Jonatas Adilson Marques

Ferramenta para Diagnóstico de Provisionamento de Recursos em Nuvens IaaS

Joinville

2014

UNIVERSIDADE DO ESTADO DE SANTA CATARINA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Jonatas Adilson Marques

FERRAMENTA PARA DIAGNÓSTICO DE PROVISIONAMENTO DE RECURSOS EM NUVENS IAAS

Trabalho de conclusão de curso submetido à Universidade do Estado de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciência da Computação

Rafael Rodrigues Obelheiro
Orientador

Joinville, Junho de 2014

FERRAMENTA PARA DIAGNÓSTICO DE PROVISIONAMENTO DE RECURSOS EM NUVENS IAAS

Jonatas Adilson Marques

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção do título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo Curso de Ciência da Computação Integral do CCT/UDESC.

Banca Examinadora

Rafael Rodrigues Obelheiro - Doutor (orientador)

Guilherme Piêgas Koslovski - Doutor

Maurício Aronne Pillon - Doutor

Ricardo José Pfitscher - Mestre

Resumo

Uma tecnologia que tem crescido em popularidade e uso atualmente é a computação em nuvem. Neste modelo, provedores de nuvem gerenciam recursos computacionais e alugam a sua utilização como serviços aos clientes. Um dos modelos de serviço em nuvens é infraestrutura como serviço (IaaS), neste o provedor disponibiliza recursos como processador, memória, disco e largura de banda de rede, normalmente associados a máquinas virtuais. Em serviços IaaS, clientes podem reservar os recursos de acordo com suas necessidades, entretanto, identificar a capacidade dos recursos adequada ao propósito do cliente é um desafio. Para facilitar esta identificação (PFITSCHER; PILLON; OBELHEIRO, 2014) propôs um modelo baseado em monitoração para diagnosticar o provisionamento de recursos. Este trabalho apresenta uma ferramenta genérica para realizar o diagnóstico de provisionamento de recursos em nuvens IaaS, que foi instanciada com o modelo de (PFITSCHER; PILLON; OBELHEIRO, 2014), mas que pode ser facilmente adaptada para outros modelos que se baseiem na análise de séries temporais de métricas de desempenho.

Palavras-chaves: Computação em Nuvem, Virtualização, Diagnóstico de Provisionamento

Abstract

A technology that has grown in popularity and use nowadays is cloud computing. In this model, cloud providers manage computing resources and rent the use of these as services to customers. One of the cloud computing service models is Infrastructure as a Service (IaaS), where the provider offers resources such as processor, memory, disk and network bandwidth, typically associated with virtual machines. In IaaS, customers can reserve resources according to their needs, however, identifying the proper capacity for the client's purpose is a challenge. To help in this task (PFITSCHER; PILLON; OBELHEIRO, 2014) proposed a monitoring based model to diagnose resource provision. This paper presents a generic tool to diagnose IaaS clouds resource provisioning, that was instantiated with the model of (PFITSCHER; PILLON; OBELHEIRO, 2014), but that can be easily adapted to other models which are based on analysis of performance metrics time series.

Keywords: Cloud Computing, Virtualization, Provisioning Diagnosis

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

A computação em nuvem, tecnologia de grande popularidade atual, tem sido adotada por organizações que buscam otimizar seus investimentos em infraestrutura para permitir acesso a informação. Essa otimização de investimentos torna-se possível em razão de características como a elasticidade de reserva de recursos e a tarifação por recursos reservados, pois permite que os custos fixos para manter a infraestrutura do cliente sejam transferidos para um provedor (ARMBRUST et al., 2010).

Os recursos em nuvens são oferecidos como serviços e podem se apresentar de três formas: aplicações, plataformas de desenvolvimento ou infraestruturas computacionais (MELL; GRANCE, 2011). Em serviços em forma de infraestrutura computacional, chamados Infrastructure-as-a-Service (IaaS), os recursos dispostos são capacidade de processamento, memória, disco e/ou largura de banda de rede, sendo possível reservar tais recursos segundo a necessidade do cliente (SULEIMAN et al., 2012). (PFITSCHER; PILLON; OBELHEIRO, 2014) apresenta um desafio pertinente a este paradigma, o de provisionar adequadamente os recursos computacionais necessários para atender às demandas dos clientes.

Em uma solução IaaS, cliente e provedor possuem o mesmo objetivo: minimizar seus custos, porém diferem no modo de atingir esse objetivo. O primeiro busca reservar o mínimo de recursos suficientes para obter um desempenho aceitável, enquanto o segundo prima por ter o máximo de seus recursos físicos reservados, desde que isso não venha a comprometer a qualidade de seus serviços. Quando a demanda por recursos é variável, estas estratégias se mostram conflitantes. Se um recurso físico tiver toda a sua capacidade reservada, em casos de picos de carga, este não poderá acomodar a nova demanda sem comprometer a qualidade dos serviços. O caso, entretanto, de um recurso ter capacidade ociosa (reservada mas não utilizada) é de grande interesse ao provedor (PFITSCHER; PILLON; OBELHEIRO, 2014).

O acordo entre cliente e provedor é geralmente realizado através de Service Level Agreements (SLAs). Um SLA define certos indicadores (SLIs - Service Level Indicators) e restrições que tais indicadores devem satisfazer, chamados Service Level Objetives

(SLOs). Desta forma o problema de definir um SLA pode ser reduzido ao problema de encontrar valores adequados para SLOs (SAUVÉ et al., 2005). (PFITSCHER; PILLON; OBELHEIRO, 2014) ressalta que na prática, definir SLOs que garantam o desempenho desejado para as aplicações a serem executadas, a um custo aceitável, não é trivial. Em razão disso, muitos clientes acabam por sub ou superdimensionar os seus recursos.

O trabalho de (PFITSCHER; PILLON; OBELHEIRO, 2014) aponta que quando recursos são subprovisionados, o desempenho aquém do esperado é perceptível ao usuário, mas muitas vezes não é fácil identificar quais recursos precisam de mais capacidade. Esse aponta também que o superprovisionamento é menos perceptível pelo desempenho e mais pelo custo financeiro. E conclui que a ausência de parâmetros que permitam constatar que aplicações similares conseguem obter desempenho satisfatório a um custo mais baixo e a dificuldade em identificar quais SLOs podem ser reduzidos tornam ainda mais complexa a questão, justificando assim o objetivo de seu trabalho: a apresentação de um modelo para facilitar o diagnóstico do provisionamento de recursos.

Para fornecer um diagnóstico do provisionamento de recursos, (PFITSCHER; PILLON; OBELHEIRO, 2014) propôs métricas para a avaliação de processador, memória e rede. Estes recursos foram escolhidos por sua facilidade em serem reservados em monitores de máquinas virtuais atuais, e por serem objeto de SLA em alguns provedores IaaS. Não foi, entretanto, apresentada uma ferramenta que permita a utilização deste modelo por usuários de soluções IaaS.

Medições sobre recursos computacionais são sujeitas a oscilações, assim, a abordagem empregada por (PFITSCHER; PILLON; OBELHEIRO, 2014) foi o uso de séries temporais (sequências de medições realizadas em intervalos de tempo uniformes) e estatísticas como média e coeficiente de variação sobre estas, o que atribui confiabilidade ao modelo. Na indústria, o armazenamento e processamento de séries temporais pode ser realizado através de bancos de dados relacionais ou ferramentas especialistas (RRDTOOL, 2014; DERI; MAINARDI; FUSCO, 2012). Ao escolher uma ferramenta para este propósito deve-se atentar para algumas características como o tamanho (o número de métricas a serem monitoradas), escalabilidade para atualizar um número potencialmente grande de métricas em tempo hábil, e a granularidade (o intervalo entre medições necessário para uma avaliação válida). No caso da granularidade ser fina, pode ocorrer o crescimento acelerado de dados, e nem todas as ferramentas estão aptas a tratar este crescimento (DERI; MAINARDI; FUSCO, 2012).

Apesar do modelo proposto por (PFITSCHER; PILLON; OBELHEIRO, 2014) ter sido validado, um projeto de ferramenta deve buscar desacoplar ao máximo a lógica do modelo de diagnóstico, facilitando a substituição deste por novos modelos que venham a surgir, e o gerenciamento dos dados obtidos em medições, permitindo a adição de outras métricas (não consideradas originalmente) que venham a se fazer necessárias. O objetivo do presente trabalho é, então, projetar uma ferramenta para o diagnóstico do provisionamento de recursos em nuvens IaaS.

1.1 Objetivos

1.1.1 Objetivo Geral

Projetar e desenvolver uma ferramenta genérica para diagnóstico, baseado na análise de séries temporais de métricas de desempenho, do provisionamento de recursos em nuvens IaaS.

1.1.2 Objetivos Específicos

- Estudar técnicas e ferramentas de monitoração em sistemas operacionais.
- Estudar técnicas e ferramentas de armazenamento e processamento de séries temporais.
- Definir uma arquitetura modular para a ferramenta de diagnóstico, de forma a permitir a adaptação desta a diversos modelos de diagnóstico baseados na análise de séries temporais de métricas de desempenho.
- Implementar a ferramenta proposta, instanciando como modelo de diagnóstico o proposto por (PFITSCHER; PILLON; OBELHEIRO, 2014).

1.2 Estrutura do Trabalho

Considerando o objetivo geral deste trabalho o conteúdo seguinte foi dividido em quatro capítulos. O Capítulo 2 contextualiza o domínio alvo de aplicação da ferramenta e apresenta o modelo de diagnóstico de provisionamento de recursos que será instanciado.

O Capítulo 3 apresenta princípios sobre a monitoração de ambientes computacionais, técnologias para a monitoração e para armazenamento dos dados coletados. O Capítulo 4 apresenta a ferramenta desenvolvida para o diagnóstico de provisionamento de recursos em nuvens IaaS. O Capítulo 5 descreve testes realizados para verificar a efetividade da ferramenta. Por fim, no Capítulo 6, são apresentadas as considerações finais deste trabalho.

2 Diagnóstico de Provisionamento de Recursos em Nuvens IaaS

Este capítulo contextualiza o domínio alvo de aplicação da ferramenta e apresenta o modelo de diagnóstico de provisionamento de recursos que será instanciado pela ferramenta. A Seção 2.1 conceitualiza o termo computação em nuvem e apresenta suas principais características, seus modelos de serviço e suas categorias quanto a finalidade. A Seção 2.2 discorre sobre o provisionamento de recursos em nuvens IaaS. A Seção 2.3 descreve o modelo proposto por (PFITSCHER; PILLON; OBELHEIRO, 2014).

2.1 Computação em Nuvem

Nuvens computacionais podem ser definidas, em linhas gerais, como ambientes de computação com localidade transparente ao usuário. Ou, em outras palavras, estruturas com recursos computacionais que podem ser acessados e utilizados remotamente por usuários (sem que os clientes necessitem se preocupar com a localização ou organização destas estruturas). Como este termo tem sido utilizado para representar diferentes propósitos, diversas definições foram apresentadas por pesquisadores da área. Contudo, considerando as similaridades existentes entre definições, este trabalho adotou a definição concebida pelo NIST (*National Institute of Standards and Technology*), que também foi utilizada por outros trabalhos da área (PFITSCHER; PILLON; OBELHEIRO, 2014; ZHANG; CHENG; BOUTABA, 2010; HOEFER; KARAGIANNIS, 2010):

> "Computação em nuvem é um modelo para permitir o acesso ubíquo, conveniente, sob demanda e remoto a um pool compartilhado de recursos computacionais configuráreis (por exemplo, redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente provisionados e liberados com um mínimo de esforço de gerenciamento ou interação com o provedor de serviços." (MELL; GRANCE, 2011)

A partir desta definição, pode-se observar cinco características essenciais às nuvens (MELL; GRANCE, 2011):

- Auto atendimento sob demanda: Um cliente pode provisionar recursos computacionais (ex. capacidade de processamento e largura de banda de rede) automaticamente, sem a necessidade de uma interação humana com cada provedor de serviços. Isto normalmente é possível através de interfaces de software;

- Acesso via rede: As capacidades provisionadas são acessíveis via rede, segundo padrões que permitem este acesso por uma gama heterogênea de plataformas clientes, tais como *workstations*, *laptops*, *tablets* e celulares;

- Agrupamento de recursos: Os provedores mantêm um conjunto de recursos computacionais compartilhados por seus clientes. Os recursos são designados e redesignados aos clientes de acordo com suas necessidades. Ainda que em alguns casos os clientes possam determinar a localização dos recursos segundo um alto nível de abstração (país, estado ou *datacenter*), essa informação é em geral transparente a estes;

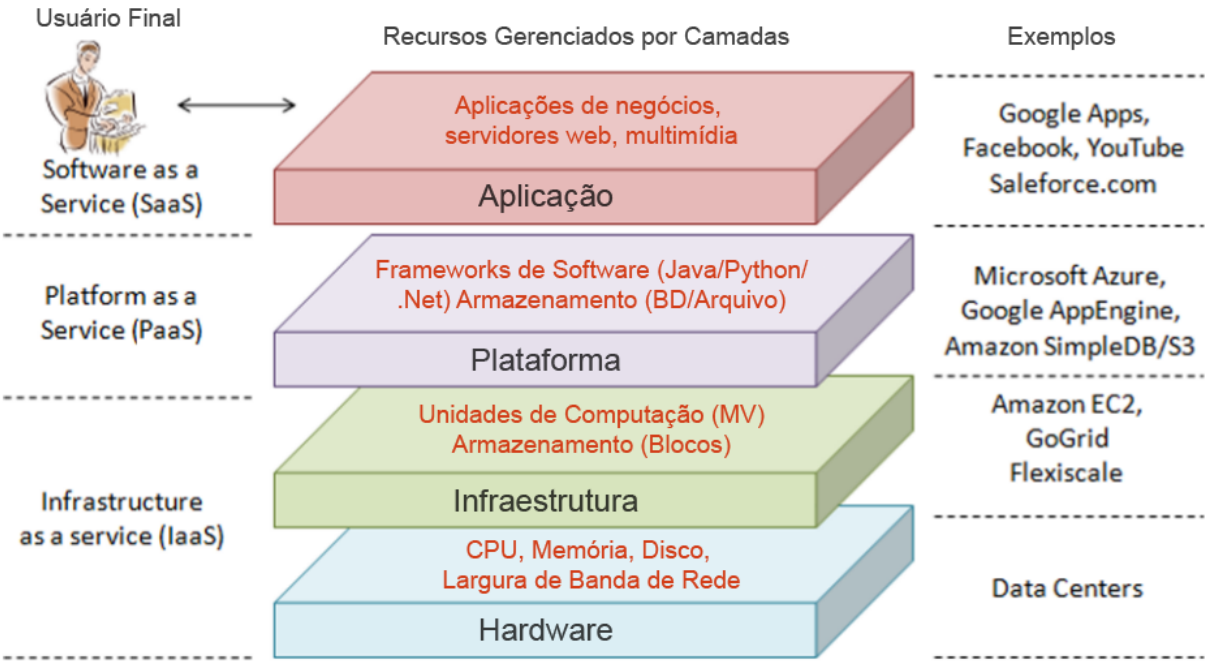
- Elasticidade: As capacidades provisionadas podem ser ajustadas (para mais ou para menos), ao longo do tempo, de forma a atender apropriadamente novas demandas de recursos do cliente. Desta forma, clientes podem ver as capacidades de provisionamento como ilimitadas, podendo atender a qualquer demanda em qualquer momento;

- Monitoração do serviço: Sistemas em nuvens controlam e otimizam automaticamente o uso de recursos aproveitando as métricas (armazenamento, processamento, largura de banda) apropriadas de acordo com o tipo de serviço disponível. A medição, controle e relato do uso de recursos provê uma transparência tanto para o cliente quanto para o provedor do serviço.

A computação em nuvem utiliza um modelo de negócio orientado a serviço. Ou seja, os recursos são oferecidos como serviços sob demanda. (ZHANG; CHENG; BOUTABA, 2010) As nuvens computacionais apresentam modelos de negócio que podem ser classificados em três modelos de serviço, segundo o tipo de recurso oferecido: *Software* (SaaS – *Software as a Service*), Plataforma (PaaS – *Plataform as a Service*) e Infraestrutura

(IaaS – *Infrastructure as a Service*) (ZHANG; CHENG; BOUTABA, 2010; MELL; GRANCE, 2011). A Figura 2.1 relaciona os modelos de serviços, seus recursos oferecidos e exemplos de provedores. Observa-se que o modelo IaaS engloba duas camadas da nuvem (infraestrutura e hardware) enquanto os modelos PaaS e SaaS englobam apenas uma camada cada (plataforma e aplicações, respectivamente).

Figura 2.1: Modelos de Serviços em Nuvens



Fonte: Traduzido de (ZHANG; CHENG; BOUTABA, 2010)

Feitas as observações sobre a Figura 2.1, os três modelos de serviço são definidos como (MELL; GRANCE, 2011):

- *Software as a Service* (SaaS): O provedor oferece como serviço a capacidade de utilizar suas aplicações, sendo que estas estão em execução sobre uma infraestrutura de nuvem. O acesso a estas aplicações é normalmente realizado via navegadores web, porém existem aplicações que utilizam outros protocolos de acesso. A infraestrutura da camada inferior é transparente ao cliente, dando a impressão de ser infinita, sendo que este não tem a capacidade de gerenciá-la. Este gerenciamento é de responsabilidade do provedor da nuvem;

- *Plataform as a Service* (PaaS): O provedor oferece como serviço a capacidade do cliente implantar suas aplicações na nuvem utilizando as linguagens de programação, bibliotecas, serviços e ferramentas suportadas pelo provedor. Semelhante ao SaaS,

este modelo não possibilita o gerenciamento da infraestrutura da camada inferior (como rede, servidores, sistemas operacionais). O cliente tem, entretanto, o controle sobre as aplicações implantadas e possivelmente sobre as configurações do ambiente da aplicação;

- *Infrastructure as a Service* (IaaS): O provedor oferece como serviço a possibilidade do cliente provisionar recursos na forma de capacidade de processamento (CPUs), armazenamento (memória de acesso aleatório, espaço em disco), e rede (largura de banda). Além disto, o cliente pode ter acesso direto aos recursos provisionados para implantar suas aplicações ou serviços. Na maioria das vezes, os recursos são fornecidos com base em instâncias de máquinas virtuais (MVs) configuráveis e gerenciáveis pelo cliente. Esse tem, porém, seu acesso limitado as suas instâncias virtuais, sem permissão de acesso à camada de hardware e ao ambiente de virtualização.

A ferramenta desenvolvida neste trabalho é de utilidade a clientes do modelo de serviço IaaS (que podem basear-se no diagnóstico gerado para provisionar adequadamente suas aplicações e serviços), pois apenas este modelo possibilita o ajuste dos recursos (CPU, memória, e rede) que são diagnosticáveis através da análise de métricas de desempenho.

Além da classificação quanto ao modelo de serviço, as nuvens computacionais podem ser categorizadas segundo a finalidade destas, expressa por modelos de implantação. Os modelos de implementação são quatro (MELL; GRANCE, 2011; ZHANG; CHENG; BOUTABA, 2010):

- Nuvens Privadas: Também conhecidas como nuvens internas, estas são projetadas para o uso exclusivo por uma única organização. Uma nuvem privada pode ser construída e gerenciada pela própria organização ou por organizações terceiras. Essa característica permite o mais alto controle sobre o desempenho, confiabilidade e segurança da infraestrutura, uma vez que os clientes, por serem internos à organização, são conhecidos. Apesar disto, estas são criticadas por serem similares às *server farms* proprietárias tradicionais, necessitando de um investimento inicial de capital em infraestrutura;

- Nuvens Comunitárias: Muito semelhantes às nuvens privadas, as nuvens comunitárias são projetadas para o uso exclusivo por uma comunidade de clientes, normalmente organizações com objetivos compartilhados (missão, requisitos de segu-

rança, políticas e considerações de conformidade). Estas nuvens podem pertencer, ser gerenciadas e operadas por uma ou mais organizações da comunidade, por uma organização terceira, ou alguma combinação destas;

- Nuvens Públicas: Nuvens nas quais o provedor oferece seus recursos como serviços para o público geral. Estas nuvens são gerenciadas pelo provedor da infraestrutura, trazendo diversos benefícios para os clientes, incluindo não necessitar de investimento inicial de capital em infraestrutura e delegar os riscos infraestruturais ao provedor;

- Nuvens Híbridas: A infraestrutura da nuvem é a composição de duas ou mais infraestruturas distintas (privada, comunitária, ou pública). Uma organização pode elaborar uma nuvem híbrida através da composição de sua nuvem privada com a utilização dos recursos de uma nuvem pública.

A ferramenta para diagnóstico apresentada neste trabalho pode ser utilizada em todos os modelos de implantação, desde que o cliente tenha acesso em nível de sistema operacional às instâncias provisionadas, de forma a permitir a medição de métricas de desempenho. O exemplo mais direto da utilização da ferramenta é em nuvens públicas, onde há maior interesse em equilibrar custo e benefício por parte dos clientes. Usuários de outros modelos de implantação podem, entretanto, utilizar a ferramenta para otimizar os provisionamentos através da racionalização adequada de recursos.

2.2 Provisionamento de Recursos em Nuvens IaaS

Como apresentado anteriormente, em nuvens IaaS os recursos são oferecidos na forma de capacidade de processamento, armazenamento, redes entre outros; esses geralmente são associados a instâncias de máquinas virtuais. Essa associação é feita pela tecnologia de virtualização, em que uma camada de software, denominada monitor de máquinas virtuais (MMV) ou hipervisor, é inserida entre o hardware e o sistema operacional, de forma que este último tem acesso apenas a abstrações (recursos virtuais) dos recursos físicos subjacentes, que são gerenciados pelo MMV. Isso permite que diversas máquinas virtuais sejam executadas sobre um mesmo hardware, com o hipervisor garantindo o isolamento entre as MVs e a transparência no acesso aos recursos físicos (TANENBAUM, 2008).

O provisionamento (reserva e distribuição) dos recursos a máquinas virtuais pode variar em diferentes níveis de granularidade, segundo o modo que o provedor trata os recursos. Existem três definições de tratamento de recursos em nuvens IaaS (LAGAR-CAVILLA et al., 2009):

- Na primeira recursos são reservados de forma não particionada (unidades de CPU, interfaces de rede);
- Na segunda estes são reservados de forma particionada (percentual de CPU, percentual de largura de banda);
- Na terceira estes são reservados em pacotes (instâncias de máquinas virtuais com características fixas de recursos).

Em seu trabalho, (PFITSCHER; PILLON; OBELHEIRO, 2014) assume que a granularidade em que os recursos podem ser oferecidos está relacionada com a capacidade tecnológica do ambiente de virtualização em entregá-los. No hipervisor Xen, cada tipo de recurso pode ser provisionado de formas diferentes: alguns (processador e rede) são compartilhados pelas máquinas virtuais, ao passo que outros (memória e disco) são distribuídos em porções fixas. Mesmo para os recursos compartilhados é possível definir limites de utilização. Os itens abaixo descrevem a possibilidade de reserva de recursos com base no hipervisor Xen segundo (PFITSCHER; PILLON; OBELHEIRO, 2014) (outras características semelhantes podem ser encontradas em outros hipervisores):

- Processador: compartilhado; no MMV Xen, cada máquina virtual possui um ou mais processadores virtuais (VCPU). O escalonamento das VCPUs a processadores físicos (PCPUs) pode ser realizado de três maneiras (MATTHEWS, 2008; TAKEMURA; CRAWFORD, 2009):
  - A VCPU é fixada a uma PCPU dedicada; isto é possível quando o número de VCPUs é inferior ao número de PCPUs, e não há, portanto, competição entre VCPUs por uma PCPU;
  - Uma PCPU é multiplexada no tempo; de acordo com a demanda, a VCPU escalonada na PCPU é alternada ao longo do tempo. No caso de duas VCPUs concorrerem por uma PCPU e uma não possuir demanda de processamento, a

outra irá executar segundo sua demanda, tendo como único limite a capacidade da PCPU;

  - É realizado um rodízio entre as VCPUs, permitindo que cada uma utilize uma PCPU por certa fração de tempo, desta forma mesmo que a VCPU escalonada não demande processamento, outras VCPUs não podem utilizar a PCPU enquanto não forem escalonadas.

- Rede: compartilhado através da multiplexação do recurso físico ao longo do tempo; o provisionamento é particionado, definindo-se a largura de banda disponível para cada máquina virtual em Mbps. O hipervisor Xen não permite a alteração deste valor em tempo de execução, ainda assim há alternativas para limitá-lo a valores inferiores ao definido;

- Memória: distribuído em porções fixas; isto se deve principalmente à necessidade de manter o isolamento entre máquinas virtuais. No hipervisor Xen, a quantidade de memória disponível é definida na criação da máquina virtual, e alguns mecanismos podem ser utilizados para alterar esse valor em tempo de execução (como o *balloon driver* (WALDSPURGER, 2002));

- Disco: tradicionalmente distribuído em porções fixas; o método comum é associar um arquivo de disco virtual de tamanho fixo ou variável às máquinas virtuais.

A elasticidade do provisionamento de recursos, uma das características essenciais que definem uma nuvem (como visto na Seção 2.1), é um dos fatores que influenciaram no crescimento da adoção das nuvens (PFITSCHER; PILLON; OBELHEIRO, 2014), pois possibilita a utilização de modelos do tipo *pay-as-you-go*. Com isto, por exemplo, uma *startup* na Internet pode atender mais rapidamente a um pico na demanda quando seu serviço se tornar popular, sem a necessidade de estimar previamente essa demanda, como ocorre no uso de servidores privados (ARMBRUST et al., 2010). Além de vantagens para o cliente da nuvem, a elasticidade traz também benefícios para o provedor, como o aumento na porcentagem média de uso de seus recursos. Porém, uma desvantagem causada pela elasticidade é o incremento da complexidade do gerenciamento do sistema pelo provedor (VAN; TRAN; MENAUD, 2009). O provisionamento em nuvens IaaS pode ser realizado de duas maneiras (PFITSCHER; PILLON; OBELHEIRO, 2014):

- Provisionamento Estático: O cliente determina previamente quais são as necessidades de recursos, reservando-os a uma ou múltiplas instâncias de máquinas virtuais. A reserva de recursos não é permitida em tempo de execução da máquina virtual. Caso o cliente identifique que necessita alterar suas definições, deve desligar, reconfigurar e reiniciar a máquina virtual;

- Provisionamento Dinâmico: Em oposição ao estático, o provisionamento dinâmico permite que recursos sejam reservados em tempo de execução da máquina virtual, de forma a atender o aumento ou diminuição na demanda de recursos. Este ajuste no provisionamento pode ser realizado pelo provedor ou pelo cliente, e frequentemente estes utilizam técnicas de aprendizado de máquina e modelos preditivos de desempenho para realizá-lo.

A partir de um diagnóstico confiável do provisionamento de recursos é possível atingir um dos principais objetivos dos provedores e clientes de nuvens IaaS, o de reservar e distribuir adequadamente os recursos segundo as necessidades das aplicações.

2.3 Modelo de Diagnóstico

Para diagnosticar o provisionamento de recursos, é necessário definir um conjunto de métricas (modelo) que possam indicar se esse provisionamento está adequado, subprovisionado ou superprovisionado. Um diagnóstico de superprovisionamento significa que a capacidade provisionada é superior a necessária às aplicações (é possível reduzir a capacidade provisionada sem afetar significativamente o desempenho dessas aplicações). Além disso, o aumento da capacidade não representa uma melhora relevante no desempenho. O subprovisionamento significa que a capacidade provisionada é inferior à necessária para um desempenho adequado (o aumento da capacidade dos recursos representa uma melhora relevante no desempenho). O diagnóstico adequado representa um estado de provisionamento onde a redução de capacidade afeta significativamente o desempenho e o aumento da capacidade dos recursos não representa uma melhora relevante no desempenho (PFITSCHER; PILLON; OBELHEIRO, 2014).

Como comentado no Capítulo 1, o presente trabalho instancia o modelo de (PFITSCHER; PILLON; OBELHEIRO, 2014) para diagnosticar o provisionamento de recursos. Nas subseções seguintes são apresentados os modelos propostos para diagnóstico de

três recursos: processador, memória e rede. É importante ressaltar que as apresentações buscam apenas documentar o modelo, sem justificativas para as escolhas de seus autores. O texto que segue foi baseado em (PFITSCHER; PILLON; OBELHEIRO, 2014) (salvo indicação em contrário) e referências a este trabalho serão omitidas para evitar repetição.

2.3.1 Modelo para Processador

Para diagnosticar o provisionamento de processador, duas métricas são avaliadas: a utilização de processador (U_{cpu}) e o percentual de "roubo" ($\%Steal$). A utilização U_{cpu} representa quanto tempo o processador estava executando tarefas em um determinado período. O $\%Steal$ é uma métrica que representa o percentual de tempo em que uma máquina virtual deseja executar (possui processos em estado de pronto), mas seu processador virtual (VCPU) não está alocado em uma CPU física, devido ao rodízio entre máquinas virtuais.

Como U_{cpu} pode alternar entre picos de utilização durante um período, o modelo utiliza valores como média ($\overline{U_{cpu}}$) e índice de saturação de processador (φ). A média $\overline{U_{cpu}}$ é definida como a média aritmética entre todas as medições da utilização de processador durante um determinado período.

Sendo U^i_{cpu} um ponto de medição de U_{cpu}, θ o limiar de saturação de CPU, N o número de pontos no período e N' o número de pontos com CPU saturada ($U^i_{cpu} \geq \theta$), o índice φ é definido como N'/N. Em outras palavras, φ é o percentual de pontos de medição com CPU saturada, em relação ao total de pontos medidos em um período. Um limiar ω sobre φ pode ser definido, de forma que a relação entre estes (igual, maior, menor) seja usada para o diagnóstico do provisionamento de processador. As equações (2.1), (2.2) e (2.3) demonstram matematicamente o cálculo do percentual de pontos em saturação.

$$u_i = \begin{cases} 1, & \text{se } U^i_{cpu} \geq \theta \\ 0, & \text{se } U^i_{cpu} < \theta \end{cases} \tag{2.1}$$

$$N' = \sum_{i=0}^{N} u_i \tag{2.2}$$

$$\varphi = \frac{N'}{N} \tag{2.3}$$

A Tabela 2.1 expressa o diagnóstico de provisionamento de processador conforme as métricas propostas. Os valores adotados para os limiares θ e ω são 95% e 20%, respectivamente.

Tabela 2.1: Diagnóstico de provisionamento de CPU

	$\%Steal \geq 1\%$	$\%Steal < 1\%$
$\overline{U_{cpu}} \geq 80\%$	Subprovisionado	Subprovisionado
$50\% \leq \overline{U_{cpu}} < 80\%$	Subprovisionado	Adequado
$\overline{U_{cpu}} < 50\% \wedge \varphi > 20\%$	Subprovisionado	Adequado
$\overline{U_{cpu}} < 50\% \wedge \varphi \leq 20\%$	Subprovisionado	Superprovisionado

Fonte: (PFITSCHER; PILLON; OBELHEIRO, 2014)

A Tabela 2.1 define a atribuição do diagnóstico de subprovisionamento para dois casos: o percentual $\%Steal$ ser maior ou igual a 1%, ou a média $\overline{U_{cpu}}$ ser maior ou igual a 80%. O diagnóstico de superprovisionamento é dado quando o percentual $\%Steal$ for menor que 1%, a média $\overline{U_{cpu}}$ for menor que 50% e o índice φ for menor ou igual a 20%. O diagnóstico adequado indica que, além de o percentual $\%Steal$ ser menor que 1%, a média $\overline{U_{cpu}}$ está entre 50% e 80%, ou, é inferior a 50% e o índice φ é maior que 20%.

2.3.2 Modelo para Memória

Para diagnosticar o provisionamento de memória, duas métricas são avaliadas: memória residente e memória reservada (*committed*). Devido ao gerenciamento de memória ser bastante dependente de plataforma, a discussão a seguir descreverá essas métricas em função de métricas disponíveis no Linux, mas outros sistemas (como Windows) dispõem de métricas semelhantes (CHERUBINI, 2013). Pelo fato da taxa de oscilação da utilização deste recurso ser menor do que em outros recursos (processador, rede), a abordagem de avaliação tomada é preventiva. Além disso, o esgotamento deste recurso afeta o desempenho sensivelmente, pois o sistema operacional (SO) passa a utilizar a área de *swap*. A memória residente expressa a ocupação da memória, e indica a memória utilizada pelos processos e pelo SO, sem considerar o espaço em memória sendo utilizado para *buffers* de entrada e saída e para *cache* de páginas, pois este é adequado ao que resta da memória que

não está sendo utilizada pelos processos e pelo SO. Matematicamente a memória residente (*MemRes*) pode ser calculada como a subtração da memória total pela memória livre, a para *buffers* e a para *cache* ($MemRes = MemTotal - MemFree - Buffers - Cache$). Para o propósito de diagnóstico, *MemRes* foi categorizada nos níveis alta, confortável e baixa. Estas categorias podem ser definidas informalmente como segue:

- Memória residente alta: quando *MemRes* está muito próxima da memória total. Nesta situação, o SO começa a mover páginas para o espaço de *swap* para disponibilizar memória física, afetando o desempenho.
- Memória residente baixa: quando *MemRes* está muito abaixo da memória total. Nesta situação, a memória disponível pode ser reduzida sem impactar o desempenho da MV.
- Memória residente confortável: quando *MemRes* não está nem baixa nem alta. A MV tem memória suficiente para garantir o nível de desempenho esperado e boa parte da memória está em uso.

A métrica de memória reservada indica a quantidade de memória alocada pelos processos e pelo SO, a qual com frequência não é completamente utilizada. Em função disso, muitos SOs fazem *overcommit* de memória, permitindo que os processos aloquem mais memória do que o disponível no conjunto memória física e área de *swap*, postergando falhas para quando (e se) ocorrer uso de memória acima da capacidade. No Linux, a quantidade de memória reservada é fornecida pela métrica *Committed_As*; essa métrica indica a quantidade de memória necessária (no pior caso) para acomodar todos os processos, e pode ser usada para sugerir a possibilidade da memória residente crescer a ponto de afetar o desempenho. Desta forma, a memória reservada pode ser classificada como relevante (quando *Committed_AS* é alta a ponto de sugerir risco considerável) ou irrelevante (quando o risco é desprezível).

Formalmente, para diagnosticar o provisionamento, são utilizadas duas métricas, média de memória residente ($\overline{m_r}$) e média de memória reservada ($\overline{m_c}$), e três limiares, limiar de memória residente alta (α), limiar de memória residente baixa (β), e limiar de memória reservada relevante (γ). Considerando a obtenção periódica dos valores, $\overline{m_r}$ e $\overline{m_c}$ são a média dos valores obtidos no intervalo de diagnóstico. As Equações 2.4 e 2.5 mostram o cálculo de um ponto de medição para m_r e m_c, respectivamente. Assim, a

memória residente é alta quando $\overline{m_r} \geq \alpha$, confortável quando $\beta \leq \overline{m_r} < \alpha$, e baixa quando $\overline{m_r} < \beta$; e ainda, a memória reservada é relevante quando $\overline{m_c} \geq \gamma$ e irrelevante quando $\overline{m_c} < \gamma$. Os valores utilizados para os limiares são $\alpha = 70\%$, $\beta = 50\%$ e $\gamma = 150\%$.

$$m_c = \frac{MemRes}{MemTotal} \tag{2.4}$$

$$m_r = \frac{Committed\_AS}{MemTotal} \tag{2.5}$$

A Tabela 2.2 define o dianóstico de provisionamento de memória. Este é subprovisionado quando a memória residente é alta ou quando esta é confortável e a memória reservada é relevante. O superprovisionamento ocorre quando a memória residente é baixa e a reservada é irrelevante. O provisionamento é considerado adequado quando, a memória residente é confortável e a reservada é irrelevante ou a primeira é baixa e a segunda é relevante.

Tabela 2.2: Diagnóstico de provisionamento de memória

	$\overline{m_c} \geq \gamma$ (relevante)	$\overline{m_c} < \gamma$ (irrelevante)
$\overline{m_r} \geq \alpha$ (alta)	Subprovisionado	Subprovisionado
$\beta \leq \overline{m_r} < \alpha$ (confortável)	Subprovisionado	Adequado
$\overline{m_r} < \beta$ (baixa)	Adequado	Superprovisionado

Fonte: (PFITSCHER; PILLON; OBELHEIRO, 2014)

2.3.3 Modelo para Rede

Para diagnosticar o provisionamento de rede, duas métricas são avaliadas: o consumo de banda e o tamanho da fila na interface de rede da MV. Semelhantemente ao uso de CPU, a largura de banda consumida está sujeita a grandes variações ao longo do tempo. Verificou-se experimentalmente que, quando uma MV dispõe de largura de transmissão de banda adequada, seu consumo de banda é aproximadamente constante. Em função disso, o coeficiente de variação CV_{lb} (razão entre o desvio padrão e a média) dos valores obtidos nas medições do consumo de banda é avaliado. Sobre este coeficiente CV_{lb} é definido um limiar para indicar variações significantes, o valor adotado para este limiar é 5%.

O consumo constante de largura de banda também pode caracterizar subprovisionamento de rede. Nesse caso, uma máquina virtual utiliza toda a capacidade dis-

ponível e permanece com necessidades de transmissão não satisfeitas. Para entender de que forma essa situação pode ser caracterizada, é necessário compreender como funciona o provisionamento de rede em ambientes virtuais. Cada MV possui uma ou mais interfaces virtuais de rede, que são multiplexadas sobre as interfaces físicas de rede (do hospedeiro) [Matthews 2008]. Cada interface virtual possui a sua própria fila de transmissão, e essas filas são conectadas à fila de transmissão da interface física. As interfaces virtuais podem competir pela banda disponível na interface física (multiplexação estatística), ou podem ter capacidade provisionada pelo hipervisor. Quando uma MV tenta transmitir quadros com uma vazão superior à capacidade disponível, ocorre formação de fila em sua interface virtual, independentemente do fator limitante ser a capacidade reservada para a interface virtual ou a capacidade da interface física subjacente. Dessa forma, o enfileiramento de quadros na interface virtual de rede pode ser usado como uma segunda métrica no diagnóstico do provisionamento. Para a avaliação utiliza-se a média $\overline{fila}$ de todas as medições em um período, e define-se sobre essa um limiar que indica qual o número máximo de quadros na fila tolerado pelo modelo, utiliza-se o valor zero para indicar que não é tolerado que existam quadros na fila da interface de rede.

O diagnóstico da rede segue o seguinte modelo, caso exista fila na interface de rede da MV ($\overline{fila} > 0$), o diagnóstico é de subprovisionamento. Se não existir fila e o consumo de banda for constante ($CV_{lb} \leq 5\%$) o provisionamento está adequado. No outro caso, não existe fila e o consumo é variável ($CV_{lb} > 5\%$), está ocorrendo o superprovisionamento. A Tabela 2.3 expressa o modelo de diagnóstico de provisionamento de rede descrito.

Tabela 2.3: Diagnóstico de provisionamento de rede

	$\overline{fila} > 0$	$\overline{fila} = 0$
$CV_{lb} > 5\%$	Subprovisionado	Superprovisionado
$CV_{lb} \leq 5\%$	Subprovisionado	Adequado

Fonte: (PFITSCHER; PILLON; OBELHEIRO, 2014)

2.4 Considerações do Capítulo

Este capítulo contextualizou o domínio alvo da ferramenta objetivo deste trabalho. A computação em nuvem foi conceitualizada como um modelo para permitir o acesso ubíquo,

conveniente, sob demanda e remoto a um pool compartilhado de recursos computacionais configuráreis que podem ser rapidamente provisionados e liberados com um mínimo de esforço de gerenciamento ou interação com o provedor de serviços (MELL; GRANCE, 2011) e discorreu-se sobre a capacidade dos clientes de provisionar recursos computacionais segundo suas necessidades.

Por fim, descreveu-se o modelo de diagnóstico de provisionamento de recursos proposto em (PFITSCHER; PILLON; OBELHEIRO, 2014) que será instanciado pela ferramenta fruto deste trabalho. Resumidamente, o modelo analisa o percentual de uso e de "roubo" do processador; as memórias residente e reservada; e o consumo de banda e o tamanho da fila na interface de rede para diagnosticar o provisionamento do processador, da memória e da rede, respectivamente.

No capítulo seguinte conceitua-se o termo monitoração no âmbito do presente trabalho e são estudadas algumas ferramentas que auxiliam na monitoração de métricas referentes ao modelo de diagnóstico apresentado no presente capítulo e no armazenamento dos dados coletados.

3 Monitoração

Este capítulo apresenta princípios sobre a monitoração de ambientes computacionais, ferramentas para a monitoração e para armazenamento dos dados coletados. A Seção 3.1 conceitua o termo monitoração no contexto do trabalho e descreve os tipos de monitores quanto à abordagem de implementação e as classes de monitores quanto ao tipo de métricas observadas, o local da monitoração e o propósito sobre os dados obtidos. A Seção 3.2 apresenta algumas ferramentas que permitem a obtenção de métricas de desempenho. A Seção 3.3 apresenta algumas ferramentas para o armazenamento das métricas.

3.1 Princípios

A monitoração é uma técnica utilizada para observar as atividades de um ambiente computacional, seja este uma rede, um servidor ou uma máquina virtual (MV). Isso é realizado através de monitores, ferramentas que têm como principal função coletar métricas relacionadas ao consumo de recursos pelo sistema monitorado. Dentro do contexto de provisionamento de recursos, estas métricas podem ser avaliadas para definir o nível de adequação dos recursos reservados (PFITSCHER; PILLON; OBELHEIRO, 2014; MENASCE; ALMEIDA, 2001).

Idealmente, um monitor deve ser apenas um observador do sistema monitorado, e de forma alguma um participante deste. Normalmente o monitor necessita utilizar alguns dos recursos que monitora, entretanto, o *overhead* imposto pelo monitor sobre o sistema monitorado deve ter impacto insignificante sobre seu desempenho. Existem três diferentes tipos de monitores, dependendo da sua abordagem de implementação: *hardware*, *software* e híbrido (PFITSCHER; PILLON; OBELHEIRO, 2014; MENASCE; ALMEIDA, 2001).

- Monitor de *hardware*: Um monitor de *hardware* examina o estado do sistema computacional estudado através de sensores eletrônicos ligados ao *hardware* deste. Estes sensores permitem obter informações sobre os registradores, posições de memória e canais de entrada e saída. Os monitores de *hardware* possuem algumas vantagens

associadas. Como estes são externos ao sistema monitorado, eles não consomem recursos do sistema. Isto quer dizer que monitores deste tipo tipicamente não impõem nenhum *overhead* sobre o sistema monitorado. Além disso, estes normalmente podem lidar com taxas de dados maiores que os monitores de *software*, ou seja, apresentam melhor desempenho. Existem, entretanto, alguns desafios associados ao uso de monitores de *hardware*. Estes, normalmente, não têm acesso a informações relacionadas a *software*, dificultando o uso deste tipo de monitor para obter medidas de desempenho relacionadas a aplicações ou cargas de trabalho.

- Monitor de *software*: Um monitor de *software* consiste em uma aplicação que executa sobre o sistema operacional (SO) e coleta informações sobre o consumo de recursos dos programas ou do sistema. Esta coleta é realizada por um conjunto de rotinas de captura que são ativadas na ocorrência de eventos específicos ou em intervalos predefinidos. Algumas vantagens associadas aos monitores de *software* são: a flexibilidade de configuração, a quantidade de informação que deve ser registrada destes recursos e a facilidade de instalação e utilização. Muitas vezes, este tipo de monitor utiliza os mesmos recursos (como processador e memória) que estão sendo monitorados, introduzindo uma perturbação que, além de ser difícil de mensurar, pode influir significativamente nas métricas observadas.

- Monitor híbrido: Monitores híbridos são monitores que combinam características de monitores de *hardware* e *software*. Algumas ferramentas de avaliação de desempenho de rede utilizam-se de monitores híbridos, facilitando a captura simultânea de dados de *hardware* e *software*.

No contexto de nuvens IaaS é impraticável aos clientes a utilização de monitores de *hardware* (e consequentemente de híbridos) visto que estes clientes não têm acesso ao *hardware* físico. Portanto, os monitores mais apropriados para nuvens IaaS são os de *software*, pois este tipo de monitor pode coletar dados sem a necessidade de permitir acesso ao *hardware* (MENASCE; ALMEIDA, 2001). Os dados coletados por monitores de *software*, no contexto de monitoração em nuvens IaaS, podem ser divididos em duas classes, de acordo com o tipo das métricas observadas, o local da monitoração e o propósito sobre os dados obtidos: *guest-wide* e *system-wide* (DU; SEHRAWAT; ZWAENEPOEL, 2011; ACETO et al., 2013; PFITSCHER; PILLON; OBELHEIRO, 2014).

- *Guest-wide:* Este tipo de monitoração, realizada em nível de máquina convidada, utiliza métricas interessantes principalmente aos clientes, e são obtidos dados relacionados a uma MV. O monitor é inserido a nível de SO da MV, e desta forma o cliente observa o consumo de recursos no intervalo de tempo monitorado.

- *System-wide:* Este tipo de monitoração, realizada em nível de sistema hospedeiro, é interessante principalmente ao provedor da infraestrutura de virtualização, pois pode apresentar métricas e dados relacionados a múltiplas máquinas virtuais e ao próprio monitor de máquinas virtuais. Possibilitando identificar e resolver problemas relacionados.

O presente trabalho propõe uma ferramenta que pode ser classificada como monitor de *software*, pois será executada sobre o SO de uma MV; e coleta dados do tipo *Guest-wide*, informações relacionadas a uma MV e seus recursos provisionados.

3.2 Monitores de *Software*

Nas subseções seguintes são apresentados alguns monitores de *software Guest-wide* que permitem a obtenção de métricas de desempenho utilizadas por (PFITSCHER; PILLON; OBELHEIRO, 2014). Estes foram estudados para auxiliar na escolha de quais serão utilizados na instanciação do modelo de diagnóstico. Os monitores apresentado são para SOs baseados em Linux, porém semelhantes podem ser utilizados em outros SOs. Cada subseção apresenta um monitor, relacionando os recursos e as métricas que este permite monitorar e apresentando um breve exemplo de sua utilização.

3.2.1 vmstat

O vmstat provê informações sobre processos, memória, paginação, e atividade de CPU. No contexto deste trabalho este monitor permite a obtenção de métricas (percentual de uso e de roubo da CPU, e memória livre) para o diagnóstico do provisionamento de processador e de memória. Este monitor impõe um baixo *overhead* sobre o sistema monitorado, o que permite que seja executado mesmo durante um período onde a carga sobre o sistema estiver alta (CILIENDO; KUNIMASA, 2007; HOCH, 2010). O exemplo abaixo demostra o vmstat executando sob intervalos de 1 segundo durante 5 segundos.

```
# vmstat 1 5
procs ---------memory---------- -swap- --io- -system-- ------cpu-----
 r  b swpd   free   buff  cache si  so bi bo   in   cs us sy id wa st
 1  0    0 136844 101420 745844  0   0  0  0  417  810 51  1 48  0  0
 1  0    0 136844 101420 745844  0   0  0  0  430  826 53  1 47  0  0
 1  0    0 136844 101420 745844  0   0  0  0  421  789 51  1 48  0  0
 1  0    0 139540 101420 745844  0   0  0  0  426  799 54  1 46  0  0
 1  0    0 139324 101420 745844  0   0  0  0  406  786 51  1 48  0  0
```

A Tabela 3.1 descreve as métricas apresentadas pelo vmstat relevantes ao presente trabalho. Observa-se que o vmstat não apresenta as métricas de CPU individualmente para cada processador, no caso de máquinas com múltiplos processadores as estatísticas de utilização de CPU mostram a média dos processadores.

Tabela 3.1: Métricas do vmstat

Campo		**Descrição**
cpu	*us*	Percentual de tempo gasto executando código a nível de usuário (tempo de usuário)
	sy	Percentual de tempo gasto executando código do *kernel* (tempo de sistema)
	id	Percentual de tempo ocioso
	wa	Percentual de tempo gasto esperando por estrada/saída
	st	Percentual de tempo roubado de uma máquina virtual
memory	*swpd*	Quantidade de memória virtual utilizada
	free	Quantidade de memória livre
	buff	Quantidade de memória utilizada como *buffer*
	cache	Quantidade de memória utilizada como *cache*

Fonte: autoria própria

3.2.2 mpstat

Como visto na subseção anterior o vmstat pode ser utilizado para monitorar a atividade da CPU. No caso de um sistema com múltiplos núcleos de processamento, entretanto, pode ser interessante monitorar cada um dos núcleos individualmente. Este monitoração pode ser realizada através do monitor mpstat (CILIENDO; KUNIMASA, 2007; HOCH, 2010). O

exemplo abaixo demostra o mpstat com o parâmetro "-P ALL", que apresenta as métricas individuais de todos os núcleos, além da média.

```
# mpstat -P ALL
Linux 3.13.0-24-generic (linuxpc) 14-05-2014 _x86_64_ (2 CPU)

23:24:31 CPU   %usr %nice %sys %iowait %irq %soft %steal %guest %gnice
   %idle
23:24:32 all  51,50  0,00 1,00    0,00 0,00  0,00   0,00   0,00   0,00
   47,50
23:24:32   0 100,00  0,00 0,00    0,00 0,00  0,00   0,00   0,00   0,00
   0,00
23:24:32   1   3,00  0,00 1,00    0,00 0,00  0,00   0,00   0,00   0,00
   96,00
```

A Tabela 3.2 descreve as métricas apresentadas pelo mpstat. Observa-se na saída anterior que os processadores possuem uma média de $51,50\%$ de utilização pelo usuário (*%usr*) (conforme apresentado também pelo vmstat na subseção anterior), mas, além disso, o mpstat permite identificar que um dos processadores (CPU 0) está completamente carregado enquanto o outro (CPU 1) está quase que totalmente ocioso.

3.2.3 free

O monitor free provê informações sobre a memória de um sistema. As métricas apresentadas indicam o total de memória livre e em uso (incluindo a área de *swap*), além da quantidade de memória compartilhada, e utilizada como *buffer* ou *cache* (CILIENDO; KUNIMASA, 2007). O exemplo abaixo demostra a saída de uma execução do free.

```
# free
             total       used       free     shared    buffers     cached
Mem:       2042728    1949524      93204     189692       5916     292216
-/+ buffers/cache:    1651392     391336
Swap:      2094076     123984    1970092
```

Na saída acima, os valores mostrados na linha iniciada por "Mem:" indicam, respectivamente: a memória total do sistema, a quantidade de memória em uso, a quantidade de memória livre, a quantidade de memória compartilhada, a quantidade de memória usada como *buffer* e a quantidade de memória usada como *cache*. A linha iniciada por "-/+ buffers / cache:" apresenta dois valores que indicam, respectivamente: a quantidade

Tabela 3.2: Métricas do mpstat

Campo	**Descrição**
CPU	Número do processador. A palavra-chave *all* indica métricas calculadas como a média dentre todos os processadores
%usr	Percentual de tempo gasto executando código a nível de usuário (tempo de usuário)
%nice	Percentual de tempo gasto executando código a nível de usuário com prioridade *nice*
%sys	Percentual de tempo gasto executando código do *kernel* (tempo de sistema)
%iowait	Percentual de tempo gasto esperando por entrada/saída
%irq	Percentual de tempo gasto atendendo a interrupções de *hardware*
%soft	Percentual de tempo gasto atendendo a interrupções de *software*
%steal	Percentual de tempo roubado de uma máquina virtual
%guest	Percentual de tempo gasto executando uma CPU virtual
%gnice	Percentual de tempo gasto executando uma CPU virtual com prioridade *nice*
%idle	Percentual de tempo ocioso

Fonte: autoria própria

de memória em uso e a quantidade de memória livre, desconsiderando a quantidade usada como *buffer* ou *cache*. Os valores mostrados na última linha (iniciada por "Swap:") indicam, respectivamente: a memória total da área de *swap*, a quantidade em uso da área de *swap*, e a quantidade livre da área de *swap*. Os valores do exemplo acima estão em *kilobytes*, mas é possível especificar outras unidades de medida, como: *bytes*, *megabytes*, *gigabytes* e *terabytes* (THE LINUX MAN-PAGES PROJECT, 2014a).

3.2.4 /proc/meminfo

O arquivo /proc/meminfo contém informações sobre a memória do sistema (THE LINUX MAN-PAGES PROJECT, 2014b). O exemplo abaixo (com algumas linhas omitidas) exem-

plifica o conteúdo do /proc/meminfo em um instante.

```
MemTotal:          973128 kB
MemFree:           266612 kB
Buffers:            68972 kB
Cached:            450752 kB
           [...]
Committed_AS:      964444 kB
           [...]
```

O conteúdo do arquivo respeita uma estrutura onde cada linha consiste no nome da métrica, seguido por dois pontos, o valor da métrica, e uma unidade de medida (ex. kB) (THE LINUX MAN-PAGES PROJECT, 2014b). A Tabela 3.3 descreve cada uma das métricas apresentadas no exemplo. As métricas apresentadas são as que possuem relação com o modelo de (PFITSCHER; PILLON; OBELHEIRO, 2014). As métricas "MemTotal", "Buffers" e "Cached" permitem o cálculo da memória residente; enquanto "Committed_AS" representa a própria métrica de memória reservada do modelo.

Tabela 3.3: Métricas do /proc/meminfo

Campo	**Descrição**
MemTotal	A memórita total usável do sistema
MemFree	A quantidade de memória livre
Buffers	A quantidade de memória usada como *buffer*
Cached	A quantidade de memória usada como *cache*
Committed_AS	A quantidade de memória reservada

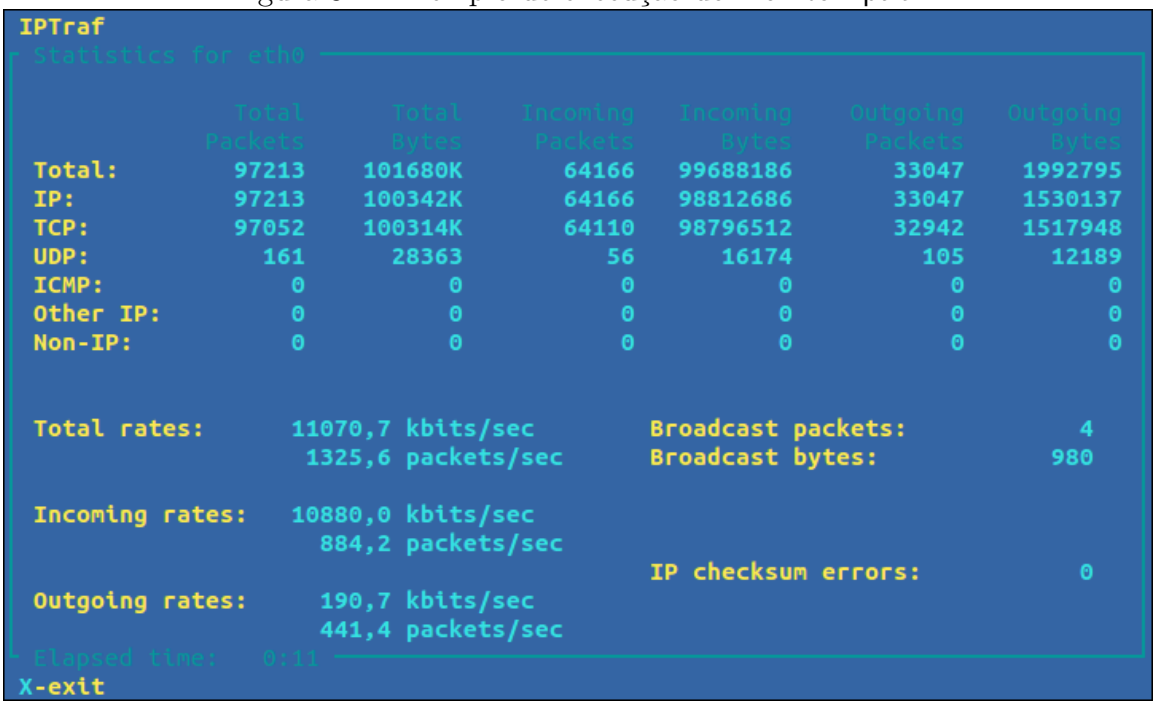
Fonte: autoria própria

3.2.5 iptraf

O iptraf monitora o tráfego TCP/IP em tempo real e apresenta métricas úteis à análise deste tráfego. As métricas do tráfego TCP/IP apresentadas pelo iptraf podem ser relacionadas a uma sessão, uma interface de rede, ou a um protocolo. No contexto deste trabalho, as métricas fornecidas pelo monitor que são de interesse são as relacionadas a interfaces de rede, possibilitando o diagnóstico do provisionamento de rede (CILIENDO; KUNIMASA, 2007; HOCH, 2010). A Figura 3.1 demostra o iptraf executando usando o parâmetro "-d eth0", que indica ao monitor que apresente informações relacionadas à

interface eth0.

Figura 3.1: Exemplo de execução do monitor iptraf



Fonte: autoria própria

As métricas de interesse da saída apresentada na Figura 3.1 são *Incoming rates*, *Outgoing rates* e *Total rates*. Estas métricas indicam a taxa (em kbits por segundo) com que a interface está recebendo dados, a taxa (em kbits por segundo) com que a interface está enviando dados e a taxa total (em kbits por segundo) de dados transitando pela interface, respectivamente. A Figura 3.1 exemplificou o caso de um arquivo sendo transferido da Internet à taxa de 10Mbits por segundo.

3.2.6 tc

O tc é um utilitário que permite a exibição e manipulação de configurações e variáveis de controle do tráfego das interfaces de rede. A métrica apresentada por esta ferramenta interessante ao diagnóstico de provisionamento é o tamanho da fila da interface de rede em determinado instante (THE LINUX MAN-PAGES PROJECT, 2014c). O exemplo abaixo demostra o tc executando com o parâmetro "-s qdisc show dev eth0", que indica a ferramenta que apresente informações estatísticas relacionadas a interface de rede eth0.

```
# tc -s qdisc show dev eth0
qdisc pfifo_fast 0: root refcnt 2 bands 3 priomap 1 2 2 2 1 2 0 0 1 1 1
 Sent 5009947 bytes 56906 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Da saída anterior, a métrica *backlog* indica a quantidade de bits (acompanhada de "b") e a quantidade de pacotes (acompanhada de "p") na fila da interface de rede.

3.2.7 /proc/net/dev

O arquivo /proc/net/dev contém informações sobre as interfaces de rede de um sistema que podem ser utilizadas para diagnóstico de provisionamento de rede. Os quadros abaixo exemplificam o conteúdo do arquivo /proc/net/dev em um instante (THE LINUX MAN-PAGES PROJECT, 2014b). O conteúdo do arquivo foi dividido em dois quadros em razão do número de colunas, o primeiro quadro apresenta as métricas relacionadas à recepção de dados nas interfaces de rede, enquanto o segundo apresenta as métricas relacionadas ao envio de dados.

```
# cat /proc/net/dev
Inter-| Receive                                                       [...]
 face | bytes     packets errs drop fifo frame compressed multicast[...]
  eth0: 127582135 99661   0    0    0    0     0          0         [...]
    lo: 339996    2702    0    0    0    0     0          0         [...]
```

```
[...]
[...] | Transmit
[...] | bytes   packets errs drop fifo colls carrier compressed
[...]   5485090 57569   0    0    0    0     0       0
[...]   339996  2702    0    0    0    0     0       0
```

A coluna *bytes* informa o número total de *bytes* de dados transmitidos ou recebidos pela interface de rede (THE LINUX MAN-PAGES PROJECT, 2014b), e pode ser utilizada para estimar a taxa de transmissão ou recebimento através da diferença entre medições em intervalos regulares.

3.3 Armazenamento de Séries Temporais

Como visto na Seção 3.2, monitores podem ser configurados para coletar e/ou exibir métricas em intervalos de tempo uniforme. Uma sequência formada por estas medições é denominada Série Temporal (*Time Series* - TS) (DERI; MAINARDI; FUSCO, 2012). A TS de um determinado intervalo de tempo pode ser utilizada, junto de suas propriedades como média e desvio-padrão, para analisar as medições realizadas e, a um nível mais alto, para diagnosticar o provisionamento de recursos (PFITSCHER; PILLON; OBELHEIRO, 2014). Um desafio que surge do uso de TSs é a necessidade de armazenar cada medição para posterior análise, considerando que uma TS pode ter um valor significativamente grande de medições. Existem diversas ferramentas que podem ser utilizadas para esta tarefa de armazenamento, as subseções a seguir apresentam as de uso mais comum.

3.3.1 Bancos de Dados Relacionais

Os Bancos de Dados Relacionais (BDR) vêm sendo, por anos, utilizados para armazenar dados permanentemente. Nos BDRs, dados com características uniformes são organizados em tabelas associadas por relacionamentos. Cada tabela é dividida logicamente em linhas e colunas, sendo cada linha identificada por uma chave primária única (DERI; MAINARDI; FUSCO, 2012). Para armazenar uma TS em que cada elemento possui cada uma das métricas necessárias ao diagnóstico de provisionamento pode-se criar uma tabela em que cada linha possui colunas para as métricas e tem como chave primária o instante da medição. O uso de BDRs para o propósito de armazenar TSs, entretanto, deve considerar duas características dos BDRs (DERI; MAINARDI; FUSCO, 2012):

- A cada novo intervalo de medição, a tabela é acrescida de novos dados que incrementam a cardinalidade da tabela. A consequência disto é que tanto a cardinalidade quanto o espaço em disco aumentam linearmente com o número de medições realizadas. Por exemplo, caso a cada segundo for realizada uma medição e armazená-la em uma tabela, ao final do período de um dia, 86400 novas linhas terão sido inseridas na tabela;
- Quando os índices das tabelas tornam-se muito grandes para serem armazenados na *cache*, a obtenção de dados do BDR torna-se lenta, prejudicando o desempenho da aplicações dependentes do BDR e impondo um *overhead* sobre o sistema operacional.

3.3.2 RRDTool

O *Round Robin Database Tool* (RRDTool) é uma ferramenta especialista para armazenamento de TSs. Sua implementação é realizada através de *buffers* circulares persistidos em arquivos, onde os dados são armazenados de acordo com o *timestamp* no momento da inserção. No momento da criação de um banco de dados (BD) especifica-se o número máximo de elementos que a TS pode ter e com que frequência novos elementos serão inseridos (RRDTOOL, 2014). Por exemplo, um BD pode ser criado com capacidade para armazenar 60 elementos, com um novo elemento sendo inserido a cada segundo. Este BD tem a capacidade de armazenar medições relativas a um período de um minuto; no momento em que a inserção do sexagésimo primeiro elemento for requisitada, o elemento mais antigo é substituído pelo novo, conservando a cardinalidade da TS. Este exemplo explicita uma das qualidades do RRDTool: o tamanho de um arquivo de BD é constante ao longo do tempo. Assim como os Bancos de Dados Relacionais, algumas características do RRDTool devem ser consideradas quanto ao seu uso (DERI; MAINARDI; FUSCO, 2012):

- O número de arquivos de BD do RRDTool é o mesmo que o número de TSs sendo armazenadas, ou no caso deste trabalho, o número de métricas que devem ser armazenadas;
- A manipulação das TSs envolve a abertura, o salvamento e o fechamento de arquivos. O desempenho na realização desta tarefa depende do número de TSs que necessitam manipulação, e este número de TSs e a frequência da manipulação podem tornar o uso do RRDTool impraticável;
- O RRDTool é orientado a arquivos, portanto seu desempenho não pode exceder o desempenho do sistema de arquivos e do disco utilizados. Apesar de o diagnóstico de provisionamento não ser realizado para o disco, a utilização excessiva deste recurso pode impactar significativamente no desempenho de outros recursos (processador, memória).

3.3.3 TSDB

O *Time Series Data Base* (TSDB) é outra ferramenta especialista para armazenamento de TSs. As ferramentas apresentadas anteriormente podem tratar apenas um número

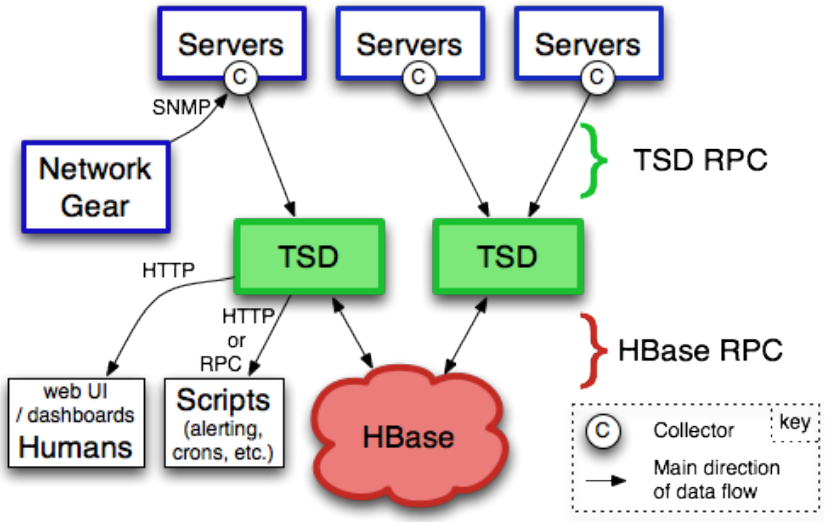
limitado de TSs, em razão de seu desempenho não ser satisfatório quando o número de TSs é consideravelmente grande (ex. 100 mil ou mais). Esta foi a principal motivação para o desenvolvimento do TSDB, a capacidade de tratar milhões de TSs com um desempenho que permita realizar a inserção e o obtenção de dados das TSs com menor duração possível (DERI; MAINARDI; FUSCO, 2012).

Em razão do número elevado de TSs que o TSDB foi projetado para tratar, seus autores julgaram necessária a compressão de dados para evitar que o tamanho do banco de dados torne-se excessivamente grande. Esta característica deve ser considerada quando do uso do TSDB para um número pequeno de TSs, pois os processos de compressão e descompressão (que não podem ser desabilitados) podem fazer com que o desempenho seja inferior a de outras ferramentas (DERI; MAINARDI; FUSCO, 2012).

3.3.4 OpenTSDB

O OpenTSDB é um banco de dados distribuído e escalável para armazenamento de TSs. A arquitetura desta ferramenta possibilita a obtenção de métricas de diversos dispositivos e a centralização destes em um banco de dados para monitoração pelo administrador através de gráficos e estatísticas (OPENTSDB, 2014). Em razão do propósito do projeto desta ferramenta e suas características, ilustradas na Figura 3.2, torna-se desaconselhável seu uso neste trabalho, que armazena dados referentes somente ao próprio sistema onde está sendo executado.

Figura 3.2: Arquitetura do OpenTSDB



Fonte: (OPENTSDB, 2014)

3.4 Considerações do Capítulo

Este capítulo apresentou inicialmente os princípios de monitoração em ambientes computacionais. Monitoração foi definida como uma técnica de observação das atividades de um ambiente computacional, seja este uma rede, um servidor ou uma máquina virtual (MV).

Foram apresentadas e estudadas, posteriormente, algumas ferramentas para monitoração (denominadas monitores) que permitem observar os recursos e obter as métricas de desempenho utilizadas pelo modelo de diagnóstico que será instanciado pela ferramenta. Foram também estudadas algumas ferramentas para o armazenamento dos dados coletados durante a monitoração. Alguns destes monitores e uma ferramenta de armazenamento serão usados pela ferramenta objetivo do trabalho. As ferramentas escolhidas e as justificativas são apresentadas na Seção 4.2 do capítulo seguinte.

4 Ferramenta REPD

Este capítulo descreve a ferramenta desenvolvida, denominada REPD (REsource Provisioning Diagnosis), segundo o objetivo deste trabalho. As seções estão organizadas da seguinte forma. A Seção 4.1 apresenta a análise de requisitos funcionais e não-funcionais. A Seção 4.2 define a linguagem de programação, as ferramentas e outros recursos para monitoração e armazenamento de séries temporais das quais a REPD depende. A Seção 4.3 descreve cada um dos componentes da REPD. A Seção 4.4 realiza algumas considerações sobre o uso da REPD e descreve os seus fluxos de execução. Por fim, a Seção 4.5 descreve o processo de adição de um novo modelo de diagnóstico a REPD.

4.1 Requisitos

Como previamente apresentado, este trabalho objetiva desenvolver uma ferramenta para diagnóstico de provisionamento de recursos em nuvens IaaS. Mais especificamente, a ferramenta deve ser capaz de coletar e armazenar métricas de desempenho que permitem aplicar um modelo de diagnóstico baseado na análise de séries temporais (como o modelo proposto por (PFITSCHER; PILLON; OBELHEIRO, 2014)). Considerando este objetivo, foram definidos alguns requisitos funcionais para a REPD:

- Capacidade de coletar métricas de desempenho através de monitores ou técnicas de monitoramento;
- Capacidade de armazenar séries temporais das métricas para aplicação de modelos de diagnóstico baseados na análise destas;
- Capacidade de aplicar modelos de diagnóstico, incluindo o cálculo de médias, desvios-padrão, entre outros.

Pode-se, ainda, acrescentar requisitos de ordem prática. Estes requisitos não estão estritamente relacionados ao funcionamento da REPD, entretanto, apresentam características desejáveis:

- Facilidade de implantação: Este requisito está associado à dificuldade de instalação e configuração da REPD. Entende-se que esta dificuldade aumenta quando a implantação da REPD depende da instalação de diversas outras ferramentas não fornecidas normalmente com o sistema operacional. Isto orienta a escolha de ferramentas auxiliares, considerando que deve-se minimizar a dificuldade de implantação sem prejudicar o desempenho ou capacidade da REPD;

- Baixo *overhead*: Este requisito está associado a uma característica indispensável aos monitores. Como a REPD consiste (em parte) em um monitor, seu *overhead* deve ter impacto insignificante sobre a utilização de recursos, ou seu diagnóstico será de pouco proveito;

- Modularidade: Este requisito está associado à organização da arquitetura da REPD. Entende-se por arquitetura modular uma em que seus componentes são módulos com diferentes propósitos com alta coesão e fraco acoplamento. A coesão é uma medida que indica se um componente tem um função bem definida no sistema. O acoplamento, por sua vez, é uma medida que indica o quanto dois componentes relacionados precisam conhecer de detalhes internos do funcionamento do outro (SOMMERVILLE, 2011). Um sistema altamente coeso e fracamente acoplado permite que seus componentes sejam mais facilmente substituídos. No caso da REPD isto possibilita substituir: o modelo de diagnóstico, no caso do surgimento de novos modelos; a ferramenta de armazenamento, agregando escalabilidade; e os módulos de coleta de métricas, agregando portabilidade por permitir que um mesmo módulo seja implementado segundo as ferramentas disponíveis em determinado sistema operacional.

4.2 Dependências

Nesta seção são citadas e justificadas as linguagens de programação, ferramentas e outros recursos escolhidos para dar suporte à REPD. Cada um dos tópicos a seguir apresenta uma ou mais escolhas relacionadas:

- Ferramenta para gerência de séries temporais: SQLite – O SQLite representa um banco de dados relacional e foi escolhido principalmente por possuir desempenho

próximo a de outras opções como MySQL (BUEVICH et al., 2013), além de sua implantação ser simples pelo fato de não necessitar configuração e tampouco de um servidor para atender requisições (SQLITE, 2014). Inicialmente considerou-se a utilização do TSDB, mas a compressão que este realiza e a escassez de documentação tornaram sua utilização impraticável.

- Linguagem de programação: Python – As características do Python que justificam sua utilização são (LUTZ, 2009):

  - Facilidade de implantação: O interpretador Python vem pré-instalado em boa parte dos sistemas operacionais (ex. Linux e Mac OSX) e sua implantação em outros (ex. Windows) é simples. Esta característica tem relação com o requisito de facilidade de implantação da ferramenta;
  - Legibilidade: A sintaxe do linguagem tem enfoque na legibilidade e coerência. No contexto deste trabalho é interessante que os códigos desenvolvidos sejam de fácil entendimento para permitir a reutilização e manutenção por trabalhos futuros;
  - Portabilidade: A maioria dos programas em Python podem ser executados sem alterações nos principais sistemas operacionais. Esta portabilidade pode facilitar a adaptação da REPD para outros sistemas operacionais (considerando que o desenvolvimento atual é voltado para o Linux), pois apenas módulos que tratam de ferramentas específicas de sistemas operacionais (como monitores) necessitarão de substituição;
  - Suporte a bibliotecas: O interpretador Python vem acompanhado de uma coleção ampla de funções (ou bibliotecas) pré-instaladas e portáveis, que suportam a realização de diversas tarefas de programação. Uma das bibliotecas pré-instaladas de interesse a este trabalho é a que permite a comunicação com um banco de dados SQLite.

- Ferramentas e recursos para monitoração: mpstat, /proc/meminfo, tc e /proc/net/-dev – A escolha das ferramentas de monitoração é guiada pelas métricas que precisam ser observadas, que por sua vez dependem das necessidades do modelo de diagnóstico. Como não é possível definir monitores genéricos, a escolha destes se baseou nas métricas usadas no modelo de (PFITSCHER; PILLON; OBELHEIRO, 2014).

mpstat permite monitorar as duas métricas de processador: a utilização de processador e o percentual de *steal*. /proc/meminfo permite monitorar as métricas de memória residente e de memória reservada. tc permite monitorar o tamanho da fila na interface de rede. /proc/net/dev permite obter o consumo de banda;

4.3 Componentes

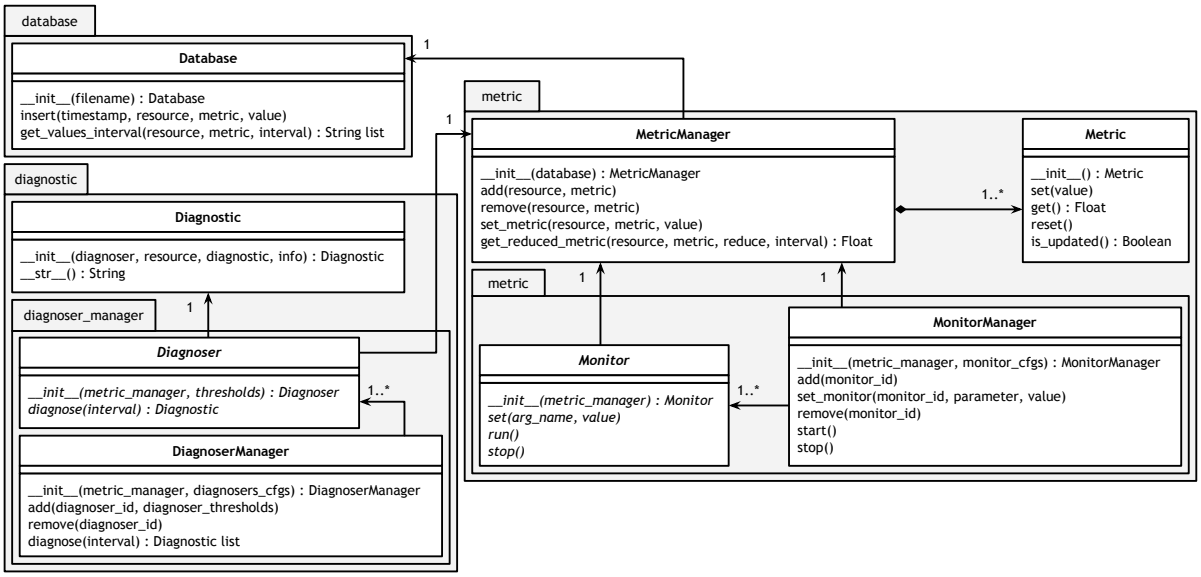
Nesta seção são apresentados os componentes da ferramenta REPD. Em razão da ferramenta ter sido desenvolvida na linguagem Python, cada um de seus componentes pode ser categorizado como módulo, pacote ou *script* principal. Módulos são elementos que definem funções e/ou classes que podem ser usadas por outros módulos. Pacotes são uma categoria especial de módulos que podem conter outros módulos além de definir funções e/ou classes; seu uso primário está no agrupamento e estruturação de módulos com propósitos semelhantes (por exemplo o pacote metric agrupa módulos voltados a coleta de métricas). *Scripts* principais são ainda outra categoria especial de módulos que definem uma função a ser chamada diretamente pelo interpretador da linguagem. Esta função, comumente denominada *main*, tem por papel coordenar outras classes e funções de forma a realizar a lógica de negócio da aplicação (LUTZ, 2009).

A Figura 4.1 apresenta os componentes da ferramenta e suas relações através de um diagrama de pacotes. Os módulos puros foram abstraídos, sendo mostradas apenas as classes definidas por estes. Os *scripts* principais e o módulo statistics não foram exibidos, pois contêm apenas funções. Nas subseções a seguir são descritos cada um dos componentes da REPD (incluindo os não presentes na Figura 4.1) e são realizadas considerações sobre como o modelo de diagnóstico foi instanciado sobre estes.

4.3.1 Módulo **database**

O módulo database é responsável pela comunicação entre a ferramenta e o banco de dados. Este módulo define a classe Database que, em razão das tecnologias escolhidas para desenvolvimento, utiliza o módulo padrão da linguagem sqlite3 (THE PYTHON SOFTWARE FOUNDATION, 2014) para gerenciar um banco de dados SQLite, apresentado anteriormente na Seção 4.2. Esta classe foi implementada para armazenar os valores coletados de cada métrica em uma tabela separada. Cada tabela possui três campos: um identificador do

Figura 4.1: Diagrama de classes da ferramenta



Fonte: Autoria própria

registro na tabela, o *timestamp* e o valor da métrica obtido naquela coleta. O nome de cada tabela é definido pelo concatenamento entre o nome do recurso ao qual a métrica é referente e o nome da métrica (ex. à tabela que armazena o percentual de memória residente foi atribuído o nome de memory_resident)

4.3.2 Pacote **metric**

O pacote metric agrupa os módulos responsáveis pela coleta, tratamento e envio das métricas para o módulo database, além de definir a classe Metric, que guarda o valor de uma métrica. O módulo statistics (não mostrado na Figura 4.1) define as funções para redução de métricas de um intervalo (média, desvio-padrão, entre outros). O módulo metric_manager define uma classe MetricManager, que gerencia diversas instâncias da classe Metric associando-as a recursos e as envia para uma instância da classe Database quando em momentos adequados.

O pacote monitor_manager define a classe MonitorManager, que gerencia as classes dos módulos contidos neste pacote, chamadas monitoras. Cada monitora é implementada seguindo a interface Monitor apresentada na Figura 4.1. Para realizar o objetivo de instanciar o modelo de diagnóstico apresentado na Seção 2.3, foram implementadas quatro classes monitoras, de nomes mpstat, procmeminfo, procnetdev e tc que encapsulam o monitor mpstat, o pseudo-arquivo /proc/meminfo, o pseudo-arquivo /proc/net/dev e a

ferramenta tc, respectivamente.

As instâncias da classe monitora são criadas a partir da função __init__, que recebe um MetricManager. Parâmetros de configuração como intervalo entre medições e interface de rede são informados à monitora através da função set. Cada monitora executa em uma *thread* separada (iniciada a partir da função run e finalizada pela stop) e interage periodicamente com o monitor, o pseudo-arquivo ou a ferramenta encapsulados obtendo valores de métricas e repassando-os para o MetricManager.

4.3.3 Pacote diagnostic

O pacote diagnostic agrupa os módulos responsáveis pelo diagnóstico dos recursos e define a classe Diagnostic, que representa um diagnóstico, contendo informações como o próprio diagnóstico (superprovisionado, subprovisionado ou adequado), a diagnosticadora que o gerou, e o recurso a que este se refere.

O pacote diagnoser_manager define a classe DiagnoserManager, que gerencia as classes dos módulos contidos neste pacote, chamadas diagnosticadoras. Cada diagnosticadora é implementada seguindo a interface Diagnoser apresentada na Figura 4.1. Para atingir o objetivo de instanciar o modelo de diagnóstico apresentado na Seção 2.3, foram implementadas três classes diagnosticadoras, de nomes pftischer_cpu, pfitscher_memory e pfitscher_network, que aplicam o modelo para processador, memória e rede, respectivamente.

A interface Diagnoser define duas funções. A função __init__ deve criar uma nova instância da diagnosticadora, recebendo como parâmetros um MetricManager e os limiares do diagnóstico e inicializando variáveis necessárias para o funcionamento da diagnosticadora. A função diagnose deve receber um intervalo do qual gerará o diagnóstico considerando métricas reduzidas obtidas do MetricManager.

4.3.4 *Scripts* principais repdd e repd

A REPD possui dois *scripts* principais com dois papéis diferentes e complementares. O repdd realiza a lógica de negócio dos dois primeiros requisitos funcionais definidos na Seção 4.1, a saber, a coleta das métricas referentes ao modelo de diagnóstico e o armazenamento destas como séries temporais para a aplicação do modelo. O repd realiza a lógica de

negócio do terceiro requisito funcional, a aplicação do modelo de diagnóstico sobre as métricas obtidas através do repdd. A configuração destes *scripts* principais, incluindo o diretório para armazenamento do banco de dados, as monitoras e as diagnosticadoras a serem executadas, é definida através de arquivos de configuração no formato JSON localizados dentro do diretório settings da REPD. A seguir são apresentados exemplos de arquivos de configuração, o primeiro define as configurações do módulo database, o segundo do pacote metric e o terceiro do pacote diagnostic.

```
{
    "filename": "database/db.sl3"
}
```

Neste primeiro arquivo (denominado database.cfg) é apenas definido o local e o nome do arquivo para o armazenamento do banco de dados, neste exemplo o local definido é o diretório database (a partir do diretório da ferramenta) e o nome é db.sl3.

```
[
    {   "id": "cpu",
        "argv": {"interval": 5},
        "monitors": [
            {"id": "mpstat", "args": ["interval"]}
        ]
    },
    {   "id": "memory",
        "argv": {"interval": 5},
        "monitors": [
            {"id": "procmeminfo", "args": ["interval"]}
        ]
    },
    {   "id": "network",
        "argv": {"interval": 5, "interface": "eth0"},
        "monitors": [
            {"id": "procnetdev", "args": ["interval", "interface"]},
            {"id": "tc", "args": ["interval", "interface"]}
        ]
    }
]
```

Este segundo (denominado monitor_manager.cfg) define grupos de monitoras por recursos e seus parâmetros. Cada objeto JSON dentro da lista mais externa define um

grupo de monitoras. O campo id identifica o grupo e é normalmente o nome do recurso que é monitorados pelas monitoras. O campo argv define um conjunto de pares que indicam os parâmetros e seus valores a serem utilizados pelas monitoras. A lista definida por monitors pode conter vários objetos JSON, cada um informa uma das monitoras pertencentes ao grupo e quais parâmetros devem ser informados a esta. Neste exemplo foram definidos três grupos, todos possuem como parâmetro o intervalo entre medições. O último também possui a interface de rede. No primeiro grupo encontra-se apenas a monitora mpstat que utiliza o único parâmetro do grupo. O segundo grupo agrupa apenas a monitora procmeminfo que utilizam o intervalo entre medições. O terceiro grupo contém duas monitoras procnetdev e tc, as duas utilizam o intervalo entre medições e a interface de rede como parâmetros.

```
[
    "pfitscher_cpu",
    "pfitscher_memory",
    "pfitscher_network"
]
```

O último arquivo (denominado diagnoser_manager.cfg) define quais as diagnosticadoras devem ser executadas através de uma lista de nomes. Neste exemplo são listadas três diagnosticadoras, que são as que aplicam os modelos de processador, memória e rede de (PFITSCHER; PILLON; OBELHEIRO, 2014). O funcionamento dos *scripts* principais ou, em outras palavras, da REPD, é apresentado em maiores detalhes na Seção 4.4.

4.4 Fluxos de Execução

Como descrito na Seção 4.3 a ferramenta desenvolvida possui dois *scripts* principais. O primeiro *script*, de nome repdd, foi projetado e desenvolvido para ser executado como um *daemon* – um programa executado como um processo em segundo plano e normalmente iniciado junto ao sistema operacional – com o propósito específico de coletar valores de métricas para que posteriormente seja possível realizar o diagnóstico do provisionamento. O *script* repd, por sua vez, foi desenvolvido para ser executado diretamente e sob-demanda pelo usuário para fornecer o diagnóstico do provisionamento, usando os valores coletados pelo repdd.

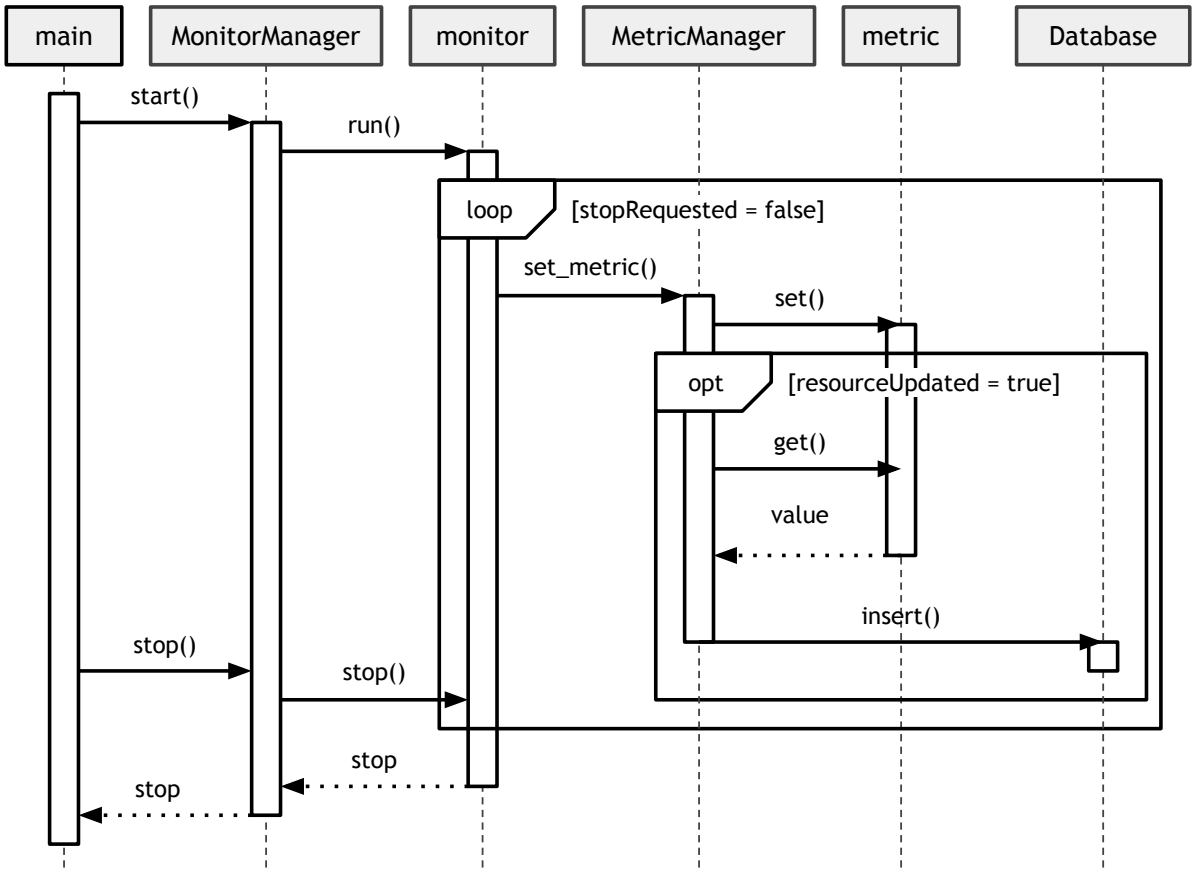
Cada um destes *scripts* principais irá definir um fluxo de execução interagindo

com subconjuntos distintos dos componentes descritos na Seção 4.3. As subseções a seguir apresentam estes fluxos de execução pela enumeração de passos e diagramas de sequencia. A Subseção 4.4.1 apresenta o fluxo definido pelo repdd e a Subseção 4.4.2 o definido pelo repd.

4.4.1 Repdd

A seguir é apresentado o fluxo de execução definido pelo *script* principal repdd. A Figura 4.2 representa o diagrama de sequência da interação entre os componentes (alguns componentes foram omitidos para facilitar a visualização).

Figura 4.2: Diagrama de sequência do fluxo de execução definido por repdd



Fonte: Autoria própria

1. Para iniciar o fluxo de execução o interpretador da linguagem executa o *script* principal repdd, que por sua vez executa sua função main;

2. A função main cria uma instância da classe Database (informando a localização do banco de dados SQLite a ser utilizado), uma da MetricManager (referenciando a

instância Database) e uma da MonitorManager (informando a instância MetricManager e uma lista contendo as monitoras que devem ser criadas);

3. O MonitorManager criado instancia então cada uma das monitoras da lista com uma referência para o MetricManager e informando parâmetros como intervalo entre medições e interface de rede para monitoração, e após isso aguarda comando da main para iniciar a monitoração;

4. A main inicia a monitoração e aguarda indefinidamente um comando do usuário para encerrá-la;

5. O MonitorManager inicia então cada uma das monitoras (que são *threads*) que passam a executar os subpassos seguintes até que o usuário encerre a monitoração:

   (a) Após o intervalo de medição definido, a monitora mede as métricas de sua responsabilidade e envia os valores para o MetricManager;

   (b) Agrupando as métricas por recursos, o MetricManager identifica quando todas as métricas de um recurso estão atualizadas e as envia para a Database associando um *timestamp* às métricas;

   (c) Recebidos os valores pela Database, esta insere os valores das métricas associados aos *timestamps* em tabelas separadas para cada uma das métricas;

6. Mediante a requisição do usuário para encerramento da monitoração, o MonitorManager encerra cada umas da monitoras;

7. Com todas as monitoras encerradas, a função main finaliza o fluxo de execução.

A seguir é apresentada a saída referente a execução deste fluxo. A primeira linha é exibida na inicialização do *script* principal e, a segunda quando o usuário solicita seu encerramento.

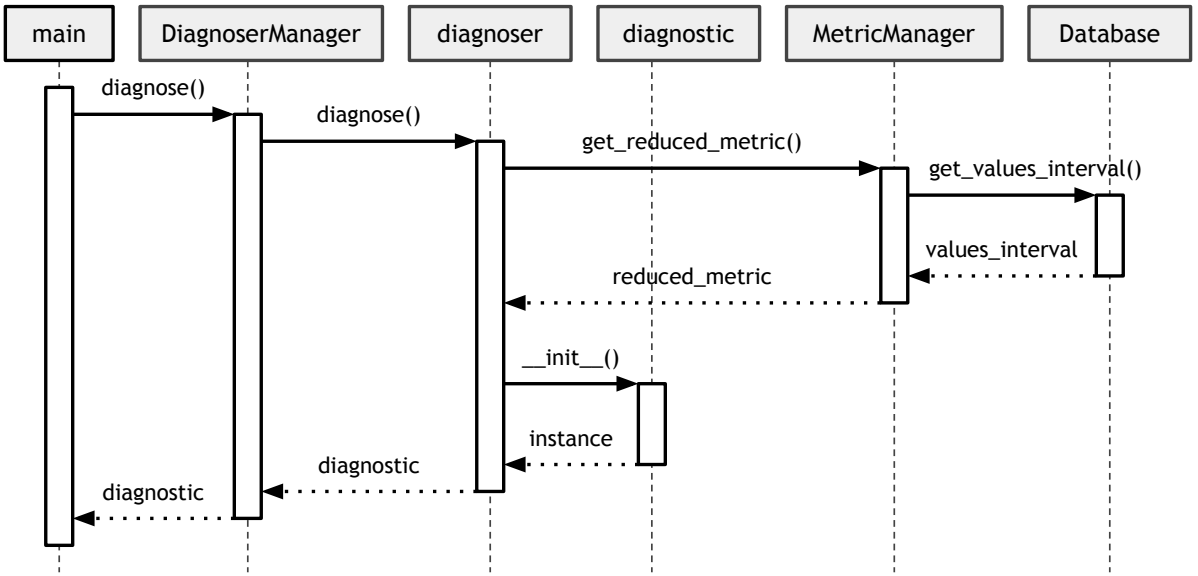
```
Press control-c to quit
Quitting, this may take a while
```

4.4.2 Repd

Semelhante à Subseção 4.4.1, esta subseção apresenta o fluxo de execução definido pelo *script* principal repd. A Figura 4.3 mostra o diagrama de sequência das interações entre

os componentes (alguns componentes foram omitidos para facilitar a visualização).

Figura 4.3: Diagrama de sequência do fluxo de execução definido por repd



Fonte: Autoria própria

1. Para iniciar o fluxo de execução, o interpretador da linguagem executa o *script* principal repdd, que por sua vez executa sua função main;

2. A função main cria uma instância da classe Database (informando a localização do banco de dados SQLite a ser utilizado), uma da MetricManager (referenciando a instância Database) e uma da DiagnoserManager (informando a instância MetricManager e uma lista contendo as diagnosticadoras que devem ser criadas);

3. O DiagnoserManager criado instancia então cada uma das diagnosticadoras da lista, informando uma referência para o MetricManager;

4. Após a criação do DiagnoserManager, a função main requisita o diagnóstico dos recursos referentes ao período de interesse do usuário a este gerente, que repassa para as diagnosticadoras;

5. Cada uma das diagnosticadoras realiza então o diagnóstico segundo o modelo de (PFITSCHER; PILLON; OBELHEIRO, 2014), requisitando os valores reduzidos de métricas (média, desvio-padrão, entre outros) ao MetricManager e comparando-os aos valores dos seus limiares;

6. Os diagnósticos são então retornados das diagnosticadoras ao DiagnoserManager, e deste à função main, que exibe ao usuário os diagnósticos;

7. Com os diagnósticos exibidos ao usuário, a função main finaliza o fluxo de execução.

A seguir é apresentada a saída referente à execução deste fluxo. O comando utilizado informa dois *timestamps* que delimitam o período a ser diagnosticado. Para que o diagnóstico seja realizado o *script* principal repdd deveria estar em execução no período informado.

```
$ python3 repd.py 0 300
=== CPU ===
diagnostic: OVERPROVISIONED
diagnoser: pfitscher_cpu
info:
    - average utilization: 29.71%
    - utilization above 95%: 0.00%
    - average steal: 0.01%

=== MEMORY ===
diagnostic: UNDERPROVISIONED
diagnoser: pfitscher_memory
info:
    - average resident: 53.74%
    - average committed: 191.39%

=== NETWORK ===
diagnostic: CORRECT
diagnoser: pfitscher_network
info:
    - average queue size: 0.00 packets
    - CV for transmission rate: 0.00
```

4.5 Estendendo a REPD

Nesta seção será apresentado um exemplo de como a ferramenta REPD pode ser estendida. Esse exemplo adiciona um novo modelo de diagnóstico, que requer o uso de um monitor diferente. Mais amplamente, a extensão consiste em dois passos. O primeiro é a adição de uma nova monitora, para coletar as métricas necessárias ao novo modelo. O segundo é a adição de uma nova diagnosticadora, para analisar as métricas coletadas e dar o diagnóstico. As subseções a seguir apresentam estes dois passos em maiores detalhes,

nestas foi considerada a adição de um modelo de diagnóstico para o consumo de espaço em disco denominado marques_disk. Neste modelo são definidos os seguintes níveis de provisionamento:

- subprovisionado: quando o espaço ocupado é superior a 90%;
- adequado: quando o espaço ocupado fica entre 50 e 90%;
- superprovisionado: quando o espaço ocupado é inferior ou igual a 50%.

Este diagnóstico foi criado com o intuito de exemplificar os passos para a extensão da REPD, e em função disto sua validade não é discutida.

4.5.1 Adicionando uma nova monitora

O primeiro passo na adição de um novo modelo à REPD é a identificação de quais são as métricas necessárias para aplicá-lo e como estas podem ser obtidas. No caso do modelo marques_disk, a única métrica necessária é o percentual de espaço em uso em disco. No Linux, esta métrica pode ser obtida através do comando df. A seguir é apresentado um exemplo da saída deste comando; o valor de interesse para o modelo marques_disk é o da coluna Use% da linha referente ao sistema de arquivos /dev/sda1.

```
Filesystem          1K-blocks      Used Available Use% Mounted on
/dev/sda1            41251136   5134392  34380832  13% /
udev                   245920        12    245908   1% /dev
tmpfs                   50180       352     49828   1% /run
```

Com o monitor definido, é possível então desenvolver uma monitora que o encapsule. Como comentado na Seção 4.3, as monitoras devem implementar a interface de programação monitor, que define três funções. O quadro a seguir mostra a diagnosticadora implementada no arquivo df.py dentro do diretório metric/monitor_manager da REPD. A função __init__ é utilizada para criar instâncias da monitora, esta inicializa variáveis úteis à monitoração e informa ao MetricManager as métricas que irá monitorar (neste caso apenas o espaço em uso do disco) através da função add. A função set configura o valor dos parâmetros intervalo entre medições e sistema de arquivos quando as chaves apropriadas são informadas. A função run executa a monitoração propriamente dita. Inicialmente é verificado se todas os parâmetros foram inicializados, e em seguida é realizado um laço

de repetição até que o encerramento da execução seja requisitado; neste laço, a monitora fica inativa durante o intervalo definido, depois executa o comando df e processa a saída enviando o valor obtido da métrica ao MetricManager. Ao sair do laço, e antes de encerrar a execução, é informado ao MetricManager que a métrica não será mais fornecida (em função do encerramento iminente) através da chamada da função remove. A função stop marca a variável stop_flag como True, resultando na conclusão do laço de repetição em run.

```
import subprocess
import threading
import time

class df(threading.Thread):
    def __init__(self, metric_manager):
        super(df, self).__init__()
        self.stop_flag = False
        self.interval = None
        self.filesystem = None
        self.mm = metric_manager
        self.mm.add('disk', 'used')

    def stop(self):
        self.stop_flag = True

    def set(self, parameter, value):
        if parameter == 'interval':
            self.interval = value
        elif parameter == 'filesystem':
            self.filesystem = value

    def run(self):
        if self.interval is None or self.filesystem is None:
            raise RuntimeError
        while not self.stop_flag:
            time.sleep(self.interval)
            lines = subprocess.check_output(['df']).splitlines()
            for line in lines:
                ls = line.decode('ascii').split()
                if ls[0] == self.filesystem:
```

```
                    self.mm.set_metric('disk', 'used', float(ls[4][:-1])
                        )
        self.mm.remove('disk', 'used')
```

Para que a REPD utilize a nova monitora desenvolvida, deve-se adicionar um grupo ao arquivo de configuração monitor_manager.cfg, o quadro a seguir exemplifica este grupo. Os parâmetros definidos são intervalo entre medições igual a 5 segundos e sistema de arquivos igual a /dev/sda1. Conforme informado no objeto JSON da lista monitors, a monitora df deve receber ambos os parâmetros do grupo.

```
[
    {   "id": "disk",
        "argv": {"interval": 5, "filesystem": "/dev/sda1"},
        "monitors": [
            {"id": "df", "args": ["interval", "filesystem"]}
        ]
    }
]
```

Realizados os passos acima descritos, a REPD irá executar a nova monitora quando o *script* principal repdd for executado.

4.5.2 Adicionando uma nova diagnosticadora

Com a monitora implementada, pode-se então desenvolver a diagnosticadora que irá utilizar as métricas coletadas pela primeira. As diagnosticadoras devem implementar a interface de programação diagnoser, como comentado anteriormente na Seção 4.3. Esta interface define duas funções, __init__ e diagnoser. O quadro a seguir mostra como estas foram implementadas na nova diagnosticadora, denominada marques_disk. A função __init__ cria uma nova instância, e inicializa os limiares com os valores necessários para o diagnóstico. A função diagnose realiza o diagnóstico de um intervalo. Inicialmente esta requisita a média da métrica *used* do recurso *disk* e cria um novo objeto Diagnostic; logo após, ela compara o valor reduzido da métrica aos limiares e atribui o diagnóstico; por fim, ela define como informação adicional a média de espaço em uso do disco para o intervalo e retorna o diagnóstico gerado.

```
import diagnostic
```

```
class marques_disk:
    def __init__(self, metric_manager):
        self.mm = metric_manager
        self.t_used_low = 50
        self.t_used_high = 90

    def diagnose(self, interval):
        v_used_mean = self.mm.get_reduced_metric('disk', 'used', ('mean
            ', ), interval)
        diag = diagnostic.Diagnostic(diagnoser='marques_disk', resource
            ='disk')

        if v_used_mean is None:
            diag.diagnostic = None
        elif v_used_mean <= self.t_used_low:
            diag.diagnostic = diagnostic.OVER
        elif v_used_mean <= self.t_used_high:
            diag.diagnostic = diagnostic.CORRECT
        else:
            diag.diagnostic = diagnostic.UNDER

        if v_used_mean is not None:
            diag.info = '\n    - average used: %.2f%%' % v_used_mean
        return diag
```

O quadro a seguir exemplifica o conteúdo do arquivo de configuração diagnoser_manager.cfg para que marques_disk seja utilizada pela REPD.

```
[
    "marques_disk"
]
```

Concluídos estes passos, pode-se obter o diagnóstico para o modelo marques_disk pela execução do *script* principal repd, que gerará uma saída semelhante à do quadro a seguir.

```
=== DISK ===
diagnostic: OVERPROVISIONED
diagnoser: marques_disk
info:
    - average used: 13.00%
```

Na Seção 5.2 serão realizadas algumas considerações sobre o processo de extensão da REPD.

4.6 Considerações do Capítulo

Neste capítulo foi apresentada a ferramenta desenvolvida REPD. Inicialmente, descreveu-se os requisitos que foram considerados para o desenvolvimento, que são as capacidades de coletar as métricas de desempenho; armazenar séries temporais das métricas; e aplicar modelos de diagnóstico.

Foram também citadas e justificadas as dependências da REPD, sendo escolhida como linguagem de programação Python; como ferramentas de monitoração as ferramentas mpstat e tc e os pseudo-arquivos /proc/meminfo e /proc/net/dev (apresentadas na Seção 3.2); e como ferramenta para armazenamento o banco de dados relacional SQLite (apresentada na Seção 3.3).

Foram ainda descritos os componentes da ferramenta desenvolvidos e os dois fluxos de execução definidos pelos *scripts* principais repdd, que coleta valores de métrica para posterior diagnóstico, e repd, que realiza o diagnóstico de provisionamento. E, por fim, foi descrito o processo necessário para adição de novos modelos de diagnóstico a ferramenta, exemplificando cada um dos passos necessários.

No próximo capítulo analisa-se a efetividade da ferramenta REPD em realizar o diagnóstico do provisionamento e atender os requisitos pré-estabelecidos na Seção 4.1 deste capítulo.

5 Testes

Para validar a ferramenta REPD segundo os requisitos definidos na Seção 4.1 este capítulo apresenta os testes realizados voltados aos requisitos funcionais e descreve algumas características da REPD com o intuito de conformar com os requisitos não funcionais. A Seção 5.1 apresenta os testes dos requisitos funcionais e a Seção 5.2 descreve as características da REPD relacionadas aos requisitos não funcionais.

5.1 Requisitos Funcionais

Nesta seção são apresentados os testes realizados com a ferramenta REPD para verificar se esta possui as capacidades (coletar métricas, armazená-las e aplicar modelos de diagnóstico) definidas pelos requisitos funcionais descritos na Seção 4.1. Os testes foram realizados em um computador hospedeiro com processador AMD Phenom(tm) II X4 B93 (quatro núcleos de processamento), com 4GB de memória e largura de rede de 100 Mbps e uma máquina virtual (VM) com 1 processador virtual (VCPU), 512MB de memória e largura de rede de 100 Mbps. Ambos os ambientes executavam o sistema operacional Debian GNU/Linux 6.0.6, kernel Linux versão 2.6.32-5-amd64 e Xen versão 4.0.1-amd64.

Nos tópicos a seguir são descritas algumas características dos testes realizados:

- Foram realizados três testes para cada recurso computacional (processador, memória, rede), totalizando em nove;
- Cada um dos testes de um mesmo recurso possui diagnóstico diferente do outro, ou seja, um possui como diagnóstico provisionamento adequado, outro subprovisionamento e um terceiro superprovisionamento;
- Cada teste consiste na aplicação de uma carga de trabalho da qual tem-se conhecimento do diagnóstico segundo o modelo descrito na Seção 2.3;
- Cada teste teve a duração de cinco minutos e a REPD foi configurada para realizar medições em intervalos de cinco segundos, resultando assim em 60 medições por teste;

- A ferramenta REPD é considerada válida em um teste se apresentar o mesmo diagnóstico definido pelo modelo para a mesma carga de trabalho;

- Como os testes têm o propósito de avaliar o funcionamento da ferramenta e não de validar o modelo, não existe necessidade de coletar diversas amostras (rodadas de teste).

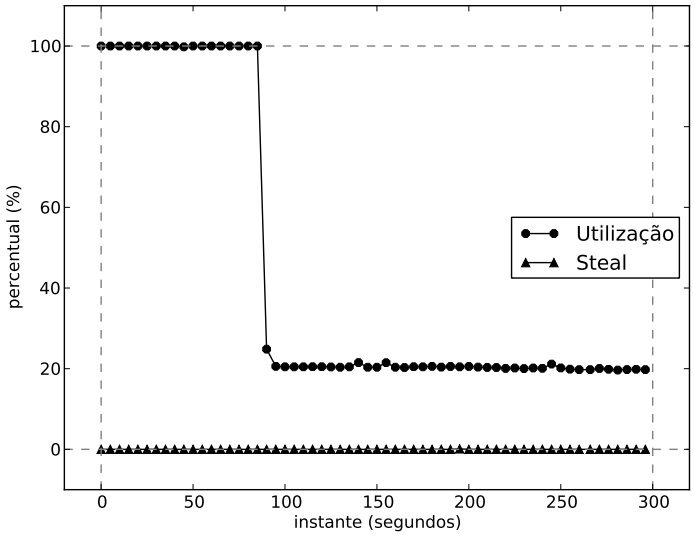
Nas subseções a seguir são descritos os testes realizados, cada uma é referente a um recurso computacional.

5.1.1 Processador

O primeiro teste relativo ao processador consistiu na aplicação de uma carga que leva a uma utilização do processador de aproximadamente 100% durante 90 segundos e de aproximadamente 20% no restante do teste (210 segundos). Para este teste, e para os outros em que são aplicadas cargas de trabalhos sobre o processador, foi usado um programa que varia entre dois estados, um em que executa instruções e outro em que fica ocioso. Através da razão entre o tempo de trabalho e de ociosidade é possível gerar uma carga que, se analisada em períodos mais longos que a soma dos dois, é semelhante à utilização parcial do processador. Como a VM era a única executando em um hospedeiro com quatro núcleos de processamento não era esperado percentual de *steal*. Realizando o cálculo aproximado da média do percentual de utilização encontra-se o valor de 44%. Já o índice de saturação do processador é de aproximadamente 30%. Em função do índice de saturação ser superior a 20%, o diagnóstico esperado é de provisionamento adequado. A Figura 5.1 mostra os valores do percentual de utilização e *steal* coletados pela REPD.

O diagnóstico fornecido pela ferramenta, de acordo com a saída a seguir, é de provisionamento adequado. Logo, neste primeiro teste a REPD é considerada válida.

Figura 5.1: Medições da REPD para o teste de processador provisionado adequadamente



Fonte: Autoria própria

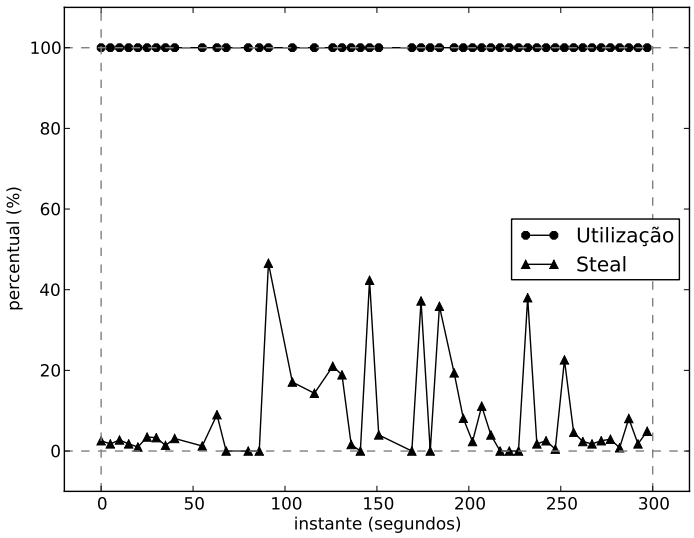
```
$ python3 repd.py 1415286634 1415286934
=== CPU ===
diagnostic: CORRECT
diagnoser: pfitscher_cpu
info:
    - average utilization: 44.30%
    - utilization above 95%: 30.00%
    - average steal: 0.00%
```

No segundo teste foram criadas outras três VMs, com 1 VCPU cada, para concorrer no escalonamento do processador físico do hospedeiro. Em cada VM foi aplicada uma carga de trabalho de 100% de utilização do processador. Isto foi realizado com o objetivo de obter um percentual de *steal* maior que 1%, visto que são cinco sistemas operacionais (contando com o hospedeiro) concorrendo por apenas quatro núcleos do processador físico. Em função do percentual de *steal* ser superior a 1% o diagnóstico esperado é de subprovisionamento.

A Figura 5.2 mostra os valores coletados pela REPD. Através desta nota-se que, apesar da REPD estar configurada para realizar medições em intervalos de cinco segundos, em alguns instantes este intervalo é maior, resultando em apenas 49 medições ao invés de 60. O comportamento observado ocorre em função do *steal* (quando uma

VCPU está pronta para execução, porém não é escalonada), a VM em que a ferramenta estava executando não foi escalonada a tempo de realizar a medição no intervalo correto (CHERKASOVA; GUPTA; VAHDAT, 2007).

Figura 5.2: Medições da REPD para o teste de processador subprovisionado



Fonte: Autoria própria

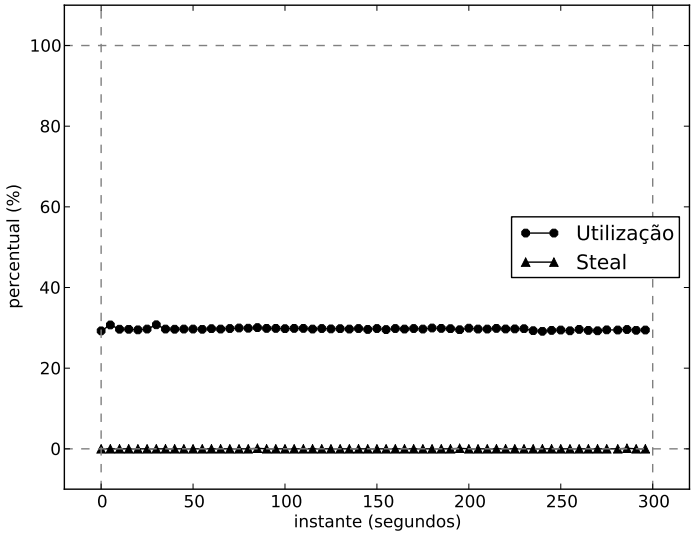
Apesar de a ferramenta ter realizado menos medições durante este teste, seu diagnóstico foi correto, de subprovisionamento (conforme a saída seguinte).

```
$ python3 repd.py 1415364681 1415364981
=== CPU ===
diagnostic: UNDERPROVISIONED
diagnoser: pfitscher_cpu
info:
    - average utilization: 100.00%
    - utilization above 95%: 100.00%
    - average steal: 8.37%
```

No último teste relacionado ao processador foi aplicada uma carga de trabalho de 30% de percentual de utilização de processador durante todo o período do teste. Neste teste a VM voltou a ser a única sendo executada, logo o percentual de *steal* esperado é aproximadamente 0%. Como todos os pontos da métrica utilização de CPU são aproximadamente 30%, o índice de saturação do processador é igual a 0%. Considerando estes valores, o diagnóstico esperado é de superprovisionamento. A Figura 5.3 mostra os valores coletados pela REPD, que segundo esperado apresentam a utilização do processador

próxima a 30%.

Figura 5.3: Medições da REPD para o teste de processador superprovisionado



Fonte: Autoria própria

O diagnóstico de provisionamento informado pela ferramenta corresponde ao esperado, superprovisionamento, conforme a saída seguinte.

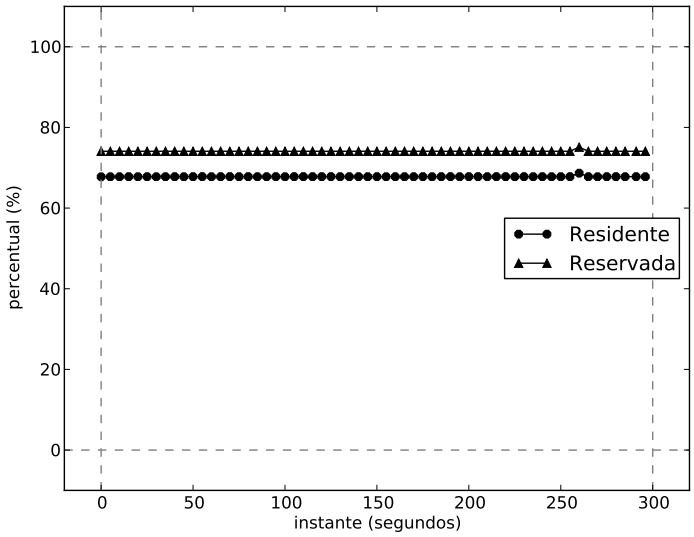
```
$ python3 repd.py 1415284509 1415284809
=== CPU ===
diagnostic: OVERPROVISIONED
diagnoser: pfitscher_cpu
info:
    - average utilization: 29.71%
    - utilization above 95%: 0.00%
    - average steal: 0.01%
```

5.1.2 Memória

Os testes realizados relacionados à memória consistiram na reserva e uso de percentuais fixos de memória. Para realizar esta tarefa foi usado um programa em que informados um número inteiro α e um percentual β, realiza a alocação de α páginas (de 4096 *bytes* cada) e atribui valores a β% destas, para que se tornem páginas sujas e sejam identificadas como memória residente pelo sistema operacional. No primeiro teste, foram reservadas aproximadamente 73500 páginas (≈56% da memória total), e destas 100% foram sujas. Esta carga, acrescida à reservada e suja normalmente pelo sistema operacional, resultou em

aproximadamente 67% de memória residente e 74% reservada. Este valor para memória reservada pode ser classificado como irrelevante, e o para residente como confortável. Em vista disso, o diagnóstico esperado é de provisionamento adequado. A Figura 5.4 mostra os valores das duas métricas coletados pela REPD.

Figura 5.4: Medições da REPD para o teste de memória provisionada adequadamente



Fonte: Autoria própria

A média da memória residente calculada pela ferramenta, de acordo com a saída a seguir, foi de 67.80% e da memória reservada foi de 74.10%. Considerando estes valores a REPD atribuiu o diagnóstico de provisionamento adequado, como era esperado.

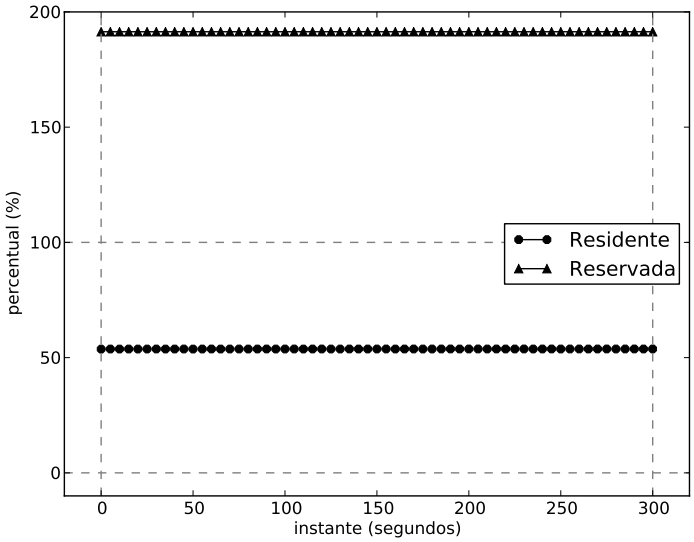
```
$ python3 repd.py 1415115536 1415115836
=== MEMORY ===
diagnostic: CORRECT
diagnoser: pfitscher_memory
info:
    - average resident: 67.80%
    - average committed: 74.10%
```

No segundo teste foram alocadas aproximadamente 227000 páginas (≈175% do total). Destas, apenas 25% foram utilizadas efetivamente (≈42% do total). Esta carga, novamente acrescida à do sistema operacional, resultou em memória residente aproximada em 53% e reservada superior a 150% em relação a memória total da VM. Desta forma, o diagnóstico esperado é de subprovisionamento, pois apesar da memória residente estar confortável, a reservada é superior a 150% da total. A Figura 5.5 mostra os valores

coletados pela REPD durante o teste. A memória reservada apresentou média de 191.39% e a residente 53.74%. Segundo estes valores a ferramenta diagnosticou o ambiente como subprovisionado em memória (como é apresentado na saída a seguir).

```
$ python3 repd.py 1415276646 1415276946
=== MEMORY ===
diagnostic: UNDERPROVISIONED
diagnoser: pfitscher_memory
info:
    - average resident: 53.74%
    - average committed: 191.39%
```

Figura 5.5: Medições da REPD para o teste de memória subprovisionada

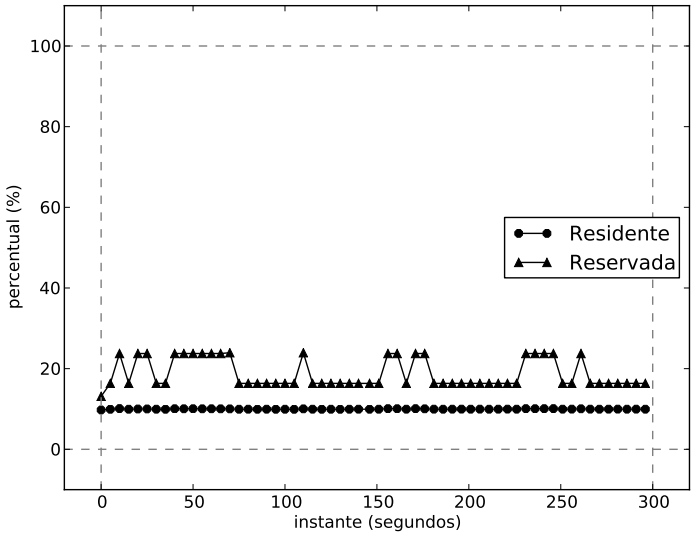


Fonte: Autoria própria

O terceiro teste com a memória consistiu em deixar apenas programas do sistema operacional (com exceção da REPD) executando, e portanto reservando e sujando a memória. Desta forma, o diagnóstico esperado é superprovisionamento, visto que o percentual da memória total requerida para a execução do sistema operacional e seus programas auxiliares é inferior a 50% da memória disponível. A saída abaixo mostra o diagnóstico dado pela REPD de superprovisionamento. A Figura 5.6 apresenta os valores da métrica coletados, que representam uma média de 9.98% para a memória residente e de 18.77% para a memória reservada.

```
$ python3 repd.py 1415116258 1415116558
=== MEMORY ===
diagnostic: OVERPROVISIONED
diagnoser: pfitscher_memory
info:
    - average resident: 9.98%
    - average committed: 18.77%
```

Figura 5.6: Medições da REPD para o teste de memória superprovisionada



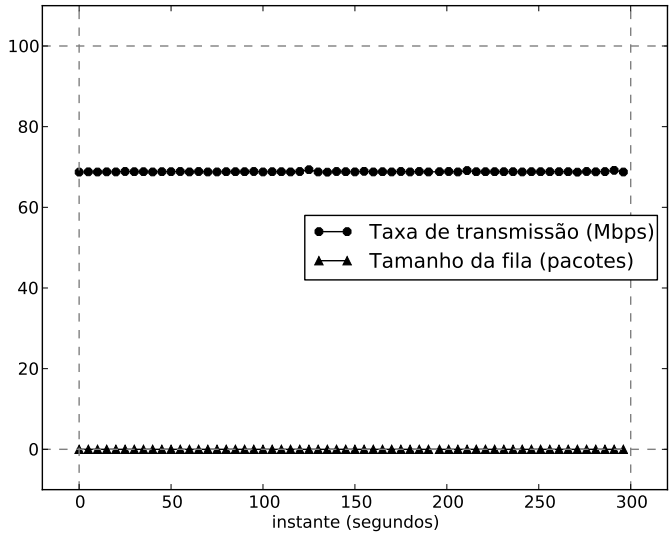
Fonte: Autoria própria

5.1.3 Rede

Os últimos três testes estão relacionados à rede. No primeiro teste foi usada a ferramenta iperf (IPERF, 2014) para gerar tráfego UDP a uma taxa nominal de 70 Mbps da VM para uma outra máquina na rede local. Como esta carga foi praticamente constante (coeficiente de variação pequeno) e inferior à largura de banda (não gera filas na interface de rede) o diagnóstico segundo o modelo é de provisionamento adequado. Os valores apresentados na Figura 5.7 que foram obtidos pela REPD e os da saída a seguir informam coeficiente de variação 0.00 (se arredondado em duas casas decimais) e tamanho médio de fila 0.00. Considerando estes valores consolidados, a ferramenta define o diagnóstico como provisionamento adequado.

```
$ python3 repd.py 1415379548 1415379848
=== NETWORK ===
diagnostic: CORRECT
diagnoser: pfitscher_network
info:
    - average queue size: 0.00 packets
    - CV for transmission rate: 0.00
```

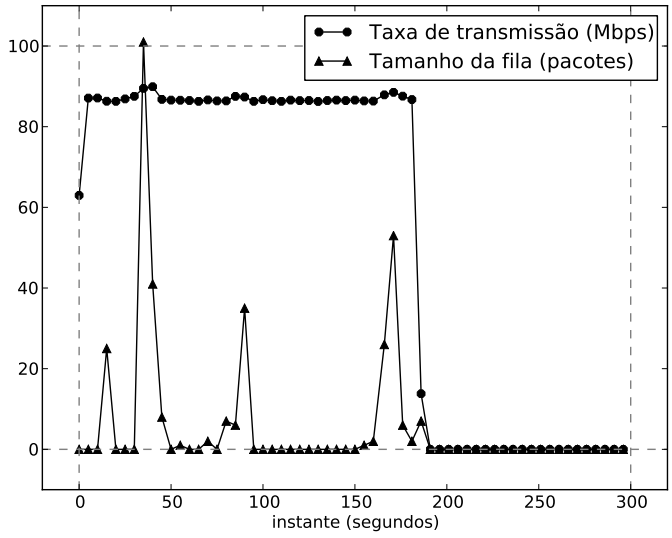
Figura 5.7: Medições da REPD para o teste de rede provisionada adequadamente



Fonte: Autoria própria

No segundo teste sobre a rede, utilizou-se novamente a iperf, porém desta vez foi enviado um arquivo de 2000 MB com conexão TCP. Era esperado que que esta carga gerasse fila na interface de rede, pois, à taxa nominal da rede (100 Mbps ou 12,5 MB/s), o arquivo levaria 160 s para ser transferido, sendo necessário ainda levar em consideração a dinâmica da janela de congestionamento do TCP, que gera rodadas de transmissão a taxas acima da capacidade da rede (JACOBSON, 1988), o que, no Xen, vai levar à formação de fila na interface de rede da MV (BARHAM et al., 2003). Diante disto, o diagnóstico correto é de subprovisionamento. A Figura 5.8 mostra as medições obtidas pela REPD, que possibilitam identificar a formação de fila na interface de rede. O diagnóstico apresentado pela ferramenta foi, conforme saída seguinte, de subprovisionamento.

Figura 5.8: Medições da REPD para o teste de rede subprovisionada



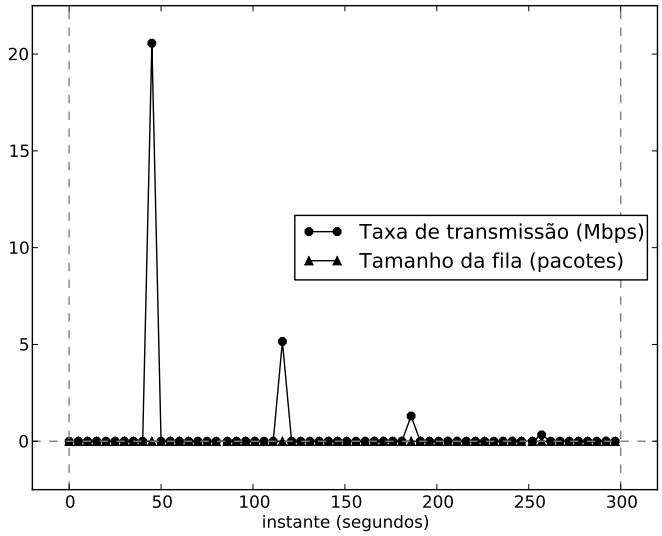
Fonte: Autoria própria

```
$ python3 repd.py 1415797593 1415797893
=== NETWORK ===
diagnostic: UNDERPROVISIONED
diagnoser: pfitscher_network
info:
    - average queue size: 5.38 packets
    - CV for transmission rate: 0.78
```

No último teste para validação das capacidades da ferramenta, foram realizados os envios de quatro arquivos separados por intervalos ociosos de aproximadamente 70 segundos. Os tamanhos dos arquivos enviados foram, em ordem, 12,8 MB, 3,2 MB, 800 KB e 200 KB. A Figura 5.9 mostra os envios dos arquivos como picos na taxa de utilização. O valor próximo a 20 Mbps encontrado pela REPD é justificado pelo fato de a taxa de transmissão ser calculada dividindo o número de bits transmitidos no intervalo de medição pelo tamanho do intervalo. Assim em um intervalo de 5 segundos (utilizado no teste) em que foi enviado apenas um arquivo de 12,8 MB tem-se uma média de $(12{,}8 \times 8) \div 5 = 20{,}48$ Mbps. Os dados coletados pela REPD apresentam uma média de tamanho de fila igual a 0.00 e coeficiente de variação da taxa de transmissão igual a 5.92. O diagnóstico correto é de superprovisionamento (em função do coeficiente de variação ser superior a 5), que foi o diagnóstico apresentado pela ferramenta, vide saída a seguir.

```
$ python3 repd.py 1415796739 1415797039
=== NETWORK ===
diagnostic: OVERPROVISIONED
diagnoser: pfitscher_network
info:
    - average queue size: 0.00 packets
    - CV for transmission rate: 5.92
```

Figura 5.9: Medições da REPD para o teste de rede superprovisionada



Fonte: Autoria própria

5.2 Requisitos não funcionais

Nesta seção são realizadas algumas considerações quanto aos requisitos não funcionais definidos na Seção 4.1 e seus atendimentos pela ferramenta REPD.

5.2.1 Facilidade de implantação

Como discutido anteriormente este requisito tem por finalidade orientar a escolha de ferramentas auxiliares para a REPD. Estas dependências são, segundo a Seção 4.2:

- SQLite: A REPD não necessita especificamente que o SQLite esteja instalado no ambiente onde é executado. Basta que o módulo padrão da linguagem Python de

nome sqlite3 esteja compilado (THE PYTHON SOFTWARE FOUNDATION, 2014). As versões do Python pré-instaladas no sistema operacional normalmente já possuem este módulo compilado e nos casos de instalação por pacotes, o SQLite é listado como dependência, sendo sua instalação coordenada pelo gerenciador de pacotes (SQLITE, 2014).

- Interpretador Python: A ferramenta REPD foi desenvolvida na linguagem de programação Python, sendo necessário um interpretador para executá-la. Como descrito anteriormente um interpretador Python que atenda a este requisito vem pré-instalado em boa parte dos sistemas operacionais (ex. Linux e Mac OSX) e sua implantação em outros (ex. Windows) é simples (LUTZ, 2009). Desta forma, a implantação não é sequer necessária na maioria dos casos;

- mpstat e tc: Estas ferramentas estão disponíveis nos pacotes sysstat e iproute2, respectivamente. Estes pacotes são facilmente instalados na maioria das distribuições Linux, pois fazem parte do repositório canônico de distribuições como Debian, Ubuntu, Fedora, CentOS e OpenSUSE. Vale aqui ressaltar, que estas são dependências em função de como foi implementado o modelo de (PFITSCHER; PILLON; OBELHEIRO, 2014), não sendo assim, intrínsecas da REPD.

5.2.2 Baixo *overhead*

Para verificar o *overhead* gerado pela ferramenta REPD sobre o ambiente foi realizado um teste sobre as mesmas máquinas hospedeira e virtual da Seção 5.1. Neste teste foi executado o fluxo repdd da ferramenta durante uma período de 24 horas e mediu-se o percentual de processador, a quantidade de memória e a largura de banda utilizados. Como era esperado, a largura de banda usada pela REPD foi igual a zero, visto que esta não realiza comunicação com o ambiente exterior à VM. A ferramenta gerou uma carga de processador próxima a 2% realizando medições a intervalos de cinco segundos (pouco mais de um segundo de execução a cada minuto). O uso de memória foi constante no valor de 8492KB. O uso do disco, para armazenamento das métricas coletadas, totalizou em $2,3$ *megabytes* (aproximandamente 840 *megabytes* ao ano). Considerando estes valores o *overhead* pode ser considerado baixo suficiente para não impactar significativamente o diagnóstico na maioria dos casos.

5.2.3 Modularidade

O atendimento a este requisito não funcional pode ser identificado principalmente em dois pacotes da REPD: monitor_manager e diagnoser_manager. Estes foram projetados de forma que novas classes monitoras ou diagnosticadoras possam ser facilmente adicionadas à ferramenta. Foi também descrito, na Seção 4.5, este processo para extensão da REPD, para auxiliar desenvolvedores no entendimento dos passos necessários para a adição de novos modelos de diagnóstico, monitores e métricas à ferramenta.

5.3 Considerações do Capítulo

Neste capítulo foi verificada a conformidade da ferramenta REPD com os requisitos definidos na Seção 4.1. Os requisitos funcionais foram verificados através de testes com aplicação de cargas de trabalho. O atendimento da REPD aos requisitos não funcionais foi realizado através da descrição de características da ferramenta e de seus componentes. No capítulo a seguir são apresentadas as considerações finais do trabalho.

6 Conclusão

A computação em nuvem, um modelo para permitir o acesso ubíquo, conveniente, sob demanda e remoto a recursos computacionais, surgiu como uma opção promissora para reduzir os custos de organizações que podem hospedar suas aplicações na Internet. Nuvens computacionais possibilitam que seus usuários reservem recursos dinamicamente ao longo do tempo segundo a necessidade de suas aplicações, sem necessidade de arcar com custos fixos de infraestruturas computacionais.

Esta dinâmica de reserva introduzida pela computação em nuvem trouxe também alguns desafios. Dentre estes encontra-se o de dimensionar adequadamente os recursos necessários em determinado momento pela aplicação. Para abordar este desafio, e auxiliar os usuários a obter sucesso em solucioná-lo, (PFITSCHER; PILLON; OBELHEIRO, 2014) propôs um modelo para diagnóstico do provisionamento de recursos computacionais voltado ao modelo de serviço Infrastructure-as-a-Service (IaaS). Neste modelo de serviço da computação em nuvem os recursos oferecidos são normalmente capacidade de processamento, memória e largura de banda de rede associados a máquinas virtuais.

O modelo proposto em (PFITSCHER; PILLON; OBELHEIRO, 2014) é de grande valia aos usuários de nuvens IaaS, porém a inexistência de uma ferramenta que o aplicasse impedia que estes últimos usufruíssem de seu diagnóstico. Para mudar este cenário, no presente trabalho foi projetada e desenvolvida uma ferramenta genérica (REPD) que permite a aplicação de modelos de diagnóstico de provisionamento de recursos baseados na análise de séries temporais de métricas de desempenho, e por consequência, do modelo de (PFITSCHER; PILLON; OBELHEIRO, 2014). Após sua implementação foram realizados testes que constataram a capacidade da ferramenta em monitorar os recursos computacionais (processador, memória e rede) de uma máquina virtual e prover o diagnóstico para o provisionamento corretamente, segundo o modelo de diagnóstico instanciado.

Além da capacidade de aplicar o modelo, o projeto e desenvolvimento da REPD foi voltado a sua extensibilidade e portabilidade. Em vista desta primeira, podem ser indicados como possíveis trabalhos futuros o aprimoramento da ferramenta pela adição de modelos de diagnóstico para outros recursos que venham a surgir (como a capacidade

de armazenamento), ou até mesmo a substituição parcial ou total do modelo atual por outros que se mostrem mais eficazes. Cabe aqui enfatizar que, embora a REPD tenha foco nos usuários de nuvem, ele também pode ser útil a pesquisadores, pois facilita a avaliação experimental de modelos alternativos de diagnóstico de provisionamento. Ainda outro trabalho relacionado a extensibilidade está na substituição do módulo do banco de dados por outra solução que realize a consolidação ou compactação de dados, de forma a diminuir significativamente o espaço em disoc necessário pela REPD a longo prazo.

Considerando a portabilidade, a ferramenta pode ser modificada para uso em outros sistemas operacionais não suportados até o momento (ex. Windows) que tem sido oferecido por provedores de nuvens IaaS (ARMBRUST et al., 2010; ZHANG; CHENG; BOUTABA, 2010). Por fim, considerando a utilização da REPD por um mesmo usuário em diversas VMs, outro trabalho futuro de grande utilidade seria a implementação de uma versão distribuída da ferramenta, facilitando a monitoração de várias VMs através de uma interface única de gerencia para todas.

Referências

ACETO, G. et al. Survey cloud monitoring: A survey. *Comput. Netw.*, Elsevier North-Holland, Inc., New York, NY, USA, v. 57, n. 9, p. 2093–2115, jun. 2013. ISSN 1389-1286. Disponível em: <http://dx.doi.org/10.1016/j.comnet.2013.04.001>.

ARMBRUST, M. et al. A view of cloud computing. *Commun. ACM*, ACM, New York, NY, USA, v. 53, n. 4, p. 50–58, abr. 2010. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/1721654.1721672>.

BARHAM, P. et al. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, ACM, v. 37, n. 5, p. 164–177, 2003.

BUEVICH, M. et al. Respawn: A distributed multi-resolution time-series datastore. In: IEEE. *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*. [S.l.], 2013. p. 288–297.

CHERKASOVA, L.; GUPTA, D.; VAHDAT, A. Comparison of the three cpu schedulers in xen. *SIGMETRICS Perform. Eval. Rev.*, ACM, New York, NY, USA, v. 35, n. 2, p. 42–51, set. 2007. ISSN 0163-5999. Disponível em: <http://doi.acm.org/10.1145/1330555.1330556>.

CHERUBINI, L. R. *Avaliação De Modelo De Diagnóstico De Provisionamento De Memória Em Ambientes Windows*. Monografia (Trabalho de Conclusão de Curso) — Centro Universitário Tupy – UNISOCIESC, Joinville, 2013.

CILIENDO, E.; KUNIMASA, T. *Linux Performance Monitoring and Tuning*. IBM Redbooks, 2007. (IBM redbooks). Disponível em: <http://www.redbooks.ibm.com/abstracts/redp4285.html>.

DERI, L.; MAINARDI, S.; FUSCO, F. Tsdb: A compressed database for time series. In: *Proceedings of the 4th International Conference on Traffic Monitoring and Analysis*. Berlin, Heidelberg: Springer-Verlag, 2012. (TMA'12), p. 143–156. ISBN 978-3-642-28533-2. Disponível em: <http://dx.doi.org/10.1007/978-3-642-28534-9_16>.

DU, J.; SEHRAWAT, N.; ZWAENEPOEL, W. Performance profiling of virtual machines. *SIGPLAN Not.*, ACM, New York, NY, USA, v. 46, n. 7, p. 3–14, mar. 2011. ISSN 0362-1340. Disponível em: <http://doi.acm.org/10.1145/2007477.1952686>.

HOCH, D. *Linux System and Performance Monitoring*. Redwood City: StrongMail Systems, 2010.

HOEFER, C. N.; KARAGIANNIS, G. Taxonomy of cloud computing services. In: *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*. [S.l.: s.n.], 2010. p. 1345–1350.

IPERF: Site. 2014. Disponível em: <https://iperf.fr>. Acesso em: 14 nov. 2014.

JACOBSON, V. Congestion avoidance and control. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 18, n. 4, p. 314–329, ago. 1988. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/52325.52356>.

LAGAR-CAVILLA, H. A. et al. Snowflock: Rapid virtual machine cloning for cloud computing. In: *Proceedings of the 4th ACM European Conference on Computer Systems*. New York, NY, USA: ACM, 2009. (EuroSys '09), p. 1–12. ISBN 978-1-60558-482-9. Disponível em: <http://doi.acm.org/10.1145/1519065.1519067>.

LUTZ, M. *Learning Python: Powerful Object-Oriented Programming*. O'Reilly Media, 2009. (Animal Guide). ISBN 9781449379322. Disponível em: <http://books.google.com.br/books?id=1HxWGezDZcgC>.

MATTHEWS, J. *Running Xen: A Hands-on Guide to the Art of Virtualization*. PRENTICE HALL COMPUTER, 2008. ISBN 9780132349666. Disponível em: <http://books.google.com.br/books?id=ffEnmAEACAAJ>.

MELL, P. M.; GRANCE, T. *SP 800-145. The NIST Definition of Cloud Computing*. Gaithersburg, MD, United States, 2011.

MENASCE, D. A.; ALMEIDA, V. *Capacity Planning for Web Services: Metrics, Models, and Methods*. 1st. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001. ISBN 0130659037.

OPENTSDB: Site. 2014. Disponível em: <http://opentsdb.net/>. Acesso em: 10 jun. 2014.

PFITSCHER, R. J.; PILLON, M. A.; OBELHEIRO, R. R. *Diagnóstico do Provisionamento de Recursos para Máquinas Virtuais em Nuvens IaaS*. Dissertação (Mestrado) — Universidade do Estado de Santa Catarina, Joinville, 2014.

RRDTOOL: Round robin database tool. site. 2014. Disponível em: <http://oss.oetiker.ch/rrdtool/>. Acesso em: 10 jun. 2014.

SAUVÉ, J. et al. Sla design from a business perspective. In: *In Proceedings of DSOM 2005*. [S.l.]: Springer, 2005.

SOMMERVILLE, I. *Software Engineering*. Pearson/Addison-Wesley, 2011. (International Computer Science Series). ISBN 9780137053469. Disponível em: <http://books.google.com.br/books?id=l0egcQAACAAJ>.

SQLITE: Site. 2014. Disponível em: <http://sqlite.org/>. Acesso em: 08 nov. 2014.

SULEIMAN, B. et al. On understanding the economics and elasticity challenges of deploying business applications on public cloud infrastructure. *Journal of Internet Services and Applications*, Springer, v. 3, n. 2, p. 173–193, 2012.

TAKEMURA, C.; CRAWFORD, L. *The Book of Xen: A Practical Guide for the System Administrator*. No Starch Press, 2009. (No Starch Press Series). ISBN 9781593271862. Disponível em: <http://books.google.com.br/books?id=vYIpAQAAMAAJ>.

TANENBAUM, A. *Modern Operating Systems*. Pearson Prentice Hall, 2008. (GOAL Series). ISBN 9780136006633. Disponível em: <http://books.google.com.br/books?id=DxNiQgAACAAJ>.

THE LINUX MAN-PAGES PROJECT. *free(1) man-page*. [S.l.], 2014. Disponível em: <http://man7.org/linux/man-pages/man1/free.1.html>.

THE LINUX MAN-PAGES PROJECT. *proc(5) man-page*. [S.l.], 2014. Disponível em: <http://man7.org/linux/man-pages/man5/proc.5.html>.

THE LINUX MAN-PAGES PROJECT. *tc(8) man-page*. [S.l.], 2014. Disponível em: <http://man7.org/linux/man-pages/man8/tc.8.html>.

THE PYTHON SOFTWARE FOUNDATION. *sqlite3 — DB-API 2.0 interface for SQLite databases*. [S.l.], 2014. Disponível em: <https://docs.python.org/3.4/library/sqlite3.html>.

VAN, H. N.; TRAN, F.; MENAUD, J.-M. Sla-aware virtual resource management for cloud infrastructures. In: *Computer and Information Technology, 2009. CIT '09. Ninth IEEE International Conference on*. [S.l.: s.n.], 2009. v. 1, p. 357–362.

WALDSPURGER, C. A. Memory resource management in vmware esx server. *SIGOPS Oper. Syst. Rev.*, ACM, New York, NY, USA, v. 36, n. SI, p. 181–194, dez. 2002. ISSN 0163-5980. Disponível em: <http://doi.acm.org/10.1145/844128.844146>.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, Springer-Verlag, v. 1, n. 1, p. 7–18, 2010. ISSN 1867-4828. Disponível em: <http://dx.doi.org/10.1007/s13174-010-0007-6>.