⁴<http://www.jcraft.com/jsch/>

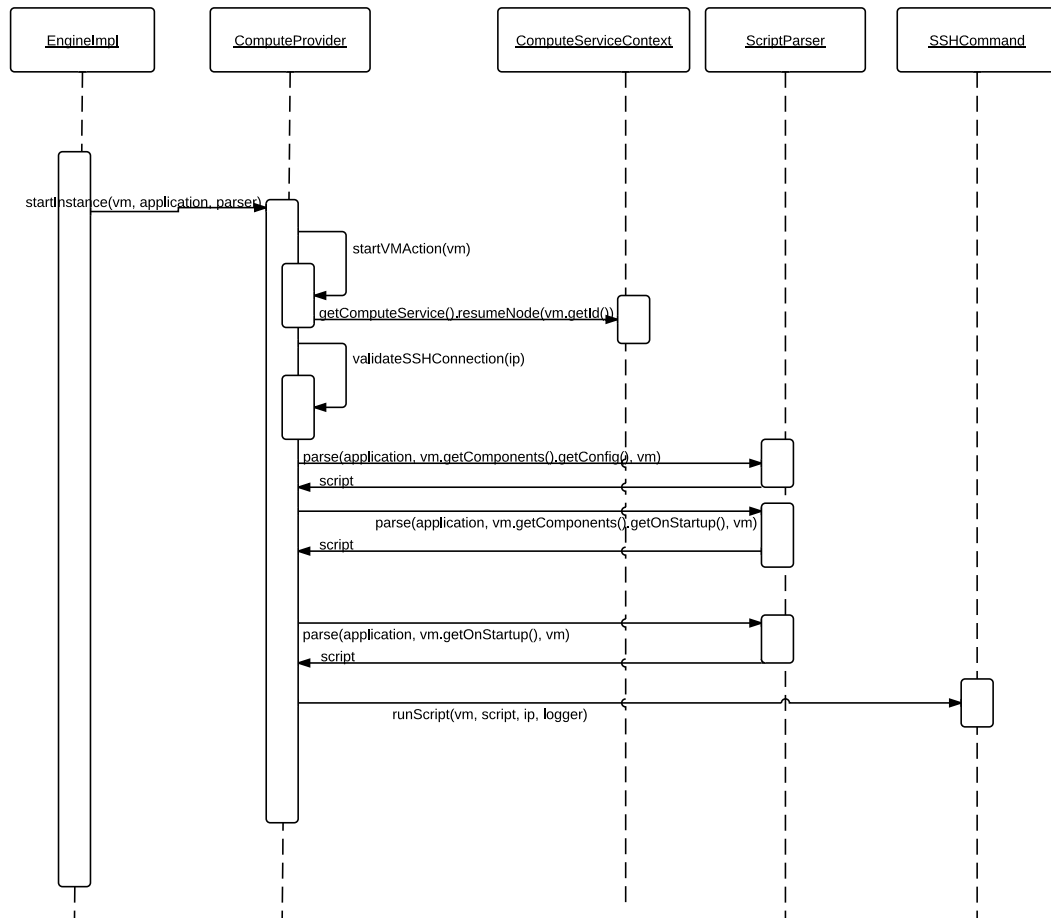


Figura 4.16: Diagrama de sequência ilustrando o procedimento de preparação dos recursos da nuvem implementado pelo módulo *Builder*.

4.3.2.4 Implementação do Módulo *Executer*

Para permitir um maior controle sobre as execuções dos *benchmarks* de avaliação de desempenho realizadas pelo *Crawler*, o módulo *Executer* disponibiliza a classe abstrata *Execution*. Essa classe recebe o cenário que será avaliado como parâmetro de entrada para o seu construtor, e disponibiliza três métodos abstratos. O primeiro é responsável por iniciar a execução do cenário e recebe como parâmetro o valor referente a carga a ser avaliada (por exemplo, a quantidade de usuários concorrentes que irão acessar a aplicação); o segundo método verifica se a avaliação está em execução; e o terceiro retorna o resultado da execução para cada métrica avaliada. Assim, para cada *benchmark* deve ser implementada uma extensão da classe *Execution*, através da qual o *Executer* será capaz de coletar as métricas de desempenho durante a execução dos cenários para cada tipo de carga a que a aplicação for exposta. Finalizada a execução de uma carga, a classe *Execution* retorna o resultado da avaliação de cada métrica, os quais são disponibilizados

ao usuário do ambiente pelo módulo *Engine*, na forma de planilhas.

4.4 Resumo do Capítulo

Neste capítulo, foi apresentado o ambiente *Cloud Crawler*, composto pela linguagem declarativa *Crawl* e o motor de execução *Crawler*. A linguagem *Crawl* oferece um conjunto de elementos sintáticos que permitem a especificação, em alto nível e de forma modular, de diversos cenários de avaliação de desempenho na nuvem. As especificações feitas em *Crawl* constituem os dados de entrada para o motor *Crawler*, que realiza, de forma automática, a configuração, execução e coleta dos resultados de cada cenário de avaliação de desempenho especificado em *Crawl*.

Como prova de conceito do ambiente, foi apresentado um protótipo do motor *Crawler* implementado totalmente em Java. A implementação do protótipo foi facilitada com a utilização de diversas ferramentas de apoio publicamente disponíveis, como *yamlbeans*, para validação e interpretação da representação textual da linguagem *Crawl*, baseada no formato de serialização de dados *Yaml*; *jclouds*, para invocação dos serviços de nuvem de forma independente de provedor; e *JSch*, para acesso remoto às máquinas virtuais criadas na nuvem.

O próximo capítulo descreve alguns experimentos de avaliação de desempenho realizados em dois provedores de nuvem comerciais, com o apoio do protótipo do motor *Crawler* apresentado neste capítulo.

Capítulo 5

Experimentos

Este capítulo descreve uma série de experimentos realizados com o intuito de ilustrar o uso, funcionalidades e benefícios do ambiente *Cloud Crawler*, em particular, do protótipo apresentado no capítulo anterior, para apoiar a execução de diversos cenários de avaliação de desempenho de aplicações em nuvens de infraestrutura. Inicialmente, a aplicação escolhida como alvo dos experimentos é apresentada. Na sequência, descreve-se a metodologia utilizada na execução dos experimentos. Por fim, os resultados obtidos são apresentados e analisados, seguidos de uma discussão sobre as suas consequências e implicações.

5.1 Aplicação Alvo

Antes da realização da parte experimental deste trabalho, foram estudadas diversas opções de *benchmark* disponíveis publicamente como possíveis aplicações candidatas a terem seu desempenho avaliado na nuvem. Um requisito importante para essa escolha era que a aplicação disponibilizada pelo *benchmark* estivesse apta a ser facilmente exposta a diferentes níveis de carga de trabalho, dada a grande variedade de recursos computacionais e modelos de precificação oferecidos pelos provedores de nuvem comerciais atuais. Além disso, era importante que a aplicação escolhida apresentasse uma arquitetura similar àquelas de aplicações comumente implantadas na nuvem, de modo a permitir que os resultados obtidos com os experimentos representassem perfis de desempenho próximos aos de aplicações reais.

Nesse contexto, foi escolhida a aplicação *Olio*,¹ originalmente desenvolvida como parte do projeto *Cloudstone* (SOBEL *et al.*, 2008), previamente discutido na seção 3.2.2.3. O *Cloudstone* disponibiliza um conjunto de ferramentas de código aberto voltadas para apoiar a

¹<http://incubator.apache.org/projects/olio.html>

realização de testes de desempenho de aplicações em nuvens de infraestrutura. Entre essas ferramentas encontra-se a *Olio*, uma aplicação web 2.0 que implementa uma rede social na forma de um calendário de eventos distribuídos. As principais funcionalidades implementadas pela aplicação são: cadastro de usuários; registro de eventos de interesse pelos usuários cadastrados, com os usuários podendo fornecer informações adicionais sobre os eventos publicados, como hora, endereço e fotos do local; e postagem de comentários acerca de eventos previamente registrados.

5.1.1 Arquitetura da Aplicação

A *Olio* foi implementada em duas versões funcionalmente equivalentes, disponíveis para as plataformas *Ruby on Rails* (BACHLE; KIRCHBERG, 2007) e *PHP* (PHP, 2012), respectivamente. A arquitetura da versão implementada para a plataforma *Ruby on Rails*, utilizada neste trabalho, segue a típica organização em três camadas — web, aplicação e dados — das aplicações web tradicionais, conforme ilustra a Figura 5.1.

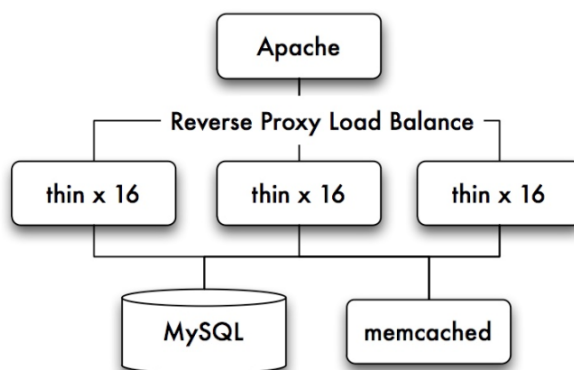


Figura 5.1: Arquitetura da aplicação *Olio* implementada na plataforma *Ruby on Rails* (extraído de (SOBEL *et al.*, 2008)).

Na camada web, a *Olio* pode ser configurada para utilizar um servidor web *Apache*² ou *Nginx*,³ que atua como *proxy* reverso e balanceador de carga para as instâncias do servidor da camada de aplicação, *Thin*.⁴ Já na camada de dados, a *Olio* pode ser configurada para utilizar um servidor de banco de dados *MySQL*⁵ ou *Postgres*.⁶ Essa arquitetura em camadas permite que os componentes da *Olio* sejam fisicamente separados em diferentes máquinas, o que possibilita a investigação de diversas opções de configuração e implantação da aplicação no ambiente de nuvem.

²<http://www.apache.org/>

³<http://nginx.org/en/>

⁴<http://code.macournoyer.com/thin/>

⁵<http://www.mysql.com/>

⁶em <http://www.postgresql.org/>

5.1.2 Gerador de Carga

O projeto *Cloudstone* também disponibiliza uma ferramenta de geração de carga, denominada *Faban*,⁷ a qual pode ser utilizada para simular diferentes níveis de demanda para uma aplicação web. O *Faban* permite a definição de fluxos de trabalho completos, que podem ser compostos por diversas requisições *HTTP*. No caso da aplicação *Olio*, por exemplo, o fluxo de trabalho definido com o *Faban* deve conter as requisições correspondentes às invocações das operações de interesse da aplicação, como cadastrar usuário ou cadastrar evento. Essas requisições são então enviadas pelo *Faban* ao servidor web da aplicação, para que sejam processadas e, se for o caso, respondidas.

Buscando simular o comportamento de usuários “reais”, o *Faban* gera sequências de fluxos de trabalho de forma estocástica, através da utilização de cadeias de *Markov* (MEYN *et al.*, 2009). Esses fluxos são intercalados por intervalos sem atividade, de modo a também simular o “tempo de pensamento” (*thinking time*) do usuário.

Outra característica importante do *Faban* é que ele pode ser configurado com parâmetros de qualidade (por exemplo, o tempo máximo para a realização de uma determinada operação da aplicação), que permitem avaliar o desempenho da aplicação a partir de critérios bem definidos (por exemplo, o percentual de operações da aplicação cujos tempos de resposta atenderam aos limites máximos pré-estabelecidos).

Por fim, o *Faban* permite configurar a geração dos fluxos de trabalho a partir de diferentes máquinas, todas coordenadas por um mesmo servidor. Isso possibilita o envio de uma grande quantidade de requisições em paralelo, oferecendo alta escalabilidade à ferramenta.

5.2 Metodologia

5.2.1 Objetivos e Hipóteses

Os experimentos realizados tiveram como objetivo investigar potenciais variações no desempenho da aplicação escolhida, *Olio*, quando submetida a diferentes níveis de demanda, considerando diferentes configurações de implantação de seus componentes na nuvem, e utilizando diferentes perfis de recursos computacionais oriundos de diferentes provedores.

Outro objetivo era investigar as potenciais diferenças entre os provedores de nuvem em termos dos preços cobrados pelos recursos utilizados e seu impacto no desempenho da

⁷<http://java.net/projects/faban>

aplicação.

A principal hipótese considerada foi de que recursos computacionais com capacidades (e preços) diferentes, oriundo de um mesmo provedor, implicariam em variações significativa no desempenho da aplicação.

Outra hipótese foi de que recursos computacionais com capacidades similares, mas oriundo de diferentes provedores, poderiam oferecer diferentes resultados em termos de custo/desempenho, haja vista que os provedores utilizam infraestruturas computacionais e modelos de precificação também diferentes.

5.2.2 Provedores de Nuvem Escolhidos

Para execução dos experimentos, foram escolhidos os provedores *Amazon EC2* e *Rackspace*, ambos já descritos na seção 2.4. Essa escolha foi devido ao fato de que esses provedores estão entre os mais utilizados atualmente, sendo considerados referências na oferta de serviços de nuvem do modelo IaaS (CHAMBERLIN; LEONG, 2012).

No caso do provedor *Amazon EC2*, foram utilizados recursos da região de disponibilidade *US East*, localizada no estado americano da Virgínia, que possui os menores preços por hora de uso dentre todas as regiões oferecidas por esse provedor.⁸ Já no caso do provedor *Rackspace*, não há diferença nos valores praticados pelas regiões de disponibilidade. Além disso, no *Rackspace* a escolha da região onde os recursos serão fisicamente alocados é transparente para o cliente da nuvem,⁹ que tem como garantia que todos os seus recursos ficarão hospedados na mesma região de disponibilidade.

5.2.3 Descrição dos Cenários de Avaliação

Com o intuito de limitar o número total de configurações de implantação da aplicação a ser investigado, em cada um dos dois provedores escolhidos foram fixados os perfis das duas máquinas virtuais utilizadas para executar o gerador de carga (no caso, *Faban*) e o servidor de banco de dados (no caso, *MySQL*), variando-se apenas o número e o perfil das máquinas virtuais alocadas aos componentes das camadas web e de aplicação (no caso, os servidores *Nginx* e *Thin*, respectivamente). Esse dois últimos componentes foram executados concorrentemente, em uma mesma máquina virtual, com o servidor *Nginx* servindo de *proxy* reverso e balanceador de carga para as múltiplas instâncias do servidor de *Thin*

⁸<http://aws.amazon.com/ec2/pricing/>

⁹http://www.rackspace.com/knowledge_center/article/what-is-ord-and-dfw

— o número exato de instâncias criadas para o servidor *Thin* era determinado de acordo com o perfil da máquina virtual alocada à sua execução. Dessa forma, era possível ajustar a escalabilidade da aplicação tanto horizontalmente, alterando-se o número de máquinas virtuais alocadas aos servidores *Nginx* e *Thin*, quanto horizontalmente, alterando-se o perfil das máquinas alocadas a esses dois componentes bem como ajustando-se os seus parâmetros de configuração de acordo com cada perfil.

A Figura 5.2 ilustra um exemplo de uma configuração de implantação dos componentes da aplicação *Olio* na nuvem, definida em consonância com a estratégia de implantação descrita acima. Note, nesse exemplo, que foram criadas cinco máquinas virtuais na nuvem, sendo uma delas alocada ao gerador de carga (*Faban*), outra ao servidor de banco de dados (*MySQL*), e três aos componentes das camadas web (*Nginx*) e de aplicação (*Thin*). Note, também, que o servidor *Nginx* e as múltiplas instâncias do servidor *Thin* são executados concorrentemente, em cada uma das três máquinas virtuais onde esses componentes foram hospedados. O fato do gerador de carga também estar localizado dentro da nuvem implica que os tempos de resposta medidos para os clientes da aplicação não consideram a latência da rede no acesso ao provedor da nuvem. Essa decisão tem a limitação de não produzir resultados realistas do ponto de vista de um cliente “real” da nuvem. Por outro lado, ao excluir a latência da rede, os resultados refletem melhor as diferenças de desempenho da aplicação causadas pela utilização de diferentes perfis de recurso da nuvem e estratégias de implantação.

Os perfis e quantidade de recursos computacionais alocados aos componentes da *Olio* em cada um dos dois provedores escolhidos foram os seguintes. Para o gerador de carga e o servidor de banco de dados, foram alocados um mesmo perfil de recurso em cada provedor: *c1.xlarge*, no provedor *Amazon EC2*, e *flavor 5*, no provedor *Rackspace*. Esses dois perfis são suficiente para atender com folga os níveis de demanda investigados neste trabalho, e foram determinados com base nas avaliações de desempenho realizadas para esta mesma aplicação na nuvem da Amazon no âmbito do projeto *Cloudstone*. Já para os componentes da camada web e de aplicação, foram alocadas e avaliadas máquinas virtuais com diversos perfis, cobrindo grande parte do espectro de tipos de máquinas virtuais oferecidos pelos dois provedores, conforme descritos nas Tabelas 2.1 e 2.2 (ver seção 2.4).

Inicialmente, foi investigada apenas a escalabilidade vertical da aplicação. Para isso, foram avaliados, de forma individual, oito perfis de máquinas virtuais oferecidos pelo provedor *Amazon EC2* (*t1.micro*, *m1.small*, *c1.medium*, *m1.large*, *m2.xlarge*, *c1.xlarge*, *m1.xlarge* e *m2.4xlarge*), e seis perfis oferecidos pelo provedor *Rackspace* (*flavors 1-6*). As configurações

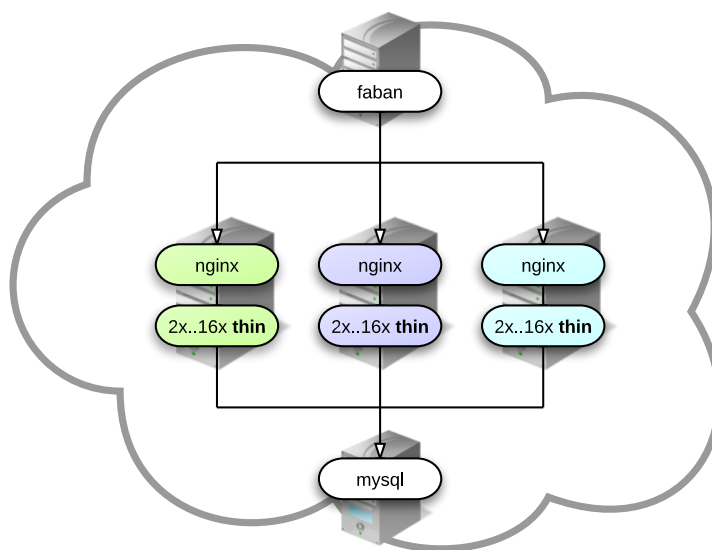


Figura 5.2: Exemplo de implantação dos componente da aplicação *Olio* na nuvem.

dos componentes *Nginx* e *Thin* foram então ajustadas para refletir as capacidades de memória e processamento disponíveis em cada um desses perfis. Por exemplo, em perfis com maior poder de processamento e/ou espaço de memória, aumentava-se o número de *workers* (*threads*) do servidor *Nginx*, bem como o número de instâncias do servidor *Thin*.

Em um segundo momento, investigou-se também a escalabilidade horizontal da *Olio*. Nesse caso, alguns perfis de recurso foram selecionados de cada provedor, dentre aqueles avaliados individualmente na primeira leva de experimentos. Novas configurações de implantação da aplicação foram então definidas, alocando-se múltiplas máquinas virtuais de cada um dos perfis escolhidos aos componentes das camadas web e de aplicação, até o máximo de três máquinas virtuais por perfil. Em seguida, essas novas configurações foram avaliadas e tiveram seus resultados comparados com os resultados obtidos nas configurações com apenas uma máquina virtual nas camadas web e de aplicação.

Em todos os cenários investigados, a aplicação foi submetida a dois níveis de demanda. Cada nível de demanda correspondia a determinadas quantidades de usuários acessando a aplicação concorrentemente, cujas requisições eram geradas e enviadas ao servidor web da aplicação pelo gerador de carga, *Faban*. No primeiro nível, denominado *demandbaixa*, o número de usuários concorrentes acessando a aplicação variava entre 25 e 150, enquanto no segundo nível, denominado *demandmoderada*, o número de usuários concorrentes variava entre 200 e 800. Durante os experimentos, a aplicação era executada três vez para

cada quantidade de usuários de cada um dos níveis de demanda avaliados em cada cenário. Cada execução da aplicação durava cerca de 800 segundos, e era considerada realizada com sucesso quando pelo menos 90% dos tempos de resposta medidos pelo gerador de carga estivessem dentro dos tempos máximos estabelecidos para as respectivas operações oferecidas pela aplicação. Nos experimentos, esses limites de tempo foram definidos como os mesmos valores previamente utilizados para avaliar o desempenho da aplicação *Olio* no projeto *Cloudstone* (SOBEL *et al.*, 2008).

5.2.4 Representação dos Cenários em *Crawl*

A Figura 5.3 mostra a representação em *Crawl* contendo a especificação dos cenários de avaliação da aplicação *Olio* nas nuvens *Amazon EC2* e *Rackspace*, conforme descritos na subseção anterior.

A parte inicial da especificação (linhas 1-3) define uma entidade do tipo *Benchmark*, representando o contexto global da avaliação, e seus atributos **rounds** e **className**. Esse dois atributos representam, respectivamente, o número de vezes (no caso, 3) que a aplicação deverá ser executada para cada unidade de carga de trabalho especificada em cada cenário, e o nome da classe Java (no caso, *OlioExecution*) que implementa a interface genérica utilizada pelo motor *Crawler* para interagir com os componentes da aplicação na nuvem (ver seção 4.3.2).

Em seguida, nas linhas 5-6, é declarada a inclusão de um arquivo externo de nome “crawl-lib.yml”. Esse arquivo contém a definição de entidades não específicas da aplicação, como provedores de nuvens e seus respectivos perfis de recursos. Mais adiante, iniciando na linha 8, é definido um conjunto de propriedades globais do *benchmark*, no caso, a lista de perfis de recursos (máquinas virtuais) que serão utilizados na definição dos cenários de avaliação. Note que cada perfil de recurso é definido como sendo do tipo *OlioInstanceType*, que corresponde à extensão da entidade *VirtualMachineType* criada especificamente para a aplicação *Olio*. É nessa extensão que são declarados os atributos de configuração específicos dos componentes da aplicação, no caso, o número de *workers* do servidor *Nginx* (atributo **workers**) e o número de instâncias do servidor *Thin* (atributo **thinServers**).

A partir da linha 25, são definidos os recursos globais do *benchmark*, comuns aos diversos cenários. Nas linhas 26-31, define-se o recurso que será alocado ao gerador de carga da *Olio*, no caso, *Faban*. Note que esse recurso foi definido como tendo o ID 20408853, que equivale ao ID do recurso correspondente previamente criado na nuvem, e o perfil *olio_m1_large*, definido na linha 9. Da mesma forma, nas linhas 33-38, define-se o recurso

1	!benchmark	43	values: 25,50,75,100,125,150
2	rounds: 3	44	
3	className: OlioExecution	45	- &moderate !workload
4		46	id: 2
5	- !include	47	values: 200,300,400,500,600,700,800
6	file: crawllib.yml	48	
7		49	scenarios:
8	properties:	50	- !foreach
9	- &olio_m1_large !OlioInstanceType	51	list: *olio_m1_small,
10	virtualMachineType: *5_m1_large	52	*olio_m1_xsmall,
11	properties:	53	*olio_m1_large
12	- workers: 2	54	count: 1
13	- thinServers: 2	55	var: type
14	- &olio_m1_small !OlioInstanceType	56	statement:
15	virtualMachineType: *2_m1_small	57	- !scenario
16	properties:	58	id: \${count}
17	- workers: 3	59	workloads:
18	- thinServers: 3	60	- *low
19	- &olio_m1_xsmall !OlioInstanceType	61	- *moderate
20	virtualMachineType: *3_m1_small	62	metrics:
21	properties:	63	- *responseTime
22	- workers: 4	64	application: !OlioApplication
23	- thinServers: 6	65	database: *mysql
24		66	driver: *faban
25	virtualMachines:	67	web:
26	- &faban !virtualMachine	68	- !virtualMachine
27	id: 20408853	69	id: 20400657
28	type: *olio_m1_large	70	name: THIN SERVER
29	name: DRIVER	71	type: \${type}
30	components:	72	name: THIN SERVER
31	- *fabanApp	73	components:
32		74	- *railsApp
33	- &mysql !virtualMachine	75	- *tomcatApp
34	id: 20406221	76	- *nginxApp
35	type: *olio_m1_large	77	- &olioApp !component
36	name: MYSQL	78	id: 6
37	components:	79	config:
38	- *mysqlApp	80	- &olioScript !scriptlet
39		81	id: 2
40	workloads:	82	scripts:
41	- &low !workload	83	- sudo/home/db.sh
42	id: 1	84	\${privateIP}

Figura 5.3: Especificação em *Crawl* descrevendo os cenários de avaliação da aplicação *Olio* na nuvem.

que será alocado ao servidor de banco de dados (no caso, *MySQL*), e que possui o mesmo perfil do recurso alocado ao gerador de carga. Por fim, nas linhas 40-47, são definidas duas entidades do tipo *Workload*, que representam os dois níveis de demanda (baixa e moderada) a que a *Olio* será submetida através do gerador de carga *Faban*. Note que as duas entidades são definidas através de uma lista de valores numéricos, representando as quantidades de usuários concorrentes que o *Faban* deverá simular dentro de cada faixa de demanda.

A definição dos cenários propriamente dita começa a partir da linha 49. Observa-se aqui o uso da entidade *!foreach*, introduzida pela linguagem *Crawl* para permitir a definição de laços de iteração. O primeiro atributo do laço (linhas 51-53) contém a lista de perfis dos recursos (no caso, *olio_m1_large*, *olio_m1_small* e *olio_m1_xsmall*) que serão utilizados

para hospedar os componentes das camadas web e de aplicação da *Olio*. Os atributos opcionais **count** (linha 54) e **var** (linha 55) definem, respectivamente, um contador de incremento unitário e uma variável de referência, à qual serão atribuídos os valores da lista de perfis do laço. O último atributo do laço é o **statement** (linha 56), que contém a definição das demais entidades do cenário. Note que, na linha 58, há uma referência ao atributo **count**, a qual será substituída pelo valor 1 na primeira iteração do laço, pelo valor 2 na segunda iteração, e assim sucessivamente, até todos os perfis da lista de perfis definida como atributo do laço terem sido utilizados. Os valores referentes a cada perfil da lista irão substituir, um a um, em cada iteração do laço, a referência a esse atributo definida na linha 71, através dos delimitadores `$[` e `]`. Por fim, nas linhas 82-84, define-se um *script* de configuração que será executado logo após a inicialização do recurso que está sendo definido. Note que esse *script* faz referência ao atributo **privateIP**, o qual será substituído, em tempo de inicialização, pelo valor do IP do recurso virtual onde está hospedado o servidor de banco de dados. Conforme explicado para o exemplo da aplicação wiki, descrita no capítulo anterior, o papel desse *script* é o informar aos componentes das camadas web e de aplicação a localização do servidor de banco de dados, cujo endereço de rede só é conhecido após a sua inicialização na nuvem.

5.3 Resultados

Os resultados obtidos em cada um dos cenários avaliados, nos dois provedores de nuvem, estão sumarizados nas Tabelas 5.1 e 5.2. Nessas tabelas, as duas primeiras colunas descrevem, respectivamente, os perfis e os preços por hora de uso de cada um dos recursos avaliados (individualmente ou em conjunto com outros recursos de mesmo perfil) nos diversos cenários executados em cada um dos dois provedores. Já as outras colunas representam o desempenho da aplicação para cada nível de demanda, considerando cada perfil de recurso avaliado, utilizando uma notação visual especificamente criada para este fim. Nessa notação, um círculo totalmente preenchido indica que aplicação atendeu plenamente a demanda com aquele perfil de recurso; por sua vez, um círculo com outro círculo circunscrito indica que a aplicação atendeu parcialmente a demanda; já um círculo com um pequeno ponto em seu centro indica que a aplicação atendeu apenas esporadicamente a demanda; finalmente, um círculo vazio indica que a aplicação não atendeu a demanda para aquele perfil.

Nos experimentos, os diferentes graus de sucesso da aplicação, representados visualmente pela notação descrita acima, foram estabelecidos de acordo com o número de execuções, de

Recurso		Demanda (# usuários concorrentes)													
Perfil	Preço (\$/h)	Baixa							Moderada						
		25	50	75	100	125	150	200	300	400	500	600	700	800	
Configurações com um único recurso nas camadas web e de aplicação															
t1.micro	0,02	●	○	○	○	○	○	○	○	○	○	○	○	○	○
m1.small	0,085	●	○	○	○	○	○	○	○	○	○	○	○	○	○
c1.medium	0,17	●	●	●	●	●	●	○	○	○	○	○	○	○	○
m1.large	0,34	●	●	●	●	●	⊙	○	○	○	○	○	○	○	○
m2.xlarge	0,50	●	●	●	●	●	●	⊙	○	○	○	○	○	○	○
c1.xlarge	0,68	●	●	●	●	●	●	●	●	⊙	○	○	○	○	○
m1.xlarge	0,68	●	●	●	●	●	●	●	○	○	○	○	○	○	○
m2.4xlarge	2,00	●	●	●	●	●	●	●	●	⊙	○	○	○	○	○
Configurações com múltiplos recursos nas camadas web e de aplicação															
c1.medium (x2)	0,34	●	●	●	●	●	●	●	●	○	○	○	○	○	○
c1.medium (x3)	0,51	●	●	●	●	●	●	●	●	●	●	●	●	○	○
Legenda															
●	Aplicação atendeu plenamente a demanda (3/3 execuções com sucesso)														
⊙	Aplicação atendeu parcialmente a demanda (2/3 execuções com sucesso)														
⦿	Aplicação atendeu esporadicamente a demanda (1/3 execuções com sucesso)														
○	Aplicação não atendeu a demanda (0/3 execuções com sucesso)														

Tabela 5.1: Resultados da avaliação de desempenho da aplicação *Olio* na nuvem *Amazon EC2*.

um total de três realizadas, em que a aplicação foi considerada bem sucedida pelo gerador de carga para cada nível de demanda e perfil de recurso avaliados. Assim, se aplicação foi bem sucedida nas três execuções, considera-se que ela atendeu plenamente aquela demanda com aquele perfil de recurso. Se a aplicação foi bem sucedida em duas das três execuções, considera-se que ela atendeu parcialmente a demanda, e assim sucessivamente.

Os resultados experimentais obtidos em cada um dos dois provedores são analisados nas próximas subseções.

5.3.1 Resultados Obtidos na Nuvem *Amazon EC2*

De acordo os resultados apresentados na Tabela 5.1, pode-se concluir que com até 25 usuários concorrentes, que correspondem ao menor valor de demanda considerado, os componentes das camadas web e de aplicação da *Olio* poderiam ser hospedados em um único recurso de perfil *t1.micro*, o de menor capacidade entre aqueles disponibilizados pela *Amazon EC2*, ao custo módico de US\$ 0,02 por hora de uso. Porém, com esse perfil de recurso, a aplicação só consegue atender a demanda esporadicamente (nos experimentos, uma única vez), o que demonstra a instabilidade dos serviços da *Amazon* para recursos com baixa capacidade computacional. Nesse sentido, utilizar um recurso com um perfil imediatamente superior, no caso, *m1.small*, ofereceria uma estabilidade bem maior, a um custo ainda relativamente pequeno (menos de 10 centavos de dólar por hora de uso).

Considerando a faixa de demanda de 50 até 150 usuários concorrentes, observa-se que

a aplicação satisfaz plenamente seus requisitos de qualidade com praticamente todos os perfis de recursos analisados. A exceção fica com o recurso de perfil *m1.large*, com o qual a aplicação atendeu a demanda apenas esporadicamente. Em vista da grande variedade de preços entre os diferentes perfis de recurso oferecidos pela *Amazon EC2*, fica claro que uma escolha inadequada nessa faixa de demanda pode implicar em um custo muito acima do realmente necessário. Por exemplo, com um recurso de perfil *c1.medium* a aplicação apresentaria um desempenho comparável (e até ligeiramente superior) ao que seria obtido com um recurso de perfil *m1.large*, embora o segundo custe o dobro do preço do primeiro. Nesse sentido, vale destacar o desempenho observado para o perfil *c1.medium*, que ofereceu a melhor relação custo/benefício para esse nível de demanda.

Estendendo a análise para a faixa de demanda moderada, percebe-se que o custo inicial para hospedar os componentes das camadas web e de aplicação da *Olio* sobe para US\$ 0,5 por hora, que é o preço cobrado pela *Amazon EC2* para um recurso de perfil *m2.xlarge*. Porém, esse perfil de recurso não apresentou um desempenho satisfatório em todas as execuções da aplicação. Portanto, para uma maior estabilidade, o perfil *c1.xlarge* seria recomendado como o de menor custo. É possível notar que recursos com perfis diferentes mas com o mesmo preço podem apresentar desempenhos bem diferentes, como é o caso dos recursos com perfil *c1.xlarge* e *m1.xlarge*. A razão dessa aparente discrepância entre os preços e os desempenhos dos recursos é que alguns perfis são mais indicados para aplicações com determinadas características (por exemplo, com maior necessidade de memória ou de processamento). No caso da aplicação *Olio*, por exemplo, vê-se claramente que seus componentes das camadas web e de aplicação se beneficiam mais de recursos com maior capacidade de processamento do que de recursos com maior capacidade de memória (note que a aplicação apresenta um melhor desempenho com o recurso de perfil *c1.xlarge*, que tem 20 unidades de processamento e apenas 7 GB de RAM, do que com um recurso de perfil *m1.xlarge*, que tem apenas 8 unidades de processamento mas 15 GB de RAM, embora ambos os perfis tenham o mesmo preço). Assim, um recurso de perfil mais caro não necessariamente vai oferecer um desempenho melhor do que outro de perfil mais barato para todos os tipos de aplicação. Isso pode ser facilmente constatado analisando o desempenho da *Olio* para os perfis *c1.xlarge* e *m2.4xlarge*. Os dois perfis propiciam desempenhos muito parecidos à aplicação, com uma ligeira vantagem para o primeiro, embora o segundo custe quase três vezes mais.

Por fim, analisam-se agora os resultados obtidos nos cenários com múltiplos recursos alocados aos componentes das camadas web e de aplicação da *Olio*, apresentados na parte inferior da Tabela 5.1. Esses resultados mostram que com dois recursos de perfil

Recurso		Demanda (# usuários concorrentes)													
Perfil	Preço (\$/h)	Baixa						Moderada							
		25	50	75	100	125	150	200	300	400	500	600	700	800	
Configurações com um único recurso nas camadas web e de aplicação															
1	0,015	●	⊙	○	○	○	○	○	○	○	○	○	○	○	○
2	0,03	●	●	●	○	○	○	○	○	○	○	○	○	○	○
3	0,06	●	●	●	●	●	⊙	○	○	○	○	○	○	○	○
4	0,12	●	●	●	●	●	●	●	○	○	○	○	○	○	○
5	0,24	●	●	●	●	●	●	●	●	○	○	○	○	○	○
6	0,48	●	●	●	●	●	●	●	●	○	○	○	○	○	○
Configurações com múltiplos recursos nas camadas web e de aplicação															
5 (x2)	0,48	●	●	●	●	●	●	●	●	●	●	○	○	○	○
5 (x3)	0,72	●	●	●	●	●	●	●	●	●	●	●	●	●	⊙
Legenda															
●	Aplicação atendeu plenamente a demanda (3/3 execuções com sucesso)														
⊙	Aplicação atendeu parcialmente a demanda (2/3 execuções com sucesso)														
○	Aplicação atendeu esporadicamente a demanda (1/3 execuções com sucesso)														
○	Aplicação não atendeu a demanda (0/3 execuções com sucesso)														

Tabela 5.2: Resultados da avaliação de desempenho da aplicação *Olio* na nuvem *Rackspace*.

c1.medium, a um custo total de US\$ 0,34 por hora de uso, a aplicação consegue atender plenamente uma demanda de até 300 usuários concorrentes. A partir daí, a aplicação só consegue atender plenamente a demanda se utilizar três recursos com esse perfil. Vale notar que a aplicação não consegue atender plenamente uma demanda de mais de 300 usuários com nenhum dos perfis de recursos avaliados, quando alocados isoladamente. Além disso, embora a aplicação consiga atender plenamente uma demanda de até 300 usuários concorrentes utilizando apenas um recurso de perfil *c1.xlarge* ou *m2.4xlarge*, com dois recursos de perfil *c1.medium* ela consegue atender esse mesmo nível de demanda a um custo significativamente menor. Esses resultados confirmam a importância de se investigar tanto a escalabilidade vertical quanto a escalabilidade horizontal de uma aplicação na nuvem, de modo a identificar as suas configurações de implantação que melhor satisfaçam as restrições de custo e/ou desempenho de seus desenvolvedores.

5.3.2 Resultados Obtidos na Nuvem *Rackspace*

Os mesmos cenários de avaliação da aplicação *Olio* executados na nuvem *Amazon EC2* foram novamente executadas na nuvem *Rackspace*, mas agora considerando apenas seis perfis (ou *flavors*, na terminologia do *Rackspace*) de recursos computacionais. Os resultados destes experimentos estão representados na Tabela 5.2.

Observando os dados dessa tabela, é possível constatar que, para os níveis mais baixos da demanda, com até 50 usuários concorrentes, os componentes das camadas web e de aplicação da *Olio* poderiam ficar hospedadas em um recurso com o perfil *flavor* 1, o de

menor capacidade dentre aqueles disponibilizados pelo *Rackspace*, a um custo de apenas US\$ 0,015 por hora de uso. Porém, com esse perfil de recurso, a aplicação só atendeu a demanda uma única vez, muito provavelmente devido às severas limitações deste perfil em termos de memória e processamento, como também de uma possível sobrecarga da máquina hospedeira. Para demandas de até 75 usuários concorrentes, um recurso com o perfil *flavor* 2 foi suficiente para a aplicação satisfazer plenamente seus requisitos de qualidade, ao dobro do custo do perfil anterior. Para o restante dessa faixa de demanda, indo até 150 usuários concorrentes, a aplicação executou com sucesso utilizando um recurso de perfil *flavor* 3, que conseguiu dobrar a quantidade de usuários atendidos em relação ao perfil *flavor* 2, embora custando o dobro do preço.

Analizando os resultados para a faixa de demanda moderada, percebe-se que o custo inicial para hospedar os componentes das camadas web e de aplicação da *Olio* sobe para US\$ 0,12 por hora, que é o preço de um recurso de perfil *flavor* 4, o de menor capacidade com o qual a aplicação consegue satisfazer seus requisitos de qualidade sob a demanda de 200 usuários concorrentes. Já com 300 usuários concorrentes, o perfil mais barato que consegue atender a demanda da aplicação é o *flavor* 5. Até este ponto, a cada novo incremento no perfil do recurso, a aplicação, em média, dobrava o número de usuários plenamente atendidos em relação à demanda atendida com o perfil anterior. Esses ganhos foram, até certo ponto, proporcionais às diferenças de preços entre os perfis, que dobram de valor a cada incremento. Essa tendência, porém, não se repetiu com o incremento do perfil *flavor* 5 para o perfil *flavor* 6, com o último mantendo o mesmo nível de desempenho da aplicação obtido com o primeiro. Dessa forma, conclui-se que 300 usuários concorrentes é a demanda máxima que pode ser atendida com os componentes das camadas web e de aplicação da *Olio* sendo alocados a um único recurso da nuvem *Rackspace*.

Níveis de demanda acima de 300 usuários concorrentes só foram atendidos na nuvem *Rackspace* com a escalabilidade horizontal da aplicação, como pode ser observado na parte inferior da Tabela 5.2. Nesses cenários, os componentes das camadas web e de aplicação foram alocados a grupos de 2 e 3 recursos com o perfil *flavor* 5, o de melhor custo/benefício dentre aqueles avaliados individualmente. Utilizando dois recursos desse perfil, a aplicação conseguiu atender satisfatoriamente até 500 usuários concorrentes, a um preço de US\$ 0,48 por hora de uso, o mesmo valor cobrado para o perfil *flavor* 6. Isso significou um aumento de 66% no número de usuários atendidos pela aplicação em relação ao desempenho obtido com o recurso de maior capacidade, mantendo o mesmo custo por hora de uso. Adicionando-se ao grupo um terceiro recurso de perfil *flavor* 5, foi possível atender com sucesso até 800 usuários concorrentes, representando um aumento

de 60% no número de usuários mas de apenas 50% no preço dos recursos.

5.3.3 Discussão

O ambiente *Cloud Crawler* mostrou-se uma valiosa ferramenta de apoio ao planejamento e à execução de experimentos de avaliação de desempenho de aplicações em nuvens de infraestrutura. O ambiente facilita o planejamento dos experimentos no sentido em que oferece uma notação simples, modular e flexível para descrever os cenários de avaliação. Além disso, ao automatizar a execução dos cenários na nuvem, o ambiente diminui o risco da ocorrência de erros devido a alguma falha humana na realização de algum procedimento manual de configuração dos componentes da aplicação ou dos recursos da nuvem, que poderiam comprometer a qualidade dos dados coletados. Esse risco aumenta à medida em que a quantidade e/ou a complexidade dos cenários de avaliação também aumenta, o que reforça a importância da utilização de ferramentas flexíveis e de fácil configuração para automatizar todo o processo de avaliação de desempenho na nuvem, da configuração e execução dos cenários, à coleta dos resultados.

Durante a realização dos experimentos de avaliação da aplicação *Olio*, reportados nas seções anteriores, observou-se que a nuvem *Amazon EC2* apresentava uma significativa flutuação na qualidade de seus serviços. Daí a necessidade de executar a aplicação múltiplas vezes para um mesmo nível de demanda e uma mesma configuração de implantação. Em alguns casos, os mesmo cenários executados com diferença de algumas horas, envolvendo os mesmos parâmetros de configuração, produziam resultados bastantes diferentes. Tal comportamento, que não está limitado à nuvem *Amazon EC2* e pode ocorrer com outros provedores de nuvem, também já foi observado em outras publicações sobre o tema na literatura, como em (WANG; NG, 2010) e (LI *et al.*, 2010). Diferentemente do que aconteceu na *Amazon EC2*, os serviços da nuvem *Rackspace* mostraram-se bem mais estáveis durante todos os experimentos, com o número de usuário atendidos pela aplicação praticamente dobrando a cada alteração do perfil dos recursos para um perfil de nível imediatamente superior, até atingir o limite máximo de usuários atendidos com um único recurso.

Uma possível explicação para esse comportamento mais estável da nuvem *Rackspace*, frente ao observado na *Amazon EC2*, talvez seja que, à época dos experimentos, o nível de sobrecarga da segunda nuvem como um todo era bem maior do que o da primeira. Outra possibilidade é que, na *Amazon EC2*, os recursos virtuais criados para os experimentos tenham sido alocados a servidores físicos bem mais sobrecarregados do que os servidores físicos a que foram alocados os recursos virtuais criados na nuvem *Rackspace*.

Demanda (# usuários)	Configuração recomendada (custo)			
	Amazon EC2		Rackspace	
25	t1.micro	(U\$ 0,02)	1	(U\$ 0,015)
50	c1.medium	(U\$ 0,17)	1	(U\$ 0,015)
75	c1.medium	(U\$ 0,17)	2	(U\$ 0,03)
100	c1.medium	(U\$ 0,17)	3	(U\$ 0,06)
125	c1.medium	(U\$ 0,17)	3	(U\$ 0,06)
150	c1.medium	(U\$ 0,17)	3	(U\$ 0,06)
200	c1.medium (x2)	(U\$ 0,34)	4	(U\$ 0,12)
300	c1.medium (x2)	(U\$ 0,34)	5	(U\$ 0,24)
400	c1.medium (x3)	(U\$ 0,51)	5 (x2)	(U\$ 0,48)
500	c1.medium (x3)	(U\$ 0,51)	5 (x2)	(U\$ 0,48)
600	c1.medium (x3)	(U\$ 0,51)	5 (x3)	(U\$ 0,72)
700	—		5 (x3)	(U\$ 0,72)
800	—		5 (x3)	(U\$ 0,72)

Tabela 5.3: Configurações recomendadas para os provedores *Amazon EC2* e *Rackspace* capazes de hospedar os componentes das camadas web e de aplicação da *Olio* em cada nível de demanda avaliado.

De uma maneira geral, os experimentos realizados neste trabalho podem ser considerados bem sucedidos à medida em que fornecem dados que ajudam a entender melhor a relação entre os diferentes níveis de demanda de uma aplicação e as diferentes opções de configuração de seus componentes na nuvem, tanto sob o ponto de vista de desempenho quanto de custo. No caso da *Olio*, essa relação pode ser analisada no âmbito de dois provedores de nuvem diferentes, como ilustram a Tabela 5.3 e o gráfico da Figura 5.4. Na Tabela 5.3, é possível observar qual a configuração de menor custo por hora de uso (considerando apenas os recursos necessários para hospedar os componentes das camadas web e de aplicação da *Olio*) de cada um dos dois provedores capaz de atender a cada um dos níveis de demanda avaliados nos experimentos. Esses mesmos valores estão representados graficamente na Figura 5.4. Esses resultados tornam evidente a grande vantagem do provedor *Rackspace* sobre o provedor *Amazon*, com o primeiro oferecendo a melhor relação custo/desempenho em praticamente todos os níveis de demanda investigados, sendo que, na maioria dos casos, a diferença entre ambos passa dos 100% em favor do primeiro.

Obviamente, a escolha de um provedor de nuvem deve levar em conta vários outros fatores, além do preço e do desempenho de seus recursos computacionais, como disponibilidade, segurança, qualidade do meio de comunicação entre os clientes e a nuvem, reputação do provedor, e até mesmo fatores de natureza legal e organizacional (BESERRA *et al.*, 2012).

Por fim, é importante ressaltar que os resultados dos experimentos reportados neste

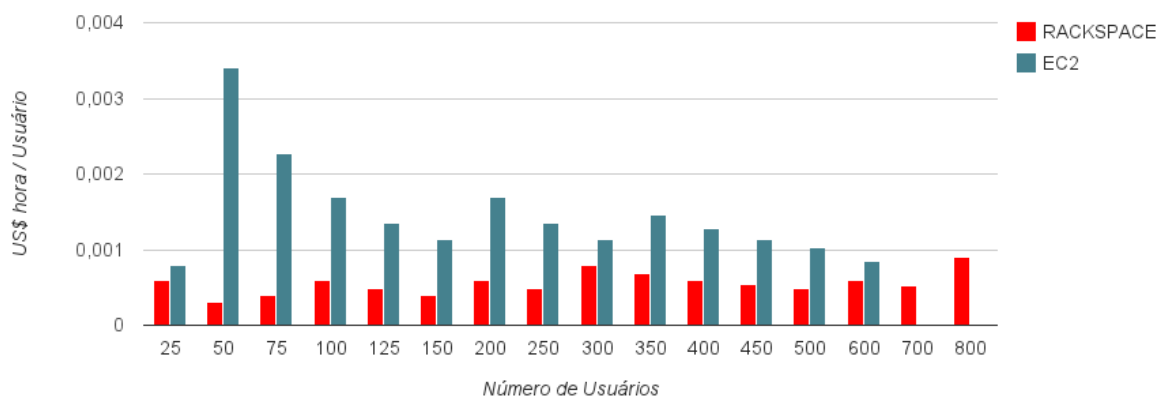


Figura 5.4: Comparação dos custos mínimos por demanda atendida observados para a aplicação *Olio* nas nuvens *Amazon EC2* e *Rackspace*.

capítulo não refletem, necessariamente, o desempenho de uma aplicação web “real” na nuvem. A razão é que, como o gerador de carga também foi executado na nuvem, juntamente com os componentes da aplicação, os tempos de resposta observados pelo gerador de carga não incluem eventuais atrasos de comunicação que poderiam ocorrer para clientes localizados fora da nuvem. Por outro lado, essa decisão de executar o gerador de carga dentro da nuvem, ao eliminar fatores externos à nuvem dos resultados, facilitou a análise e a comparação dos potenciais impactos causados no desempenho da aplicação pelo uso de diferentes configurações de implantação e diferentes perfis de recursos, que constituíam o objetivo principal deste trabalho de pesquisa. Além disso, uma vez que o *benchmark* escolhido representa um perfil de aplicação comumente utilizado atualmente em plataformas de nuvem, os resultados dos experimentos podem ser úteis como um guia para auxiliar no planejamento e na alocação de recursos da nuvem a aplicações com características similares, ajudando, assim, a evitar o excesso ou mesmo a falta de provisionamento de recursos computacionais.

5.4 Resumo do Capítulo

Este capítulo ilustrou o uso do ambiente *Cloud Crawler* para apoiar o planejamento e a execução de experimentos de avaliação de desempenho de aplicações em nuvem de infraestrutura. Os experimentos tiveram como alvo a aplicação de código aberto *Olio* e foram realizados em dois provedores de nuvem *IaaS* comerciais, no caso, *Amazon EC2* e *Rackspace*. Os resultados obtidos a partir dos experimentos mostram que o ambiente proposto pode contribuir de forma efetiva para ajudar os desenvolvedores a entenderem

melhor a relação entre os níveis de demanda e as opções de configuração de suas aplicação na nuvem, tanto em termos de desempenho quanto em termos de custo.

Capítulo 6

Conclusão

Este capítulo apresenta a conclusão deste trabalho, destacando os principais resultados obtidos e sugestões para trabalhos futuros.

6.1 Contribuições e Resultados

Alocar recursos de um determinado perfil a um ou mais componentes de uma aplicação em uma nuvem de infraestrutura, sem conhecer o seu potencial impacto no desempenho da aplicação, não apenas pode significar um provisionamento inadequado dos recursos da nuvem, como também pode se refletir em perdas financeiras para organização ou indivíduo responsável pela aplicação. Portanto, antes de uma aplicação ser implantada na nuvem, é preciso ter um bom entendimento sobre quais perfis de recursos da nuvem deveriam ser alocados a seus componentes, como esses componentes deveriam ser configurados para atender um determinado nível de demanda, e como essa configuração deveria ser alterada para se adequar a possíveis flutuações da demanda. Tomar esse tipo de decisão, definitivamente, não é uma tarefa simples para o cliente da nuvem, que muitas vezes precisa agir rapidamente, e sob pressão.

Nesse contexto, a linguagem *Crawl* e motor de execução *Crawler*, propostos e implementados neste trabalho na forma de um protótipo, como parte do ambiente *Cloud Crawler*, podem oferecer uma importante contribuição para auxiliar os clientes de nuvens de infraestrutura a melhor planejar e alocar os seus recursos de nuvem, conforme as necessidades e demandas específicas de suas aplicações, seja em nuvens públicas, privadas ou híbridas.

Como foi mostrado com os vários experimentos realizados com a aplicação *Olio* nas nuvens *Amazon EC2* e *Rackspace*, o ambiente proposto oferece um alto grau de customização e automação ao processo de avaliação de desempenho de aplicações implantadas em nuvens

de infraestrutura, exigindo um mínimo de esforço de programação por parte do usuário do ambiente.

6.2 Sugestões para Trabalhos Futuros

O desenvolvimento do ambiente *Cloud Crawler* envolve várias outras linhas de pesquisa que atualmente estão sendo investigadas, ou serão alvo de investigação em um futuro próximo. Entre elas, destacam-se:

Avaliação de aplicações de diferentes domínios — essa linha visa realizar novos experimentos de avaliação de desempenho, envolvendo aplicações de diferentes domínios. Atualmente, o ambiente já está sendo utilizado para avaliar o desempenho de uma aplicação legada do domínio acadêmico na nuvem *Amazon EC2*. O objetivo desse experimento é identificar o melhor cenário de implantação da aplicação na nuvem, e se haveria algum ganho de desempenho ao migrar apenas parte dos componentes da aplicação para a nuvem;

Definição de um modelo de dados comum — atualmente, o ambiente não impõe nenhum formato para coleta e armazenamento dos resultados dos experimentos, ficando a critério do usuário do ambiente tal definição. Como a implementação das atividades de coleta, análise e visualização dos resultados podem demandar um esforço considerável por parte do usuário, seria importante que essas atividades fossem, na medida do possível, padronizadas, de modo a facilitar a sua implementação e customização. Nesse sentido, a definição de um modelo de dados comum para armazenamento dos resultados dos experimentos seria uma importante contribuição, uma vez que as ferramentas de coleta, análise e visualização desses dados poderiam ser mais facilmente reutilizadas em diferentes contextos e aplicações;

Implantação automática da aplicação na nuvem — na sua versão atual, o ambiente *Cloud Crawler* ainda requer que a aplicação seja manualmente implantada na nuvem, antes da execução automática dos cenários de avaliação. Uma vez que existem diversas soluções para implantação automática de aplicações na nuvem (por exemplo, (OPSCODE, 2012)), a incorporação de uma ferramenta de implantação automática de aplicações ao ambiente seria uma importante contribuição que iria melhorar ainda mais a sua usabilidade e praticidade;

Implementação de uma ferramenta gráfica de edição de cenários — como o modelo conceitual de *Crawl* é independente de sua representação textual, seria possível definir uma nova notação visual para os elementos da linguagem e então disponibilizar uma ferramenta gráfica para a edição e validação dos cenários de avaliação utilizando essa notação. Tal ferramenta poderia exportar as especificações em *Crawl* para a representação atual, baseada em *Yaml*, ou para novas representações a serem propostas e implementadas no futuro;

Otimização e geração automática dos cenários de avaliação — apesar de facilitar a definição de diversos cenários de avaliação e permitir a sua execução automatizada, o *Cloud Crawler* executa apenas o que é definido pelo usuário. Portanto, o sucesso no planejamento da alocação dos recursos da nuvem vai depender da habilidade do usuário na definição dos cenários de avaliação. Desse modo, uma solução que otimize a escolha e a geração automática dos cenários de avaliação — através da criação de estratégias de menor custo ou menor tempo de execução dos cenários, por exemplo — seria uma importante contribuição para apoiar os clientes de nuvens IaaS no planejamento de seus recursos;

Implementação de mecanismos de monitoramento e controle — a versão atual do ambiente não oferece nenhum mecanismo de suporte ao monitoramento do fluxo de execução dos cenários. Como esse fluxo pode ser composto por diversos cenários e demorar horas ou mesmo dias para a sua conclusão, um mecanismo que permita ao usuário visualizar e acompanhar o andamento da execução de cada cenários seria uma importante funcionalidade a ser incorporada ao ambiente. Além disso, caso o usuário decida realizar experimentos com outras configurações, essas modificações poderiam ser feitas em tempo de execução, sem a necessidade de interromper o fluxo de execução em andamento — consequentemente, sem a necessidade de liberar os recursos da nuvem já alocados e configurados.

Referências

- APPENGINE. Google app engine. <http://code.google.com/intl/en/appengine/>. 2012.
- ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A.; KATZ, R.; KONWINSKI, A.; LEE, G.; PATTERSON, D.; RABKIN, A.; STOICA, I. *et al.* Above the clouds: A berkeley view of cloud computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*, Citeseer, 2009.
- ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A.; KATZ, R.; KONWINSKI, A.; LEE, G.; PATTERSON, D.; RABKIN, A.; STOICA, I. *et al.* A view of cloud computing. *Communications of the ACM*, ACM, v. 53, n. 4, p. 50–58, 2010.
- AZURE. Windows azure platform. <http://www.windowsazure.com/>. 2012.
- BACHLE, M.; KIRCHBERG, P. Ruby on rails. *Software, IEEE*, IEEE, v. 24, n. 6, p. 105–108, 2007.
- BARHAM, P.; DRAGOVIC, B.; FRASER, K.; HAND, S.; HARRIS, T.; HO, A.; NEUGEBAUER, R.; PRATT, I.; WARFIELD, A. Xen and the art of virtualization. *SOSP*, 2003.
- BARRETT, D.; SILVERMAN, R.; BYRNES, R. *SSH, the secure shell: the definitive guide*. [S.l.]: O'Reilly Media, 2005.
- BEN-KIKI, O.; EVANS, C.; INGERSON, B. Yaml ain't markup language (yaml) version 1.1. *Working Draft 2008-05*, v. 11, 2001.
- BESERRA, P. V.; CAMARA, A.; XIMENES, R.; ALBUQUERQUE, A. B.; MENDONCA, N. C. Cloudstep: A step-by-step decision process to support legacy application migration to the cloud. In: IEEE COMPUTER SOCIETY. *IEEE 6th International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA 2012)*. [S.l.], 2012.
- BRAY, T.; PAOLI, J.; SPERBERG-MCQUEEN, C.; MALER, E.; YERGEAU, F. Extensible markup language (xml). *World Wide Web Journal*, v. 2, n. 4, p. 27–66, 1997.
- BRODERICK, K.; BAILEY, M.; EASTWOOD, M. *Worldwide Enterprise Server Cloud Computing 2010-2014 Forecast*. [S.l.], 2010. <http://www.idc.com/about/viewpressrelease.jsp?containerId=prUS22440510>.
- CEARLEY, D.; PHIFER, G. Case studies in cloud computing. *Gartner, March*, 2010.
- CHAMBERLIN, T.; LEONG, L. Magic quadrant for managed hosting. *Gartner RAS Core Research*, 2012.

- CLARK, C.; FRASER, K.; HAND, S.; HANSEN, J.; JUL, E.; LIMPACH, C.; PRATT, I.; WARFIELD, A. Live migration of virtual machines. In: USENIX ASSOCIATION. *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. [S.l.], 2005. p. 273–286.
- CLARK, T. *Designing Storage Area Networks: a Practical Reference for Implementing Fibre Channel SANs*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 1999.
- CLARK, T. *Storage Virtualization: Technologies for Simplifying Data Storage and Management*. [S.l.]: Addison-Wesley Professional, 2005.
- CLOUDHARMONY. Cloudharmony benchmarking the cloud. <http://cloudharmony.com/>. 2012.
- CLOUDSLEUTH. Decoding the misteries of the cloud. <https://cloudsleuth.net/>. 2012.
- CROLL, A. Cloud performance from the end user perspective. <http://www.bitcurrent.com/download/cloud-performance-from-the-end-user-perspective/>. 2011.
- CUNHA, M.; MENDONCA, N.; SAMPAIO, A. Avaliação do custo por usuário de uma aplicação de rede social na amazon ec2. In: *WCGA-XI Workshop em Clouds, Grids e Aplicações (SBRC-2011)*. [S.l.: s.n.], 2011.
- CUNHA, M.; MENDONCA, N.; SAMPAIO, A. Investigating the impact of deployment configuration and user demand on a social network application in the amazon ec2 cloud. In: IEEE COMPUTER SOCIETY. *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. [S.l.], 2011. p. 746–751.
- EBS. Amazon elastic block store. <http://aws.amazon.com/ebs/>. 2012.
- EC2. Amazon elastic compute cloud. <http://aws.amazon.com/ec2>. 2012.
- EEMBC. Eembc. <http://www.eembc.org/>. 2012.
- EUCALYPTUS. Eucalyptus the open source cloud plataform. <http://open.eucalyptus.com/>. 2012.
- FONSECA, R.; SIMOES, A. Alternativas ao XML: YAML e JSON. In: *5ª Conferência Nacional em XML, Aplicações e Tecnologias Aplicadas (XATA 2007)*. [S.l.: s.n.], 2007. p. 33–46.
- FOSTER, I.; ZHAO, Y.; RAICU, I.; LU, S. Cloud computing and grid computing 360-degree compared. In: IEEE. *Grid Computing Environments Workshop, 2008. GCE'08*. [S.l.], 2009. p. 1–10.
- FOWLER, M.; SCOTT, K. *UML distilled: a brief guide to the standard object modeling language*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 2000.
- JAIN, R. *The art of computer systems performance analysis*. [S.l.]: John Wiley & Sons, 2008.

- JAYASINGHE, D.; MALKOWSKI, S.; WANG, Q.; LI, J.; XIONG, P.; PU, C. Variations in Performance and Scalability When Migrating n-Tier Applications to Different Clouds. In: IEEE COMPUTER SOCIETY. *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. [S.l.], 2011. p. 73–80.
- JAYASINGHE, D.; SWINT, G.; MALKOWSKI, S.; LI, J.; WANG, Q.; PARK, J.; PU, C. Expertus: A Generator Approach to Automate Performance Testing in IaaS Clouds. In: IEEE COMPUTER SOCIETY. *Cloud Computing (CLOUD), 2012 IEEE International Conference on*. [S.l.], 2012. p. 73–80.
- JSON. Javascript object notation. <http://www.json.org/>. 2011.
- KIVITY, A.; KAMAY, Y.; LAOR, D.; LUBLIN, U.; LIGUORI, A. kvm: the linux virtual machine monitor. In: *Proceedings of the Linux Symposium*. [S.l.: s.n.], 2007. v. 1, p. 225–230.
- KVM. Kernel-based virtual machine. http://www.linux-kvm.org/page/Main_Page/. 2012.
- LI, A.; YANG, X.; KANDULA, S.; ZHANG, M. CloudCmp: Comparing Public Cloud Providers. In: *Internet Measurement Conference*. [S.l.: s.n.], 2010.
- MALKOWSKI, S.; HEDWIG, M.; JAYASINGHE, D.; PU, C.; NEUMANN, D. Cloudxplor: A tool for configuration planning in clouds based on empirical data. In: ACM. *Proceedings of the 2010 ACM Symposium on Applied Computing*. [S.l.], 2010. p. 391–398.
- MELL, P.; GRANCE, T. The nist definition of cloud computing. *Recommendations of the National Institute of Standards and Technology*, NIST, September 2011.
- MEYN, S.; TWEEDIE, R.; GLYNN, P. *Markov chains and stochastic stability*. [S.l.]: Cambridge University Press Cambridge, 2009.
- OCCI. Open cloud computing interface working group, open cloud computing interface. <http://www.occi-wg.org>. 2012.
- OPENNEBULA. Interfaces for cloud consumers. http://opennebula.org/documentation:rel3.4#interfaces_for_cloud_consumers. 2012.
- OPENNEBULA. Open source toolkit for cloud computing. <http://www.opennebula.org>. 2012.
- OPENSTACK. Conceptual architecture. <http://docs.openstack.org/essex/openstack-compute/admin/content/conceptual-architecture.html>. 2012.
- OPENSTACK. openstack. <http://openstack.org/>. 2012.
- OPENSTACK. Selecting a hypervisor. <http://docs.openstack.org/trunk/openstack-compute/admin/content/selecting-a-hypervisor.html>. 2012.
- OPENSTACK. What is openstack? <http://docs.openstack.org/trunk/openstack-compute/admin/content/what-is-openstack.html>. 2012.

- OPSCODE. Chef. <http://www.opscode.com/chef/>. 2012.
- OSTERMANN, S.; IOSUP, A.; YIGITBASI, N.; PRODAN, R.; FAHRINGER, T.; EPEMA, D. A performance analysis of ec2 cloud computing services for scientific computing. *Cloud Computing*, Springer, p. 115–131, 2010.
- OW2. Rubbos: Bulletin board benchmark. <http://jmob.ow2.org/rubbos.html>. 2012.
- PARKHILL, D. *The challenge of the computer utility*. [S.l.]: Addison-Wesley Reading, MA, 1966.
- PHP. Php. <http://www.php.net/>. 2012.
- PRING, B.; BROWN, R. H.; FRANK, A.; HAYWARD, S.; LEONG, L. *Sizing the Cloud; Understanding the Opportunities in Cloud Services*. [S.l.], 2009. <http://www.gartner.com/DisplayDocument?id=914826>.
- RACKSPACE. The rackspace cloud. <http://www.rackspacecloud.com/>. 2012.
- REESE, G. *Cloud Application Architectures*. [S.l.]: O'Reilly Media, Inc., 2009.
- RHOTON, J. *Cloud Computing Explained: Handbook for Enterprise Implementation*. [S.l.]: Recursive Ltd., 2009.
- RIGHTSCALE. Right scale cloud management plataforma. <http://www.rightscale.com/>. 2012.
- S3. Amazon simple storage service. <http://aws.amazon.com/s3>. 2012.
- SALESFORCE. Salesforce. <http://www.salesforce.com/>. 2012.
- SOBEL, W.; SUBRAMANYAM, S.; SUCHARITAKUL, A.; NGUYEN, J.; WONG, H.; PATIL, S.; FOX, A.; PATTERSON, D. Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0. In: CITESEER. *Proc. of CCA*. [S.l.], 2008.
- SPEC. Java virtual machine benchmark 2008. <http://www.spec.org/jvm2008/>. 2008.
- SPEC. Cpu2006. <http://www.spec.org/cpu2006/results/cpu2006.html>. 2012.
- SPEC. Spec. <http://www.spec.org/>. 2012.
- VAQUERO, L.; RODERO-MERINO, L.; CACERES, J.; LINDNER, M. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, ACM, v. 39, n. 1, p. 50–55, 2008.
- VMWARE. Understanding Full Virtualization, Paravirtualization, and Hardware Assist. *white paper*, 2007.
- WANG, G.; NG, T. The impact of virtualization on network performance of amazon ec2 data center. In: IEEE COMPUTER SOCIETY. *INFOCOM, 2010 Proceedings IEEE*. [S.l.], 2010. p. 1–9. ISSN 0743-166X.
- ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, Springer, v. 1, n. 1, p. 7–18, 2010.