Universidade Federal do Paraná

Diego Henrique Pagani

Um modelo de provisionamento elástico de recursos baseado em níveis de estresse

Curitiba PR
2016

Diego Henrique Pagani

Um modelo de provisionamento elástico de recursos baseado em níveis de estresse

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Prof. Dr. Luis Carlos Erpen de Bona.

Curitiba PR
2016

P129m Pagani, Diego Henrique
Um modelo de provisionamento elástico de recursos baseado em níveis de estresse / Diego Henrique Pagani. – Curitiba, 2016.
48 f. : il. color. ; 30 cm.

Dissertação - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática, 2016.

Orientador: Luis Carlos Erpen de Bona .
Bibliografia: p. 44-48.

1. Banco de dados - Gerência. 2. Gerenciamento de memória (Computação). 3. Redes de computadores – Gerência. 4. Computação em nuvem. I. Universidade Federal do Paraná. II.Bona, Luis Carlos Erpen de. III. Título.

CDD: 005.7406

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
Setor CIÊNCIAS EXATAS
Programa de Pós Graduação em INFORMÁTICA
Código CAPES: 40001016034P5

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **DIEGO HENRIQUE PAGANI**, intitulada: "**Um modelo de provisionamento elásticos de recursos baseado em níveis de estresse.**", após terem inquirido o aluno e realizado a avaliação do trabalho, são de parecer pela sua APROVAÇÃO.

Curitiba, 08 de Julho de 2016.

Prof LUIS CARLOS ERPEN DE BONA
Presidente da Banca Examinadora (UFPR)

Prof EDUARDO CUNHA DE ALMEIDA
Avaliador Interno (UFPR)

Prof GUILHERME GALANTE
Avaliador Externo (UNIOESTE)

MINISTÉRIO DA EDUCAÇÃO - UNIVERSIDADE FEDERAL DO PARANÁ - Setor de Ciências Exatas - Coordenação de Pós-Graduação em Informática

Aos meus pais, irmão e familiares.

Agradecimentos

Após dois anos, com mais experiência e conhecimento acadêmico e de vida, gostaria de agradecer a todos aqueles que se mostraram presentes e colaboraram com o desenvolvimento deste trabalho de alguma forma.

Primeiramente a Deus, pela força e por ter permitido que seja possível aprender, me conduzido e sustentado em todos os momentos de dificuldades durante esta jornada. Aos meus pais, por me darem todo apoio e suporte durante toda a vida. A minha família, gostaria de agradecer pelo incentivo, compreensão e carinho durante todo o período que estive em Curitiba.

Aos meus amigos de Cascavel, que mantiveram presentes em minha vida e muitas vezes mudando suas agendas para que nos reuníssemos. Aos novos amigos do Departamento de Informática, pelos diversos momentos divertidos e de aprendizado, pelos inúmeros cafés na cozinha e almoços no RU. A quem esteve presente em grande parte desta jornada e demonstrou apoio, compreensão e companheirismo.

Ao Prof. Luis Carlos Erpen De Bona, por ter me aceito no Programa de Pós-Graduação, pela grande ajuda e confiança durante todo o desenvolvimento deste trabalho, o qual não seria possível. Aos professores do Departamento de Informática, que repassaram seu conhecimento de forma singular, garantindo um ótimo aprendizado.

Muito obrigado!

Resumo

A computação em nuvem pode ser definida como um modelo de compartilhamento de recursos computacionais,que podem ser adicionados e removidos de forma dinâmica e elástica possibilitando ajustar os recursos alocados de forma eficiente, reduzindo os custos operacionais e garantindo o mesmo desempenho.Os sistemas de gerenciamento de bancos de dados (SGBD) são responsáveis por armazenar, organizar, gerenciar e extrair informações, e ao longo do tempo surgiram diferentes abordagens para efetuar estas tarefas. Com as diversas implementações e abordagens, foram propostos vários *benchmarks*, para expor e avaliar as diferentes características entre as implementações, sendo possível comparar estas aplicações e definir objetivos. Um tipo de *benchmark* é o teste de estresse, que tem como objetivo descobrir falhas quando o SGBD está sobre uma alta carga de trabalho. Este tipo de teste, inicia-se quando o SGBD é instanciado, sem carga, até o ponto em que deixa de responder devido a alta demanda. Neste contexto, propõe-se classificar o desempenho de um SGBD baseado em diferentes níveis de estresse: Aquecimento, Estável, Sob-Pressão, Estresse e Falha iminente. Visando manter o SGBD sempre estável e com a velocidade controlada, alocando para o serviço apenas os recursos necessários, neste trabalho propomos um controlador elástico reativo e automático, utilizando os níveis de estresse para determinar um momento adequado para reajuste dos recursos. Os SGBD relacionais apresentam diversas técnicas que podem ser usadas para elasticidade, sendo as principais delas de particionamento, migração e replicação. Neste trabalho utilizamos a elasticidade vertical de CPU, aumentando ou diminuindo o total de núcleos disponíveis para uma máquina virtual dedicada a esta aplicação. O controlador proposto apresenta vantagens e desvantagens para diferentes cargas de trabalho, comparadas com outras duas técnicas utilizadas para provisionar recursos de forma vertical: baseada em tempo de resposta e uso de recursos.

Palavras-chave: elasticidade, nuvem computacional, banco de dados, níveis de estresse.

Abstract

Cloud computing can be defined as a computational resource sharing model, which can be added and removed dynamically and elastically allowing adjust the allocated resources efficiently, reducing operating costs and ensuring the same performance. Database management systems (DBMS) are responsible for store, organize, manage and extract information, and over time have arisen different approaches to perform these tasks. With the various approaches have been proposed several benchmarks to expose and evaluate the different characteristics between implementations, and can compare these applications and set goals. A kind of benchmark is the stress test, which aims to find fault when the DBMS is on a high workload. This type of testing begins when the DBMS is instantiated, without charge, to the point where it stops responding due to high demand. In this context, it is proposed to classify the performance of a DBMS based on different stress levels: Warmup, Stable, Under-Pressure, Stress and Thrashing state. To maintain the DBMS always stable and controlled speed, allocating to service only the necessary resources, in this paper we propose a reactive and automatic elastic controller using stress levels to determine an appropriate time for adjustment of resources. Relational DBMS have several techniques that can be used to elasticity, the main of them partitioning, migration and replication. In this work we used the vertical elasticity of CPU, increasing or decreasing the total number of available cores to a virtual machine dedicated to this application. The proposed controller has advantages and disadvantages for different workloads, compared with two other techniques used to provision resources vertically: based on response time and resource usage.

Keywords: elasticity, cloud computing , database management system, stress levels.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A computação em nuvem é um paradigma que pode ser definida como um conjunto de recursos computacionais compartilhados que podem ser acessados e utilizados [3]. Uma nuvem computacional (NC) é um modelo de compartilhamento de recursos computacionais que possibilita adicionar recursos de forma dinâmica e elástica, ganhando maior agilidade e flexibilidade na oferta de serviços. A elasticidade é a capacidade de um sistema variar os recursos disponíveis sem a interrupção do serviços, possibilitando assim manter recursos computacionais alocados apenas quando necessário. Devido a isto, as empresas e instituições de ensino começaram a utilizar este modelo, como forma de economizar recursos.

As NCs podem ser classificada entre vários aspectos, como modelos de implantação e de serviço. Os modelos de implantação são Privada, Pública, Híbrida ou Comunitária e referem-se as permissões de acesso dos clientes à nuvem. Os modelos de serviço variam conforme o nível de abstração para o cliente, sendo fornecido o acesso a Infraestrutura, a Plataforma ou ao *Software* e cada um destes modelos apresenta diferentes características e configurações para o cliente.

A elasticidade pode ser classificadas em quatro aspectos [4]: Escopo, Política, Objetivo e Método. O escopo define em qual modelo de serviço a elasticidade é aplicada. A política define o controlador elástico, alterando entre automática ou manual. O objetivo da elasticidade pode ser em otimizar o uso dos recursos, utilizando apenas o necessário e consequentemente reduzindo custos (como em modelos *pay-per-use*) ou para economizar energia. O método define a implementação da elasticidade: por replicação, migração ou redimensionamento de recursos.

Há vários trabalhos envolvendo diferentes técnicas de cálculo e escalonamento dos recursos, como orientado a tarefas, tempo de resposta ou uso de recursos. Mao [5] utiliza a orientação por tarefas e, determinado um tempo limite de execução para cada trabalho, aloca o mínimo de recursos necessários para que todos os trabalhos sejam efetuados dentro deste tempo. Han [6] e Dutreilh [7] decidiam pelo aumento dos recursos baseado no tempo de resposta das aplicações, este aumentava ou diminuía o número de máquinas virtuais e aquele alterava primeiramente a quantidade de núcleos de processamento e memória e apenas quando necessário alocava novos nós.

Devido a massificação da *World Wide Web* (WWW) na década de 90 e nos tempos atuais pela grande oferta de aplicativos em *smartphones*, informatização de serviços como a bilhetagem eletrônica e sistemas de pagamento por cartão de crédito ou débito, o uso dos Sistemas Gerenciadores de banco de dados (SGBD) para armazenar e organizar dados, tornou-se mais intenso [8]. Este aumento na demanda por SGBDs que ofereçam velocidade, robustez e segurança causou a criação de abordagens diferentes em implementa-los, mudando na forma de armazenamento dos dados, métodos de recuperação e a forma de implantação.

Ao se implantar SGBDs em NCs, cria-se alguns problemas como em determinar a quantidade de recursos necessários para o funcionamento dentro da especificação prévia e o instante de tempo ideal de alterar estes recursos. Quando os recursos necessitam ser alterados, existem três diferentes maneiras de provisionar estes recursos a um serviço em NCs: alterando número de nós de processamento (elasticidade horizontal); alterando recursos disponíveis em um único nó (elasticidade vertical); e abordagens híbridas, que envolve a combinação destes métodos.

Em geral, a elasticidade aplicada em SGBDs é implementada por meio de técnicas de particionamento, migração e replicação, com balanceamento de carga [9]. O particionamento é utilizado para escalar um banco de dados para múltiplos nós quando a carga excede a capacidade de um único servidor, dividindo os dados em partições entre os servidores. Cada partição geralmente é alocada em um servidor, como implementado em diversos SGBDs, como VoltDB [10], NuoDB[1] e ElasTras [11].

A migração consiste em mudar a aplicação para uma máquina com maior capacidade e se utilizando de máquinas virtuais, pode ser realizada utilizando técnicas de *stop-and-copy* [12] ou *live migration* [13]. As técnicas de particionamento e migração podem ainda ser utilizadas em conjunto, migrando as partições de dados para uma máquina recém instanciada. Os trabalhos de Elmore et al. [12] e Das et al.[14], empregam técnicas de migração. Curino et al.[15] e Serafini et al.[16] apresentam uma solução que combina particionamento e migração.

Técnicas de replicação são geralmente empregadas em conjunto com balanceamento de carga. Um mecanismo clona a máquina virtual que hospeda o servidor de banco de dados existente que é então iniciado em um novo servidor físico, resultando em uma nova réplica que começa a responder às requisições. Na prática, a nova instância pode não responder instantaneamente a estas requisições, pois é necessário que os dados estejam sincronizados para que as propriedades ACID (atomicidade, consistência, integridade e durabilidade) não sejam violadas, devendo-se considerar ainda a latência necessária de cópia entre as máquinas físicas e a configuração do balanceador de carga para que esta nova instância faça parte do serviço. CloudBD AutoAdmin [13] é um exemplo de solução que combina técnicas de particionamento e replicação de banco de dados.

Embora a migração possa ver vista como uma forma implementar elasticidade vertical [17], não foi encontrado em nossa revisão bibliográfica soluções que explicitamente utilizem a reconfiguração de recursos, outra técnica de elasticidade vertical, na exploração em banco de dados.

Meira [2] propõe representar um SGBD em níveis de desempenho, desde a inicialização da aplicação até a interrupção devido a excesso de carga de trabalho aplicada. Cada nível é representado por um dos cinco estados que máquina DSM (*Database State Machine*), sendo eles: Aquecimento, Estável, Sobre pressão, Estresse e Falha iminente. Ele define os estados utilizando métricas de desempenho, como a variação e vazão ao longo do tempo e prevê quando o SGBD irá deixar de responder.

A representação proposta por Meira [2] pode ser expandida para outros tipos de aplicação e aliada à possibilidade de controlar os recursos computacionais de forma dinâmica e elástica, torna-se interessante avaliar estas estratégias buscando economia de recursos computacionais, mas mantendo o desempenho da aplicação. Desta forma, este trabalho propõe um modelo de provisionamento elástico de recursos computacionais, capaz de inferir um momento pertinente para alterações destes recursos, baseado nos níveis de estresse. Ele foi projetado para operar com qualquer aplicação cliente-servidor e é dividido em quatro módulos, responsáveis por: (a) inferir o estado; (b) determinar o momento das ações de elasticidade; (c) monitorar os recursos;

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

(d) ordenar o controlador da nuvem. Como aplicação de teste, utilizaremos o SGBD relacional PostgreSQL para avaliação do seu comportamento em ambiente elástico, variando o número de núcleos de CPU (elasticidade vertical), dedicados a uma MV na qual será implantada esta aplicação.

Para validar esta proposta, foi efetuado um experimento inicial avaliando o tempo de resposta do banco de dados, em função do número de vCPUs alocadas para a máquina virtual que hospeda o SGBD. Para gerar a carga de trabalho, executou-se um conjunto de consultas do *benchmark* TPC-H em uma instância de banco de dados PostgreSQL 9.5 incrementando o número de clientes conectados até 800 clientes. Ao todo, foram executadas 5 testes, utilizando 1, 2, 4, 8 e 16 vCPUs. O tempo de resposta é o obtido por uma consulta aos dados cadastrados para a execução do TPC-H.

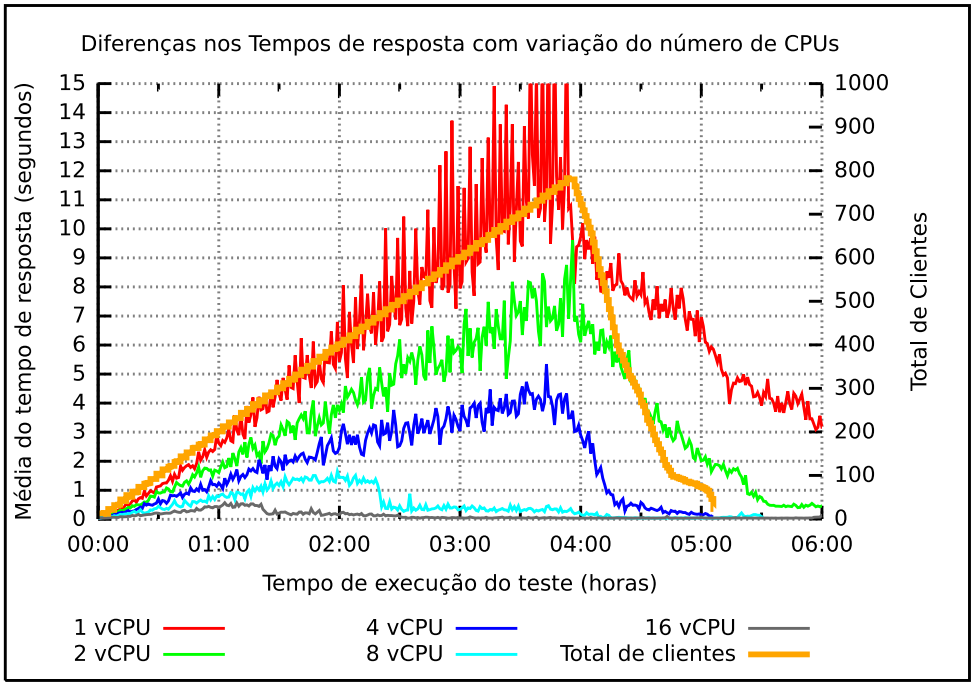


Figura 1.1: Teste com carga crescente utilizando diferentes quantidades de vCPUs.

Como pode ser observado, o número de vCPUs alocadas impactou diretamente no tempo de resposta do SGBD, sendo uma abordagem interessante, visto a diferença no desempenho do número vCPUs para esta aplicação com diferentes cargas de trabalho. Assim, dado uma determinada carga de trabalho (número de clientes) e uma expectativa de tempo de resposta, pode-se variar o número de processadores alocados em tempo de execução para alcançar o resultado desejado. A infraestrutura deve ser alocada de modo apropriado, para que não ocorra a falta de recursos para a aplicação e evite que recursos desnecessários fiquem alocados.

Utilizando estes princípios, foram executados três baterias de testes com diferentes números de clientes, com três técnicas de provisionamento, mantendo sempre a mesma carga de trabalho. As técnicas utilizadas são baseadas no tempo de resposta, na classificação em níveis de estresse e por consumo dos recursos computacionais, sendo o uso de CPU global como métrica. Foram redimensionados o número de vCPUs alocados em tempo de execução, para entendimento do comportamento do SGBD e avaliação destas três técnicas de provisionamento.

Os métodos baseados em tempo de resposta e níveis de estresse obtiveram resultados semelhantes na maioria das situações, em que o número de vCPUs alocados foram coerentes

com a demanda exigida, tendo uma variação pequena entre os métodos, com o menor índice para o método de níveis de estresse. Com o método de provisionamento baseado no consumo de recursos, o tempo de resposta da aplicação obteve os valores mais baixos, mas isso foi devido a estratégia de utilizar praticamente todos os núcleos disponíveis na maior parte do tempo. A aplicabilidade dos métodos de provisionamento, dependem muito do tipo de serviço e carga de trabalho submetida, sendo necessário uma análise da carga de trabalho.

O restante deste trabalho esta dividido da seguinte forma: No Capítulo 2, será comentado sobre a elasticidade em nuvens computacionais, abordando as definições e os métodos existentes. No Capítulo 3, abordaremos alguns conceitos de banco de dados e sobre avaliações no seu desempenho. No Capítulo 4, será discutido o MoBINE, um provisionador de recursos para aplicações cliente-servidor, que baseia-se na classificação em níveis de estresse. No Capítulo 5 será apresentado a metodologia dos testes e os resultados da aplicação do modelo e, por fim, no Capítulo 6, será apresentado as considerações finais deste trabalho e trabalhos futuros.

Capítulo 2

Elasticidade em Nuvens Computacionais

O termo "Computação em Nuvem" é a tradução de *Cloud Computing* mencionado pela primeira vez em 2006 pelo CEO da Google Eric Schmidt, ao explicar o modelo de negócios que a empresa seguia [18]. A partir disso, várias empresas começaram a adotar modelos semelhantes, tal como Amazon e o seu serviço *Amazon Elastic Compute Cloud (Amazon EC2)* [19].

Existem várias definições na literatura sobre computação em nuvem, porém, nenhuma é considerada definitiva, comprovando que ainda é um conceito em desenvolvimento [20]. Vaquero [3] em seu trabalho definiu a computação em nuvem como um grande conjunto de recursos virtualizados e compartilhados que podem ser dinamicamente reconfigurados, que permite o uso otimizado dos recursos. Com contratos de serviço personalizados (*Service Level Agreement - SLA*), este conjunto de recursos é explorado geralmente por modelos de pagamento por utilização (*pay-per-use*), em que nos contratos são definidos as garantias que o provedor da infraestrutura deve oferecer.

Esta reconfiguração de recursos é chamada de elasticidade e pode ser definida, em NCs, como a capacidade de um sistema de se adaptar a mudanças de carga, adicionando ou removendo recursos de forma autônoma [21]. Para isso, é necessário que tanto a arquitetura, quanto a aplicação suportem esta característica de alguma forma.

Tendo em vista seu amplo conceito e a quantidade de informações que englobam este assunto, este capítulo está organizado da seguinte maneira: A Seção 2.1 abordará os conceitos de nuvem computacional e suas características. Na Seção 2.2 veremos as definições de elasticidade e as estratégias utilizadas para os momentos ideais para variar os recursos.

2.1 Nuvem computacional

Uma nuvem computacional (NC) é um modelo de compartilhamento de recursos computacionais virtualizados e compartilhados que podem ser dinamicamente reconfigurados[3]. Uma NC pode ser definida por cinco características essenciais, quatro modelos de implantação e três modelos de serviço.

As características essenciais de uma NC são: (i) Acesso via rede pelos mecanismos convencionais, para acesso dos variados tipos de dispositivos; (ii) Cliente pode disponibilizar uma aplicação sem precisar consultar o provedor; (iii) Recursos do provedor são reservados para atender múltiplos consumidores usando um modelo *multi-tenant*[1] e os recursos virtuais não são atrelados a um recurso físico, sendo possível a alteração de acordo com as demandas de

[1]Modelo que descreve uma aplicação que é compartilhada entre vários clientes, mas cada cliente administra o serviço de forma independente.

cada cliente; (iv) Recursos devem ser atribuídos de forma elástica para serem compatíveis com a demanda; (v) NCs devem controlar e otimizar os RCs, baseados em métricas específicas de cada RC.

Para atender a estas características, uma das soluções possíveis é utilizar máquinas virtuais, pela agilidade e flexibilidade de abstrair a camada de recursos, possibilitando a mudança de *hardware* sem precisar realizar a reconfiguração do *software*. Uma NC determina como os recursos virtuais são gerenciados e alocados e responsável por alterar os RCs computacionais alocados, determinando a quantidade e o momento que ações de elasticidade devem ser tomadas, e quando utilizada de virtualização, possibilita o escalonamento rápido de recursos [22].

Existem quatro formas de implantação de uma NC, baseadas em permissões de acesso dos clientes. São elas [23]: (i) Pública: Acesso em escala massiva, podendo ou não cobrar pelo uso, mas usuários são considerados não-confiáveis; (ii) Privada: O uso da nuvem é exclusivo por uma organização, oferecendo serviços dentro da própria rede corporativa para usuários confiáveis; (iii) Privada-virtual: Uma nuvem privada atuando sobre uma nuvem pública, mas o armazenamento dos dados é feito em servidores dedicados não compartilhados; (iv) Comunitária: O uso da nuvem é feito apenas por um grupo específico de clientes, com interesses em comum. A nuvem é gerenciada por um deles ou por terceiros; (v) Híbrida: É a composição de dois ou mais tipos de modelos, mas são unidas de forma que permita implantação de múltiplos modelos. Permite recursos serem alternados entre nuvens públicas e privadas.

A escolha de um modelo de implantação varia dependendo das característica de uso de cada organização, relacionando o custo, segurança e controle das informações. Nas nuvens privadas, a organização tem um maior controle dos recursos e das informações, mas o custo de aquisição e configuração é elevado. As nuvens públicas, oferecem recursos sob demanda, mas com pouco controle do local de execução. Entre estas duas configurações, as nuvens híbridas podem trazer benefícios dos dois modelos ou as nuvens comunitárias voltadas para projetos colaborativos [20].

Os modelos de serviço [23] são divididos em três categorias: *Software*, Plataforma e Infraestrutura. A Figura 2.1 apresenta de forma hierárquica estes três modelos, podendo que o serviço do topo tem a maior abstração da nuvem para o usuário e o da base com a maior liberdade de configuração.

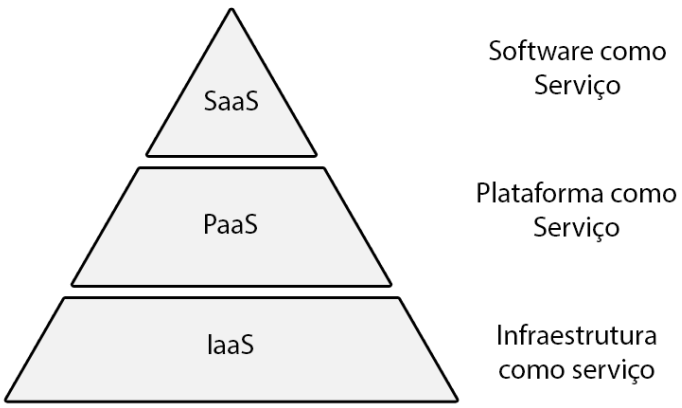


Figura 2.1: Pirâmide dos modelos de serviço, sendo SaaS o modelo com menor complexidade de configuração e IaaS com a maior liberdade de serviços.

Estes modelos de serviço devem ser escolhidos dependendo das necessidades dos clientes, sendo configurações específicas de baixo nível ou apenas uma interface simples de gerenciamento do sistema. Os provedores podem oferecer estes três modelos para clientes da seguinte forma:

- Infraestrutura como Serviço (IaaS): O provedor fornece ao cliente processamento, armazenamento, rede e outras infraestruturas fundamentais para que ele possa implantar seus próprios *softwares* ou configurações específicas. Um exemplo de serviço de IaaS, é o Amazon EC2 [19];
- Plataforma como serviço (PaaS): Permite aos clientes implantarem *softwares*, próprios ou adquiridos, usando linguagens de programação, bibliotecas ou serviços suportados pelo provedor ou pela máquina hospedeira. Os usuários deste modelo não controlam ou gerenciam a nuvem, apenas as aplicações instaladas por estes. Um exemplo deste serviço é o Microsoft Windows Azure [24]. Nesta categoria se encaixam serviço de *DBaaS - Database-as-a-Service*;
- Software como Serviço (SaaS): O cliente contrata aplicações que operam sobre a infraestrutura de nuvem, mas não há qualquer contato com as camadas inferiores, salvo alguma opção específica do *software*. Alguns exemplos são os Google Apps [25], como GMail e o Dropbox [26].

O cliente que utiliza destes serviços, necessita que os *softwares* e o sistema operacional (se for ambiente virtualizado) sejam capazes de identificar e se adaptar aos RCs disponíveis em tempo de execução. Com estas características apresentadas, na Seção 2.2 veremos os principais conceitos sobre elasticidade e suas aplicações, voltadas a computação em nuvem.

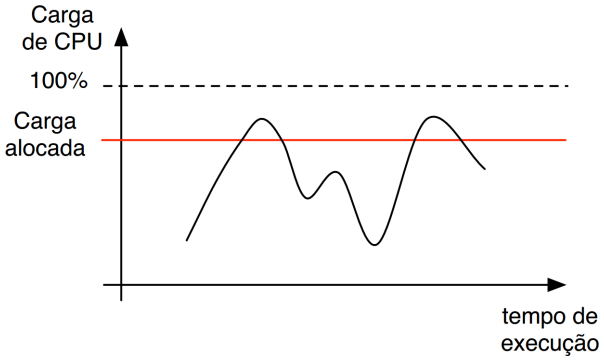
2.2 Elasticidade em nuvens computacionais

Nos últimos anos a adoção de Nuvens Computacionais em empresas e instituições de pesquisa cresceu devido à possibilidade de aquisição de RC de forma dinâmica. No ambiente destas NCs, a elasticidade é uma característica fundamental [27], pois os recursos podem ser gerenciados conforme a demanda que a aplicação esta recebendo.

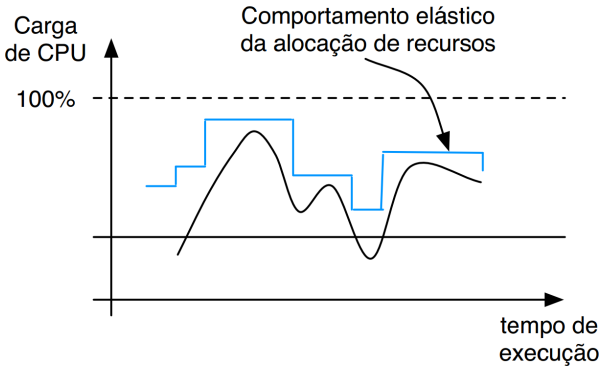
A elasticidade é a capacidade de um sistema adaptar-se a mudanças de carga, adicionando ou removendo recursos. O objetivo é que a cada instante de tempo os recursos utilizados sejam coerentes com a demanda [21], evitando que recursos ociosos continuem alocados. Segundo Galante [20], estes recursos podem ser desde componentes de *hardware* como CPUs (*Central Processor Unit*), memória e armazenamento, ou até máquinas virtuais (MV) completas. Segundo o mesmo autor, o conceito de elasticidade também pode ser estendido a aplicações, em que uma aplicação elástica é capaz de se adaptar as alterações na quantidade de recursos ou de solicitar/liberar recursos de acordo com sua necessidade.

A Figura 2.2, de Righi [1], mostra dois gráficos de carga por tempo, mostrando as diferenças entre um sistema elástico e outro sem esta característica. A Figura 2.2(a), mostra uma MV com recursos fixos, em certos momentos a carga do sistema ultrapassou os limiares definidos, podendo ocorrer uma queda no desempenho. A Figura 2.2(b) mostra o funcionamento elástico de uma MV, em que os recursos alocados são coerentes com a demanda.

Um sistema escalável e elástico é capaz de se adaptar, em tempo de execução, a uma nova configuração de recursos, mantendo o desempenho, pois apenas será alocado os recursos necessários para a carga de trabalho em execução, sem afetar a disponibilidade e o desempenho



(a) Operação sem elasticidade.



(b) Operação com elasticidade.

Figura 2.2: Comparação de um sistema elástico com um não-elástico. Retirada de Righi [1].

dos serviços. Armbrust [28] diz que os *data centers* usam, em média, 5% a 20% do processamento, podendo aumentar de 2 a 10 vezes em horários de pico, o que indica que há instantes de tempo que existem recursos alocados sem necessidade.

A Figura 2.3 mostra um gráfico da quantidade de recursos ao longo da execução de uma aplicação. Podemos notar que há períodos em que há mais recursos que o necessário (indicado pelas áreas amarelas) e períodos em que há recursos insuficientes para a demanda (indicados pelas áreas vermelhas). Estas áreas não existiriam, caso um provisionador de recursos ideal fosse utilizado, pois sempre os recursos seriam alocados ou removidos em um momento adequado.

Segundo Galante e Bona [4], a elasticidade possui quatro classificações: (i) Escopo; (ii) Política; (iii) Método; (iv) Objetivo.

O escopo define em que local a elasticidade é aplicada, na infraestrutura, plataforma ou aplicação. A infraestrutura contém um controlador de elasticidade, que disponibiliza ao usuário métodos de controlar os recursos. Nas plataformas e aplicações, o controle da elasticidade está anexado ao seu código-fonte, de forma que a aplicação ou o ambiente entre em contato com a nuvem para determinar os RCs a serem alterados [29].

A segunda classificação, política, é a forma em que a elasticidade será aplicada e se divide em duas formas: Manual e Automática. Na forma manual, o cliente é responsável por monitorar seu ambiente e realizar as ações de elasticidade, o provedor da nuvem apenas deve fornecer uma interface para que o cliente execute a ação. Na automática, o cliente determina algumas configurações e a tomada de decisão e a ação elástica ocorre de forma automática.

Existe duas formas principais de projetar a tomada de decisão automática: preditivo e reativos [30]. A reativa é uma abordagem simplista, pois é necessário monitorar os RCs e baseado

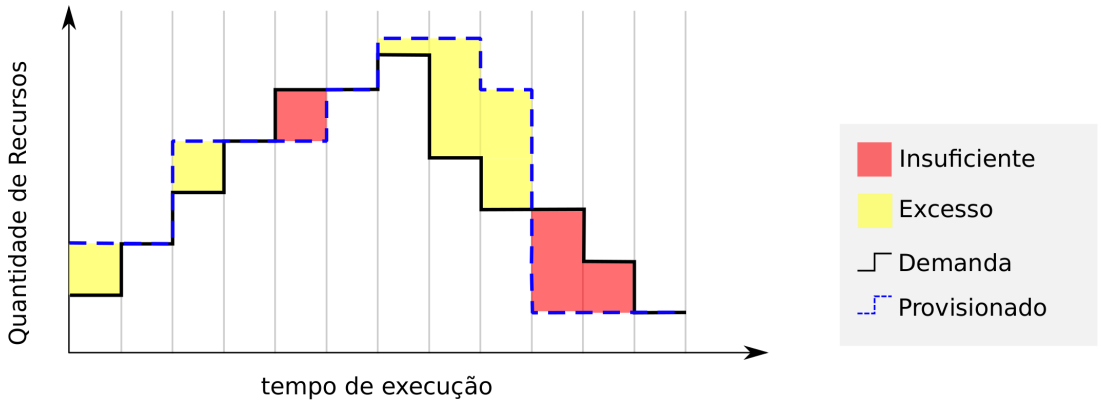


Figura 2.3: Provisionamento de recursos, que identifica as áreas em que há recursos em excesso e recursos em falta.

em regras, tomar ações elásticas. Na abordagem preditiva, utiliza-se de métodos matemáticos ou estatísticas capazes de predizer consumo dos RCs, utilizando estratégias heurísticas ou de inteligência artificial, e poder tomar ações elásticas julgando quando necessário. Os monitores baseado em regras, utilizam de uma métrica principal, como o uso de CPU, que deve manter-se dentre um intervalo e quando estes limiares são rompidos, há o disparo de ações de elasticidade. Esta abordagem é escolhida pelos provedores de nuvens comerciais devido, principalmente, por serem simples para os clientes configurarem e poderem predizer o total gasto com o serviço, porém a utilização pura deste método é questionável [31]. A teoria das filas é usada para relacionar as tarefas que são executadas em um sistema para estimar, por exemplo, o tempo de resposta. A Teoria do Controle aborda ajustar automaticamente os recursos baseado nas demandas da aplicação. O aprendizado por reforço é similar a Teoria do Controle, mas sem usar qualquer conhecimento sobre a aplicação. Ela tenta aprender, por tentativa e erro, o impacto que cada tomada de decisão em um determinado cenário, entretanto precisa de longas fases de treinamento. A análise temporal envolve métodos de detecção de padrões e métodos para prever valores futuros, baseado em amostras coletadas em uma janela de tempo.

Vários sistemas de monitoramento de recursos e tomada de decisão automática foram propostos ao longo dos últimos anos, como os trabalhos de Han [6] e Dutreilh [7]. Lorido-Botrán [31] relacionou vários trabalhos e propõe cinco categorias para estes sistemas, baseados no método em que a decisão é tomada: (i) Regras, baseada em limiares; (ii) Teoria das Filas; (iii) Teoria do Controle; (iv) Análise temporal (*Time-series analysis*); (v) Aprendizado por Reforço (*Reinforcement learning*).

Segundo Urgaonkar *et al.* [32], a abordagem preditiva é indicada quando a intensidade de uso do sistema apresenta um comportamento periódico, em que em determinados horários há uma variação considerável na utilização dos recursos. Moore et al. [33] e Urgaonkar et al. [32] apresentaram abordagens híbridas, em que o controle elástico é composto pelas duas formas, preditiva e reativa. A preditiva tomando as decisões baseadas nos padrões e a abordagem reativa avaliando se os recursos utilizados estão dentro de limiares pré-estabelecidos. Heinze et al. [34] utiliza de um monitor que verifica o tempo de resposta de um serviço periodicamente e, após n ocorrências de um determinado evento, decide as alteração necessárias. Após cada redimensionamento dos recursos, é necessário aguardar um período de tempo T para que o sistema perceba os novos recursos e comece a utilizá-los, ajustando os níveis da carga de trabalho.

A penúltima característica fundamental, método, é a forma de implementação da elasticidade, que podem ser de três maneiras: (i) Replicação (adicionar ou remover instâncias);

(ii) Migração (alterar a MV para outro hospedeiro); (iii) Redimensionamento (adicionar ou remover RCs).

A replicação envolve adicionar ou remover instâncias (elasticidade *horizontal*), como MVs completas (IaaS, aplicável também a SaaS e PaaS) para garantir que o serviço não seja interrompido, entretanto a aplicação fornecida deve suportar e estar configurada para utilizar esta estratégia. A migração envolve movimentar MVs entre máquinas hospedeiras de diferentes configurações, com o objetivo de manter o desempenho do serviço dentro dos padrões previamente definidos. O redimensionamento (elasticidade *vertical*) visa adicionar ou remover RCs a instâncias virtuais, como processadores, memória ou disco, possibilitando utilizar os novos recursos em tempo de execução.

O objetivo da elasticidade muda perante o ponto de vista: para o provedor, é ter os recursos computacionais organizados e alocados de forma otimizada, podendo oferecer mais serviços a mais clientes sem necessariamente adquirir maiores RCs físicos. Para o cliente, a elasticidade evita que recursos ociosos fiquem alocados, reduzindo os custos em serviços que pagam pelo uso (*pay-per-use*) e permite que o sistema seja dimensionado para que o desempenho não seja prejudicado em momentos de alta carga. Há também trabalhos descrevendo outros objetivos, como aumentar a capacidade de recursos locais [35] e economia de energia [36].

Ao analisarmos as aplicações científicas, estas comumente tem dois estágios principais: uma de uso intenso dos processadores e memória e outra de uso intenso de entrada e saída. Em um programa tradicional, durante o período de execução do programa, todos os recursos são alocados e só serão disponibilizados ao seu término [20]. Ao executar este programa em ambiente virtualizado e elástico, é possível que durante cada etapa seja alocado e liberado os recursos sob-demanda para cada estágio.

Dentro destas possibilidades que a elasticidade oferece junto a nuvens computacionais, torna-se interessante verificar o comportamento de aplicações neste tipo de arquitetura, mantendo estas apenas com os recursos necessários para seu funcionamento. Como exemplo de aplicação, iremos utilizar SGBDs, devido a sua alta aplicabilidade em NCs, por atenderem uma demanda variável de clientes em determinadas situações. Por isso é necessário verificarmos os tipos de SGBDs existentes, métodos de avaliação de desempenho e também métodos que possam levar o SGBD a um estado de alto consumo de recursos para que seja possível verificar se a elasticidade tem algum impacto.

Capítulo 3

Avaliação de desempenho e estresse em SGBDs

O crescente uso de redes sociais, aumento das compras via Internet e a redução dos custos de armazenamento, aumentou a quantidade de informação gerada e esta precisa estar organizada e disponível. Para isso, utiliza-se os Sistemas Gerenciadores de Banco de Dados (SGBD) para organizar e armazenar as informações. Alguns exemplos de SGBDs mais comuns são: PostgreSQL[37], MySQL[38], Oracle[39].

Recentemente, surgiu o conceito de *Big Data*, que segundo Gordon [40], pode ser definido como a combinação de cinco características: (i) Volume - quantidade de dados armazenada e analisada; (ii) Variedade - tipagem dos dados, possivelmente de vários lugares; (iii) Velocidade - os dados são produzidos em elevadas taxas; (iv) Valor - O quão importante a informação é para a organização ou empresa; (v) Veracidade - a corretude dos dados. Alguns dos SGBDs voltados para grandes dados são HBase [41] e Hypertable [42].

Possuindo estes dados agrupados em um único local, disponíveis para serem analisados, é possível verificar a existência de padrões nos dados e realizar previsões, como anúncios personalizados, informações sobre o clima ou comportamentos de consumidores. Para gerenciar essa grande quantidade de informação, é preciso utilizar SGBDs modernos, rápidos e capazes de gerenciar os dados sem apresentar inconsistências. Com o aumento do número de usuário e dos serviços que utilizam os SGBDs nos últimos anos [8], criou-se a necessidade de escalar este sistema, para que as operações OLTP (*Online Transaction Processing* ou Processamento de Transações em Tempo Real) possam ser executadas sem apresentar inconsistências ou lentidões, entretanto, foi detectado que estes SGBDs não eram escalável [43].

No início dos anos 2000, começaram a surgir os banco de dados *NoSQL* (*Not Only SQL*) [44], que se baseiam em formas diferentes dos relacionais para armazenar e manipular os dados. Um dos principais objetivos destes SGBDs é de serem escaláveis e distribuídos, mas para isso, foi necessário relaxar algumas regras aplicadas aos SGBDs relacionais. Em 2011, surgiram os chamados bancos de dados NewSQL, que são bancos de dados relacionais, mas com estratégias de escalabilidade dos bancos NoSQL [45].

Várias metodologias de análise (*benchmarks*) para SGBDs relacionais são encontrados na literatura, como Debit-Credit [46] e os desenvolvidos pela TPC [47]. Estas análises geram métricas e com elas é possível comparar e avaliar a capacidade de cada SGBD. Entretanto, estes *benchmarks* são executados com as máquinas em configurações estáticas e, não necessariamente, utilizam intensamente os recursos. Desta forma, Meira [48] propõe uma metodologia de classificação em estados para SGBDs, baseado no seu desempenho e um *benchmark* capaz de intensificar o uso de recursos e descobrir falhas que só são expostas sobre alta carga de trabalho.

O restante de capítulo está organizado da seguinte forma: Na Seção 3.1 será abordado os principais tipos de SGBD. A Seção 3.2 introduz conceitos sobre alguns testes de desempenho e estresse e a Seção 3.3 apresenta a máquina de estados DSM (*Database State Machine*).

3.1 Tipos de Bancos de dados

Existem atualmente duas categorias classificações de bancos de dados: Relacionais, NoSQL. Segundo Silberschatz [49], os bancos de dados relacionais gerenciam os dados baseados em relações, que são representados por um plano bidimensional, chamada de tabela, que contém atributos (colunas) que se armazena algum tipo de informação. MySQL [38] e PostgreSQL [37] são exemplos de SGBDs relacionais.

A manipulação dos dados é feita através da linguagem SQL (*Structured Query Language*), baseada na álgebra relacional e no cálculo relacional de tuplas, e na década de 80 tornou-se um padrão ANSI e ISO. Muitos bancos de dados não seguem exatamente estes padrões e possuem muitas vezes instruções específicas [49] na sua construção. Uma consulta SQL é um comando enviado ao SGBD para que seja executado e uma transação é um conjunto de instruções SQL que devem ser executados atomicamente.

Os SGBDs relacionais devem garantir que todas as transações submetidas atendam as propriedades ACID [50]: (i) Atomicidade (a transação deve ser executada por completo); (ii) Consistência (uma transação deve iniciar com estado consistente e finalizar em estado consistente); (iii) Isolamento (Transações concorrentes não são executadas sobre os mesmos dados); (iv) Durabilidade (após o término de uma transação, os dados devem ser persistidos, mesmo com sistema em falha). Por outro lado, a lógica complexa aliado ao aumento da quantidade de dados, trazem problemas quanto a concorrência, causando um rápido declínio no desempenho da leitura e escrita [51].

Os bancos categorizados como NoSQL (*Not only SQL*) tiveram um aumento de popularidade [52], pois os mecanismos de armazenamento e recuperação de dados são diferentes dos bancos relacionais, e suprem necessidades que foram criadas pela alta demanda dos serviços [51]: (i) Alta concorrência de escrita e leitura com baixa latência; (ii) Armazenamento eficiente para muitos dados e suportar tráfego intenso; (iii) Alta escalabilidade e disponibilidade; (iv) Redução de custos operacionais.

Os SGBDs NoSQL seguem o princípio BASE, que é um acrônimo do ACID dos relacionais, que significa: (i) Basicamente disponível (*Basically Available*); (ii) Estados Relaxados (*Soft State*); (iii) Eventualmente consistente (*Eventually consistent*). Devido a ter estas características, os problemas de consistência são passados para a aplicação, gerando um trabalho maior pela equipe de desenvolvimento [53].

O Teorema de CAP (ou Teorema de Brewer) [54] diz que é impossível um sistema possuir duas destas três propriedades: (i) Consistência (todos os nós possuem os mesmos dados ao mesmo tempo); (ii) Disponibilidade (toda requisição recebe uma resposta com sucesso ou falha); (iii) Tolerância a Partição (o sistema continua respondendo caso uma mensagem seja perdida ou falha de parte do sistema). Entretanto, 12 anos depois, Brewer [55] explica que é impossível obter estas três propriedades perfeitamente, mas os projetistas podem otimizar a consistência e disponibilidade, manipulando as partições[56]. Assim, os bancos NoSQL conseguem ganhar desempenho perante os relacionais, pois não atendem todas as propriedades ACID [57]. As características ACID forçam a consistência dos dados em cada fim de uma transação, enquanto BASE aceita que a consistência é um estado contínuo. Alguns exemplos destes bancos são MongoDB [58], HBase [41], Hypertable [42], Cassandra [59], MemcacheDB [60] e CouchDB [61].

Os bancos NoSQL podem ser classificados pela forma que os dados são organizados, como chave-valor; orientados a coluna (ou colunares); orientados a documentos [51, 62]; e em grafos [63].

O modelo chave-valor é o mais simples, pois os dados são vinculados a uma chave, e a velocidade das consultas é maior que o de uma base de dados relacional. Os colunares mantém a ideia de tabelas dos bancos relacionais, mas sem a associação entre elas. Cada coluna da tabela é armazenada separadamente, mantendo um índice principal, o que leva as operações em colunas demandarem menos uso do dispositivo de armazenamento. Os SGBDs orientados a documentos são similares ao chave-valor, entretanto o valor deste é armazenado em formatos JSON ou XML. Além disso, podem suportar um índice secundário, para facilitar o uso pela aplicação, que o modelo chave-valor não suporta. Os SGBDs baseados em grafos, possuem uma maior facilidade de manipular dados com muitas ligações, evitando junções (*SQL Joins*) recursivas [63] que custam mais tempo para serem executadas [64].

Existem perante algumas literatuass, a definição de bancos NewSQL, com diferentes técnicas de implementação, mas são bancos de dados relacionais, provendo todas as soluções transacionais e os requisitos ACID, mas com a habilidade de escalar dos bancos NoSQL [45]. Segundo Stonebraker [53], os bancos NewSQL precisam cinco características: (i) SQL como linguagem de consulta primária; (ii) Atender as propriedades ACID; (iii) Controle de concorrência não-bloqueante; (iv) Arquitetura com desempenho superior por instância, comparado aos SGBDs relacionais tradicionais; (v) Arquitetura escalável e *shared-nothing*[1] capaz de funcionar em vários nós sem gargalo. Alguns exemplos destes bancos são o NuoDB [65], VoltDB [66] e HStore [67].

3.2 Desempenho e Estresse em Banco de Dados

Dentro da área da computação, os profissionais envolvidos se preocupam em atingir os seus objetivos com máximo desempenho e menor custo possível. Para avaliar os sistemas computacionais (SC), é necessário uma metodologia padronizada, um objetivo e um resultado. A avaliação de desempenho é uma das possíveis avaliações e pode ser aplicada em cada estágio do ciclo de vida de um SC, do projeto ao uso legado, para que seja possível comparar diversas abordagens e encontrar a que melhor se encaixe nas necessidades [68]. Será abordado nesta Seção os principais testes de desempenho para banco de dados, suas aplicações e também sobre teste de estresse em SGBDs.

Existem três técnicas para avaliar um sistema: (i) Análise da modelagem; (ii) Simulação; (iii) Comparação. A escolha vai depender do estágio do ciclo de vida do SC, pois uma nova abordagem, por exemplo, só será possível realizar a análise da modelagem e a simulação, enquanto que a comparação só é possível quando há, no mínimo, dois SC com o mesmo objetivo [68].

Benchmark é o nome dado as cargas de trabalho (*workloads*) aplicadas a um SC e *benchmarking* a comparação entre SCs utilizando um *workload* e uma métrica [68]. Estas métricas variam dependendo do tipo de *workload* aplicado, o tipo de sistema em avaliação, quais parâmetros são importantes e qual o objetivo da aplicação. Para os SGBDs, uma das medidas utilizadas é o número de transações por segundo (TPS), mas alguns *benchmarks* utilizam consultas por segundo.

Existem vários tipos de *benchmarks* para SGBDs relacionais, sendo o Debit-Credit [46] um dos primeiros encontrado na literatura, que simula o funcionamento de uma rede bancária distribuída, com terminais, agências e uma unidade central, que usa a proporção de 1.000 agências, 10.000 caixas e 10.000.000 contas, para cada 100 transações por segundo (TPS) de

[1] Não há compartilhamento entre memória ou disco entre as instâncias.

objetivo. Se é necessário que o SGBD atinja 1.000 tps, o número de agências, caixas e contas deverá crescer na mesma proporção.

Em 1988, surgiu os *benchmarks* da *Transaction Processing Performance Council* (TPC) [47], em que cada um apresenta diferentes objetivos. Ela é uma organização sem fins lucrativos fundada para criar *benchmarks* de bancos de dados e processamento de transações que formalizou e definiu vários tipos de *benchmarks* ao longo de sua história, desde os que medem o desempenho de sistemas OLTP ao consumo de energia.

Os *benchmarks* TPC-A, TPC-C e TPC-E são voltados para sistemas OLTP (*Online Transaction Processing* ou Processamento de Transações em Tempo Real), simulando empresas como corretoras, livrarias e bancos. O TPC-A foi criado baseado no Debit-Credit, mas alterando alguns conceitos como o nível de isolamento, a arquitetura do sistema (abandonar arquitetura de *Mainframe* para cliente/servidor) [69] e o tempo de resposta.

Em 1990, foi criado o TPC-B, que específica um teste de estresse em SGBDs. Segundo Meira [70], o teste de estresse é uma variação do teste de desempenho e tem como objetivo submeter o sistema a uma carga de trabalho que seja capaz de causar falhas ou sobrecarregar o servidor, permitindo determinar a curva de desempenho, do baixo consumo de recursos até a degradação completa. Diferente dos outros *benchmarks*, como os de desempenho (TPC-C, TPC-H, TPC-E), o teste de estresse é voltado para a sobrecarga do sistema, sendo as métricas as mesmas (transações por segundo ou consultas por segundo).

O TPC-B é caracterizado por uma quantidade significativa de E/S em disco, tempo de execução moderado e integridade transacional e, assim como o Debit-Credit, simula uma rede bancária com vários terminais e agências. As condições para executá-lo são que:

(i) o sistema esteja em "*steady state*[2]" ; (ii) longo o suficiente para que os resultados possam ser reproduzidos; (iii) deve executar entre 15 minutos e 1 hora ininterruptos.

De todas as transações submetidas, 90% não pode ultrapassar o tempo máximo de 2 segundos e o SGBDs relacionais não pode deixar de atender as propriedades ACID, entretanto, os *benchmarks* atuais para SGBDs não são executados de forma distribuída, o que pode causar problemas de desempenho na máquina responsável pelo teste e contaminar o resultado [70].

O TPC-C é um *benchmark* projeto para simular um sistema de loja virtual, com aquisição de produtos, pagamentos, pedidos, controle de estoque, entre outras funcionalidade. Ele utiliza de transações de leitura com intensivas atualizações e a métrica utilizada é o número de novas ordens inseridas [69].

Outro fator que estes *benchmarks* propostos não avaliam, é verificar o comportamento do SGBD perante uma alta carga de trabalho, que cause perda de desempenho ou falhas geradas pela alta concorrência. Por esse motivo, Meira [48] propôs uma metodologia de teste voltado ao estresse do SGBD, que será visto na Seção 3.2.1. Este tipo de teste consegue explorar os limites de desempenho de um SGBD, sendo possível detectar possíveis gargalos para que o administrador da base de dados consiga otimizá-la, e também estimar os recursos que podem faltar durante este período de alta concorrência.

3.2.1 STEM - Metodologia para Teste de estresse

O STEM - *Stress Test Methodology* - é uma metodologia para de teste de estresse, voltado para ambientes transacionais de grande escala. Devido as limitações das metodologias de testes existentes, como a arquitetura centralizada, esta metodologia foi projetada para ser genérica e escalável, de forma que qualquer sistema transacional possa ser testado, revelando o comportamento sobre uma variedade de cargas de trabalho [48].

[2]Estado em que o sistema se encontra com um desempenho sustentável.

Este teste visa detectar degradações de desempenho e expor problemas de código interno que somente ocorrem na combinação de uma carga de trabalho alta ou de ajustes equivocados. Ele é executado com diferentes configurações do SGBD (alterando a memória de trabalho (*mem_work*) e número máximo de conexões (*max_conn*)) e de Carga de Trabalho, sendo estas simplificadas em três categorias: mínimo, médio e máximo. Como cada SGBD apresenta diferentes tipos de configuração que e a apenas a documentação de cada um pode indicar possíveis ajustes, cabe ao responsável pela análise determinar quais são os parâmetros internos relevantes para ser analisados em cada cenário neste teste.

A execução do método segue a ordem da Tabela 3.1, em que é dividido em cinco etapas, alternando nos valores da Configuração do SGBD e na carga de trabalho. A combinação destas três variáveis (memória de trabalho e máximo de conexões para a configuração do SGBD e carga de trabalho) criam os diferentes cenários possíveis. Algumas combinações, como *mem_work = minimo* e *max_conn = máximo* não são relevantes, pois um SGBD precisa de uma quantidade mínima de memória para suportar uma grande quantidade de transações.

Tabela 3.1: Tabela com as 5 etapas do STEM e seus objetivos.

	Mem.trabalho	Max. de conexões	Carga	Objetivo
1	Mínimo	Mínimo	Mínimo	Erros relacionados aos requisitos funcionais
2	Máximo	Mínimo	Máximo	Definir a base de degradação de desempenho
3	Máximo	Máximo	Mínimo	Ajuste do SGBD
4	Máximo	Máximo	Médio	Encontrar o Limite de desempenho
5	Máximo	Máximo	Máximo	Atingir a Degradação de desempenho

O primeiro teste busca verificar se o SGBD está funcionado conforme os requisitos funcionais. O segundo busca definir uma linha base de degradação, com o mínimo de conexões. O terceiro passo é para verificar o aspecto funcional após os ajustes no SGBD, se os ajustes causaram as alterações de desempenho esperadas. O quarto passo busca falhas relacionado ao limite de desempenho do SGBD. Para definir este limite, o SGDB é monitorado e comparado com o resultado do passo 2. O quinto passo tem como objetivo atingir a degradação do desempenho, causando estresse no SGBD até o momento de parada completa do serviço. Como forma de classificar o SGBD pelo desempenho, Meira [2] propõe uma máquina de estados baseado em algumas métricas de desempenho, que será detalhada na Seção 3.3.

3.3 DSM - *Database State Machine*

Um SGBD em ambiente de produção, necessita atender com eficiência, diferentes cargas de trabalho, gerada por aplicativos em *smartphones*, residências inteligentes, sistemas de alta concorrência e entre outros. Um método para representar SGBDs em ambiente de alta concorrência é utilizar a DSM - máquina de estados de banco de dados, que representa o SGBD em aspectos não-funcionais, em situações de alta concorrência, como o desempenho e a escalabilidade.

Meira [2] apresenta o DSM (*Database State Machine - DSM* - Máquinas de estados de banco de dados) como forma de classificar um SGBD pelo seu comportamento perante uma carga de trabalho, dividido em cinco estados, representado pela Figura 3.1. Para alternar entre os estados, é avaliado três variáveis de desempenho:

Variação de desempenho (Δ) - Cálculo do desvio-padrão no número de transações por segundo (TPS) em uma janela de tempo;

Vazão (δ)- A proporção de transações atendidas(y) por transações requisitadas por segundo(z). Quanto mais próximo de 1, o SGBD está atendendo a maioria das requisições. A vazão é calculada por $\delta = \frac{y}{z}$;

Tendência de desempenho (φ) - Calculado pela função $\varphi = x' - x$, onde x é o desempenho atual e x' o desempenho previsto (realiza algumas medições, interpola pelo método de quadrados mínimos e aplica a derivada).

Para cada SGDB avaliado, os limiares acima assumem valores específicos. São os cinco possíveis estados, representados pela Figura 3.1:

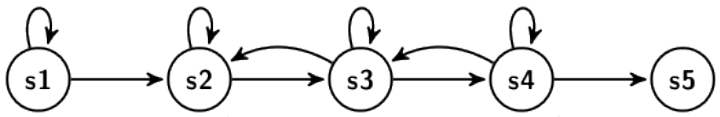


Figura 3.1: DSM - *Database State Machine*, retirado de Meira [2].

Aquecimento ($s1$): Considera toda a inicialização dos processos internos do SGBD e submissão de algumas transações para preenchimento dos *caches* na memória principal, por isso desempenho não é estável neste estado. A condição para este estado é que $\Delta < t_a$, sendo t_a o limiar de aquecimento e que Δ não esteja convergindo a zero;

Estável ($s2$): Este estado representa o SGBD quando está com o desempenho estável, os serviços inicializados e *caches* de memória preenchidos. Uma vez neste estado, o SGBD não volta ao estado de Aquecimento até que a execução do SGBD seja interrompida. A condição para este estado é que ($\Delta \rightarrow 0 \wedge \delta > t_e$), sendo t_e o limite do estado estável;

Sob-pressão ($s3$): Quando o SGBD está neste estado implica que ele está no limite do desempenho. O SGBD não consegue atender a maioria das transações de forma esperada, pois o total de requisições dos clientes está acima do limite. Talvez seja necessário um agente externo intervir para que volte ao estado Estável. A condição deste estado é $z > y \wedge \delta < t_e$ e que Δ seja estável;

Estresse ($s4$): A diferença deste estado para o estado *Sob-pressão* é que a variação de desempenho (Δ) aumenta e a vazão (δ) começa a cair. O SGBD neste estado fica extremamente vulnerável a interrupção do seu funcionamento, se nenhuma intervenção externa for feita, como alteração dos recursos computacionais ou adicionar novos nós de processamento. Se forem tomadas atitudes corretas, é possível voltar ao estado *Sob-pressão* e consequentemente ao estado Estável. A condição para estar neste estado é $\Delta < t_{st} \wedge \delta \rightarrow 0$, sendo t_{st} o limiar de estresse;

Falha iminente ($s5$): O SGBD entra neste estado quando está consumindo um grande número de recursos computacionais e atendendo um número decrescente de transações e deixa de ser possível sair deste estado, mesmo com intervenção externa. O DSM detecta a entrada do estado de *Falha iminente* se $\varphi < t_{th}$, sendo t_{th} o limiar de falha.

Este modelo não leva em consideração os recursos computacionais utilizados pelo SGBD, não sendo possível identificar diretamente quando há recursos alocados sem necessidade ou quando há demanda de novos recursos. Apesar dos estados **Sob-Pressão** e **Estresse** possam ser utilizados para indicar a necessidade de recursos adicionais, não existe garantia que estes recursos são necessários. Devido a isto, no Capítulo 4 iremos propor um modelo de provisionamento de recursos capaz de alterar, de forma elástica, os recursos computacionais alocados a uma aplicação, podendo evitar alocações desnecessárias.

Capítulo 4

MoBINE - Um modelo de provisionamento de recursos baseado em inferência de níveis de estresse

As nuvens computacionais apresentam vantagens importantes por não terem uma arquitetura centralizada, como a facilidade de acrescentar e remover recursos de forma dinâmica. Isso permite que os recursos sejam escalonados de forma eficiente, para que os usuários da nuvem utilizem os RCs necessários em cada instante de tempo, evitando que recursos desnecessários permaneçam alocados, o que aumenta os custos para o usuário ou provedor. Os principais problemas encontrados neste tipo de arquitetura é de definir um momento adequado para reajustar os recursos computacionais de uma aplicação e de estimar a quantidade de recursos necessária neste instante de tempo. Neste capítulo é apresentado uma abordagem que visa estimar um momento adequado de reajuste dos RCs baseado na classificação da aplicação em níveis de estresse diferentes, que nomeamos de MoBINE - Modelo de provisionamento Baseado em Inferência de Níveis de Estresse.

Em aplicações cliente-servidor, como SGBDs, existe uma correlação entre a quantidade de clientes, o tempo de resposta e o tipo de requisição. Com um crescente acesso de clientes, executando o mesmo tipo de requisição, os RCs da aplicação serão utilizados sem comprometer o desempenho esperado, até um instante de tempo em que a aplicação deixa de atender a sua especificação original, prejudicando o tempo de resposta para seus clientes. Quando o número de acessos dos clientes é reduzido, deixa de ser necessário manter alocado a mesma quantidade de recursos utilizados em alta demanda. Com diferentes tipos de requisição e quantidade de usuários, torna-se um desafio conhecer os recursos necessários para atender esta carga de trabalho.

Em um ambiente de nuvem computacional, com recursos elásticos, uma aplicação recebe uma carga de trabalho e ao longo da sua execução, esta carga pode sofrer variações, conforme mostra a Figura 2.2(a). Essas alterações de carga implicam em uma mudança no consumo dos RCs, que em momentos de alta carga podem não ser suficientes para atender toda a demanda, causando lentidão e instabilidade e quando há uma baixa utilização do serviço, há um aumento dos custos de se manter toda a infraestrutura. Desta forma, a possibilidade de variar os RCs baseado na carga da aplicação torna-se uma abordagem interessante, pois em situações ideais, garantirá que os RCs alocados sejam coerentes com a demanda existente gerada pela carga. Por isso, MoBINE estima um momento adequado e ordena à nuvem computacional que reajuste os recursos, de forma automática.

O restante deste capítulo está organizado da seguinte forma: A Seção 4.1 apresenta a descrição do modelo MoBINE e a sua composição, detalhando o seu funcionamento. Na Seção

4.2 será descrito o funcionamento da estratégia utilizada para realizar classificação aplicação, em diferentes níveis de estresse.

4.1 MoBINE - Descrição do modelo

MoBINE é um modelo de provisionamento elástico de recursos, projetado para ser reativo e automático, que pode ser aplicado em diferentes ambientes e serviços. Ele visa escolher um instante de tempo para reajustar os recursos, evitando que o usuário do serviço perceba oscilações no desempenho devido à carga de trabalho.

Para definir este instante de tempo, a carga é estimada pelo tempo de resposta da aplicação e classificada em níveis de estresse diferentes, que abordaremos com detalhes na Seção 4.2. Com este nível de estresse e informações sobre os RCs, como utilização e quantidade (variando as métricas dependendo do recurso), ações elásticas podem ser projetadas e comunicadas ao controlador da nuvem.

Toda aplicação necessita de requisitos mínimos de RCs para operar, o que já suporta uma determinada carga de trabalho para atender as requisições com um desempenho aceitável e dependendo do tipo da aplicação e das solicitações dos usuários, diferentes recursos podem ser consumidos mais intensamente. Um exemplo de aplicação são as científicas que comumente tem duas fases explícitas e desenvolvidas para consumir o máximo de recursos possível: de utilização intensa dos processadores e memória; e de uso intenso de entrada/saída [20]. Ao executarmos várias aplicações deste tipo simultaneamente, em um ambiente de RCs fixos, haverá um alto consumo de RCs e as aplicações irão terão um tempo de processamento acima do necessário, se fossem executadas individualmente.

Desta forma, variar os recursos computacionais nos momentos em que há alterações de carga, mensurado pelo tempo de resposta, torna-se uma abordagem interessante quando espera-se que a aplicação mantenha um comportamento especificado, como estabilidade e velocidade, pois isto reduz a necessidade de recursos e consequentemente os custos computacionais.

Inicialmente, para que seja possível atingir este objetivo de economizar RCs de uma aplicação sem prejudicar o desempenho é necessário que: (i) A infraestrutura, o sistema operacional e a aplicação suportem a elasticidade; (ii) Existam regras definidas sobre os RCs, como quantidade mínima e máxima disponível de cada recurso; (iii) A especificação dos tempos mínimos e máximos que a aplicação deve responder (velocidade); (iv) A definição das variações mínimas e máximas que o tempo de resposta deve permanecer (estabilidade).

Assim, é necessário que o modelo seja projetado para identificar alterações de desempenho, definir os recursos computacionais necessários e um momento adequado para realizar o reajuste. Por isso, o MoBINE foi projetado de forma modular, para que diferentes abordagens possam ser utilizadas em cada uma destas etapas, esperando os melhores resultados na organização e alocação dos recursos.

A Figura 4.1 mostra uma visão geral do MoBINE e como é a relação entre os módulos. Este é composto por quatro módulos: (i) Inferidor de estados; (ii) Controlador elástico; (iii) Controlador de Recursos; (iv) Monitor de Recursos. Cada um destes módulos deve possui regras específicas, especificadas no ANS (Acordo de Nível de Serviço) para que se obtenha parâmetros ideais de funcionamento da aplicação e as decisões tomadas pelo provisionador sejam adequadas.

Com uma carga submetida a uma aplicação implantada em uma nuvem, o Inferidor de estados classifica e encaminha o nível de estresse que a aplicação se encontra. O Controlador Elástico recebe este estado, determina se ações elásticas são necessárias e comunica o Controlador de Recursos (CR) que tipo de ação (aumentar ou diminuir) este deverá executar. Para o CR

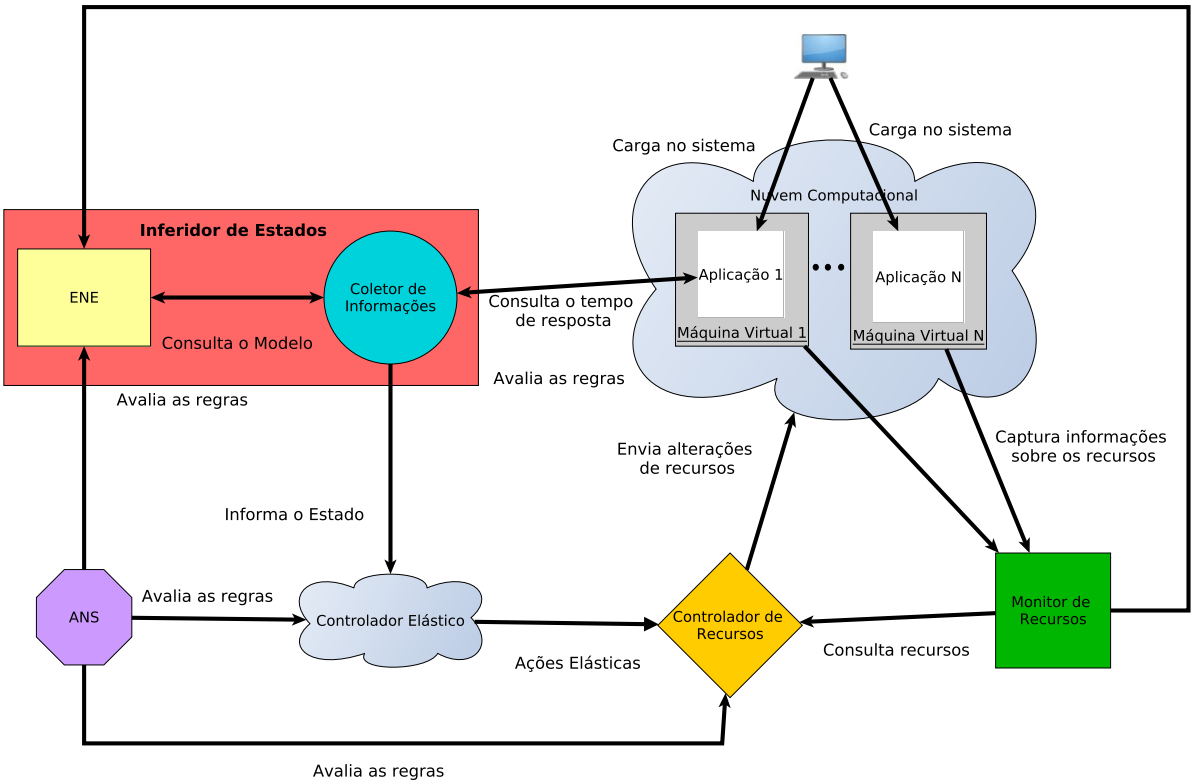


Figura 4.1: Visão geral do modelo MoBINE.

determinar qual será a nova configuração de recursos, este avalia as regras na ANS, definidas previamente, a estratégia utilizada e o consumo dos RCs, que recebe do Módulo de Monitoramento dos Recursos.

Desta forma, o modelo suporta outras estratégias de provisionamento, pois evita que exista acoplamento forte entre os módulos, facilitando a sua adaptabilidade em diferentes tipos de cenários, recursos e serviços.

Inferidor de Estados

O módulo *Inferidor de Estados* é responsável por estimar, baseado no tempo de resposta, a carga sofrida pela aplicação e comunicar ao Controlador Elástico. Este módulo é subdividido em dois itens: A Máquina de estados ENE (sua definição está na Seção 4.2) e um Coletor de Informações. Este coletor é composto por um fluxo de execução, que envia sondas de monitoramento da aplicação a cada J instantes de tempo, como mostra a Figura 4.2.

Cada sonda é responsável por abrir uma conexão com a aplicação, solicitar o processamento de uma informação, aguardar o retorno, calcular o tempo necessário para esta operação e armazenar estes dados. O processamento depende do tipo de serviço provido pela aplicação e deve ser definido por um especialista, para que seja refletido a real situação de carga do sistema. Se a maioria das solicitações dos clientes forem diferentes do processamento feito pelo inferidor, não há garantias de que os recursos alocados atendam os clientes.

As informações de todas as sondas são armazenadas em uma janela deslizante de K amostras, que a cada J instantes de tempo, calcula a média simples e a variância do tempo de resposta. Estes dados são encaminhados para a ENE que determina o nível de estresse da

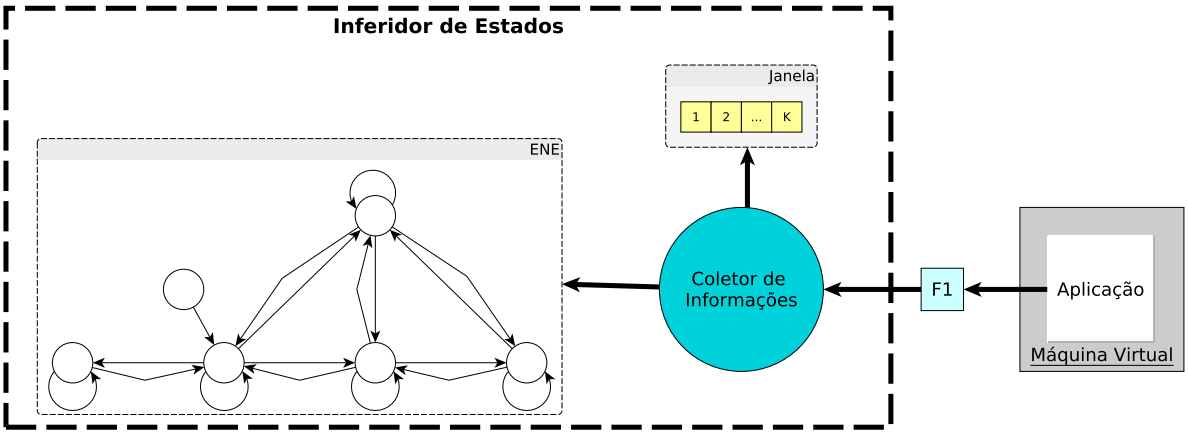


Figura 4.2: Detalhamento do módulo Inferidor de Estados. Este possui múltiplos fluxos de execução e com base nas sondas, consulta o ENE.

aplicação e então encaminha para o Controlador Elástico, para que este decida as possíveis alterações nos RCs.

Para definir os valores de K, J é preciso que seja analisado o tipo de aplicação, a relação de solicitações-consumo de RCs, para que o nível de estresse do sistema seja identificado no menor tempo possível com a maior precisão. A Figura 4.3 mostra uma representação das escolhas de valores para cada uma destas variáveis.

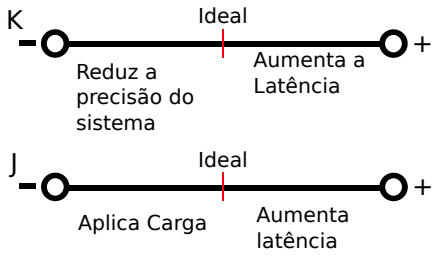


Figura 4.3: Representação das consequências de possíveis valores de K e J.

Algumas diretrizes podem ser seguidas para adequar os valores destas variáveis: Se K atingir valores maiores que o ideal, o sistema poderá demorar para identificar o nível de estresse correto, e com valores menores a precisão se perderá e o sistema poderá tomar ações desnecessárias. O intervalo de tempo J deve ser suficiente para que o Inferidor não interfira no desempenho da aplicação, mas também não aumente a latência de detecção na mudança do nível de estresse.

Controlador Elástico

O Controlador Elástico (CE) tem a função de determinar um momento adequado para alterar os RCs disponíveis para a aplicação e a Figura 4.4 mostra o diagrama de seu funcionamento. Ele é composto pelo histórico dos estados recebidos e por um Analista Automático, que é responsável por uma estratégia de avaliação do histórico e determinar as ações elásticas.

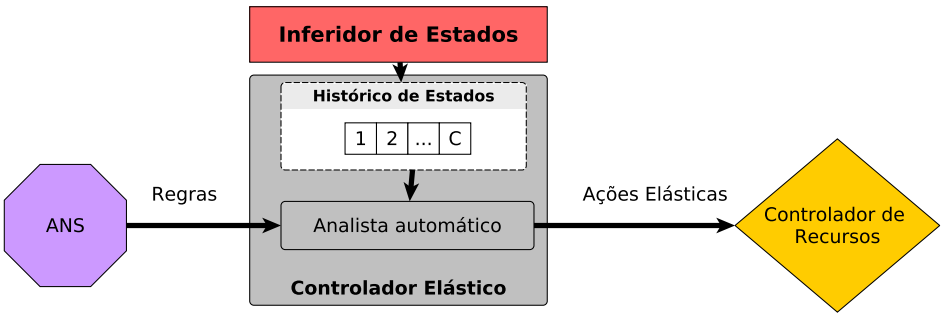


Figura 4.4: Detalhamento do módulo Controlador de Estados, que baseado na regras definidas na ANS e no histórico de estados, determina as ações elásticas que o CR deve executar.

A responsabilidade deste módulo é de enviar as ações elásticas para o Controlador de Recursos no momento adequado, para que evite desperdícios de RCs e consequentemente gere aumento de custos. Para isso, a abordagem escolhida é avaliar os últimos estados e verificar a tendência de aumento ou redução do nível de estresse. A tendência pode assumir três valores possíveis: (i) Aumento: quando há sucessivas amostras de estados que indicam estresse médio ou alto; (ii) Redução: quando há alguma leitura de nível de estresse nulo; (iii) Estabilidade: Nível de estresse entre baixo e médio. As definições dos níveis de estresse serão abordadas nas Seção 4.2.

Controlador de Recursos e Monitor de Recursos

Os módulos de Controlador de Recursos (CR) e o Monitor de Recursos (MR) são independentes entre si, mas complementares. O primeiro é responsável por estabelecer quais recursos devem ser alterados e comandar a Nuvem Computacional (NC) e o segundo por monitorar o consumo dos recursos das MVs, como por exemplo o uso de CPU e memória.

O Controlador de Recursos pode receber apenas duas ordens possíveis do CE: Aumente ou diminua os RCs em unidades pré-determinadas. Com esta ordem, ele avalia os recursos alocados e os dados atuais de consumo da aplicação para determinar quais recursos devem ser alterados e em qual quantidade. Ele também verifica regras definidas na ANS, para que se houver regras específicas de algum recurso ou grupo de recursos, possa realizar as alterações necessárias.

Como cada recurso computacional apresenta diferentes métricas, unidades de medida e latência para entrar em operação, é necessário que cada recurso seja analisado individualmente e escolhida uma abordagem adequada. Cada provedor de nuvem computacional deve fornecer estas informações sobre os RCs, devido a diferentes implementações de recursos virtualizados e custos para o usuário da NC.

O MR é responsável por coletar informações sobre o consumo dos RCs das MVs e enviar estes dados para o CR. Os dados coletados devem ser definidos baseado no tipo de aplicação e nas informações providas pela nuvem. Deve ser levado em consideração na implementação deste módulo, quais serão os métodos escolhidos para coleta das informações e análise dos possível problemas que podem neste processo, como perda temporária de conexão.

4.2 ENE - Estados de Níveis de Estresse

O ENE (Estados de Níveis de Estresse) é uma máquina de estados voltada para classificação de uma aplicação, considerando variáveis de desempenho e uso dos recursos

alocados como métricas. Esta máquina é baseada na DSM (*Database State Machine*, Máquina de estados para Banco de Dados, em tradução livre) [2], que categoriza um SGBD em cinco estados possíveis, baseados no desempenho que ele apresenta ao longo de uma carga crescente.

ENE incorpora alguns conceitos de desempenho do DSM e atribui novas variáveis, que adicionam uma maior complexidade ao modelo, para que represente o nível de estresse de uma aplicação. Com isto, foi necessário realizar alterações no número de estados, a composição das transições e as regras de disparo para mudança de estado. A Figura 4.5 apresenta o ENE em sua configuração final.

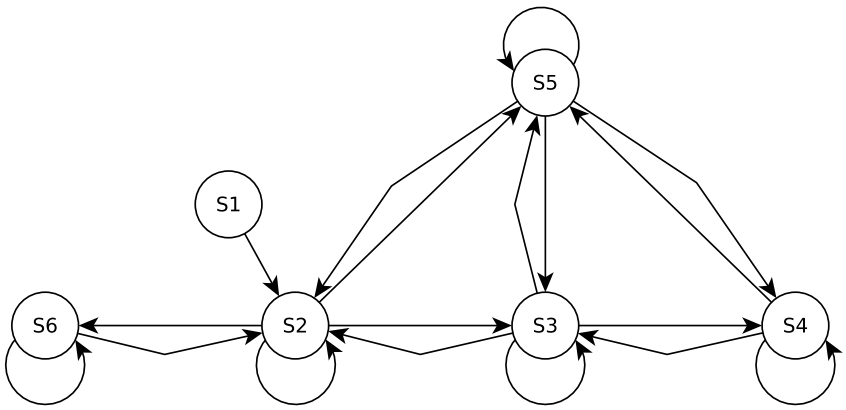


Figura 4.5: Máquina de estado ENE, baseada na DSM [2], que classifica uma aplicação em seis possíveis níveis de estresse.

A definição de cada estado foi projetada para analisar os possíveis problemas que a alta concorrência pode causar na aplicação, como lentidão, instabilidade ou perda de comunicação. A lentidão é caracterizada pelo tempo de resposta acima do esperado e a instabilidade que alguns clientes são respondidos rapidamente enquanto outros apresentam lentidão.

O estado $S1$ é chamado de **Aquecimento**, indica a inicialização da aplicação, que constitui do preenchimento de *caches* internos, que causa um atraso para primeiras solicitações. A sua transição a outro estado ocorre alguns segundos após a inicialização, quando o SGBD apresenta um desempenho estável. É considerado neste estado os recursos mínimos[1] alocados e após a estabilização do tempo de resposta em que o sistema deixa este estado.

Após a saída do estado de Aquecimento, a classificação seguinte é chamada de **Cruzeiro**, indicada por $S2$, que representa uma aplicação com baixo estresse, estável e respondendo com a velocidade controlada, com os recursos alocados serem condizentes com a carga submetida. Este é o estado indicado para que a aplicação seja mantida, visto que apresenta um menor consumo de recursos e coerência com a carga que a aplicação está sendo submetida.

Os estados $S3$ e $S4$ representam a aplicação com o desempenho aceitável, mas fora do ideal, indicando que os RCs alocados são insuficientes e é necessário alocar novos RCs para que o sistema retorne ao estado $S2$. O $S3$, chamado de **Sob-Pressão**, indica um sistema com tempo de resposta acima do esperado, caracterizando lentidão, mas apresenta estabilidade, representando um estresse baixo. $S4$ é o estado de **Estresse** alto da aplicação, tanto pela lentidão quanto pela instabilidade, sendo os recursos atuais insuficientes para a carga aplicada.

Quando torna-se impossível determinar o tempo de resposta da aplicação, ou um tempo de resposta é superior ao limite definido, a aplicação entra no estado **Fora da especificação** ou

[1]Todo *software* necessita de requisitos de *hardware* mínima para operar, que naturalmente suporta uma quantidade de carga.

desconhecido, $S5$. Isso pode ocorrer devido a alguma interferência, falha de comunicação ou um aumento intenso da carga no sistema, que sugere-se uma abordagem emergencial visto que não há informações recentes sobre o nível de estresse da aplicação. Uma das abordagens possíveis é aumentar massivamente os RCs ou solicitar interferência externa, para que seja avaliado o comportamento não esperado e solucionado a falha de comunicação ou lentidão.

Ao contrário do estado $S5$, o estado $S6$ representa uma aplicação rápida, estável, mas com recursos alocados sem necessidade e com estresse nulo, indicando que há uma **Sobra** de RCs, sendo recomendável que os recursos sejam removidos. Caso seja retirado até que se atinja o mínimo possível, respeitando as regras na ANS, o estado é alterado para $S2$.

O disparo das transições entre os estados, avaliam quatro variáveis, sendo duas de desempenho e duas binárias. As variáveis de desempenho visam identificar a velocidade e a estabilidade da aplicação e são elas:(i) **Variação do tempo de resposta** - Variância do tempo de resposta de um conjunto de sondas enviadas representado por Δ, que indica estabilidade; (ii) **Tempo médio de reposta** - O tempo de resposta em segundos de um conjunto de sondas, representado por δ, que indica a velocidade.

As variáveis binárias visam identificar se os recursos alocados são mínimos e se a aplicação está respondendo em tempo aceitável, para que exista informações atualizadas sobre seu estado. Estas são definidas como: (i) R **- Recursos mínimos** - Representa a quantidade de recursos alocados, sendo definido como mínimo ou não; (ii) M **- Há resposta** - Indicando se houve resposta da aplicação em tempo t.

As transições entre os estados representam alterações no comportamento da aplicação, seja por oscilações na carga, falha de rede ou alguma outra aplicação interferindo nos recursos em ambientes compartilhados. Para que seja definida as transições, é necessário definir alguns limiares de velocidade e estabilidade para cada estado, que estão disponíveis na Tabela 4.1.

Tabela 4.1: Definições dos limiares para cada estado do sistema, em unidades de tempo (por exemplo, segundos).

Variável	Representação
V_C	Limiar de estabilidade para o estado Cruzeiro.
V_P	Limiar de estabilidade para o estado Sob-Pressão.
V_E	Limiar de estabilidade para o estado Estresse.
T_{min_C}	Tempo de resposta inferior do estado de Cruzeiro.
T_C	Tempo de resposta superior do estado de Cruzeiro.
T_P	Tempo de resposta máximo para estado de Sob-Pressão.
T_E	Tempo de Resposta máximo para estado de Estresse.

Os valores dos limiares devem ser definidos baseado no tipo de sonda enviada pelo Coletor de Informações, para que o ENE represente o nível real de estresse da aplicação. É recomendável que seja analisado os dados enviados pelas sondas quando não há carga sendo aplicada ao sistema como base para escolha destes valores. Com estes limiares definidos, a Tabela 4.2 mostra as regras de disparo de cada transição do ENE, que irão definir o nível de estresse da aplicação.

Com esta máquina de estados acoplada ao MoBINE, torna-se possível estabelecer o nível de estresse da aplicação e compreender o comportamento, para então provisionar recursos de forma que seja utilizado apenas o necessário para manter a aplicação estável e com rapidez.

Tabela 4.2: Condições de transição entre os estados do ENE.

Origem	Destino	Condição	Explicação
S1	S2	$\delta > 0 \wedge \Delta \geq 0 \wedge M$	Há resposta da aplicação.
S2	S2	$(R \wedge (\Delta < V_C \wedge \delta < T_C)) \vee (\neg R \wedge (\Delta < V_C \wedge \delta \subset [T_{min\_C}; T_C]))$	A aplicação responde com velocidade e estabilidade, verificando os recursos mínimos.
S2	S3	$\Delta > V_C \vee \delta > T_C$	A variância ou o tempo de resposta ultrapassam os limiares de cruzeiro.
S2	S5	$\neg M$	Se não há conexão com a aplicação.
S2	S6	$\neg R \wedge \Delta < V_C \wedge \delta < T_{min\_C} \wedge M$	Se há resposta, estabilidade, velocidade e não estiver em recursos mínimos.
S3	S2	$\Delta < V_C \vee \delta < T_C$	Se o sistema está estável e rápido.
S3	S3	$\Delta \subset]V_C; V_P] \vee \delta \subset]T_C; T_P]$	Se a variância ou o tempo de resposta estão com valores superiores ao limite de cruzeiro, mas inferiores ao de pressão.
S3	S4	$\Delta > V_P \vee \delta > T_P$	Se a variância ou tempo de resposta ultrapassa os limites do estado de Pressão.
S3	S5	$\neg M$	Se o sistema não responde.
S4	S3	$\Delta < V_P \vee \delta < T_P$	Se a variância ou o tempo estão abaixo do limite do estado de Pressão.
S4	S4	$\Delta \subset]V_P; V_E] \vee \delta \subset]T_P; T_E]$	Se a variância ou o tempo de resposta estão com valores acima dos respectivos limiares, mas abaixo do limiar de Estresse.
S4	S5	$(\Delta > V_E \vee \delta > T_E) \vee \neg M$	Se o tempo de resposta ou variância estão com valores acima do limiar de estresse ou quando não há resposta da aplicação.
S5	S4	$\Delta < V_E \vee \delta < T_E \vee M$	Se as leituras do tempo de resposta e da variância estão abaixo dos limiares de estresse e há resposta.
S5	S5	$\neg M \vee (\Delta > V_E \vee \delta > T_E)$	Se o tempo ou variância estão acima dos limiares de estresse ou não há resposta.
S6	S2	$R \vee (\Delta < V_C \wedge \delta > T_{min\_C})$	Se os recursos são mínimos ou a variância está abaixo do limiar de cruzeiro e o tempo de resposta está acima do tempo mínimo de cruzeiro.
S6	S5	$\neg M$	Se não há resposta da aplicação.

Para avaliar esta proposta, no Capítulo 5 será mostrado alguns experimentos realizados com este modelo, mantendo uma mesma carga de trabalho e variando o número de clientes e será comparado com outras duas abordagens comumente usadas no provisionamento de recursos.

Capítulo 5

Resultados Experimentais

Neste capítulo será abordado alguns experimentos efetuados, com diferentes cargas de trabalho e com três técnicas de provisionamento de recursos, sendo uma delas a implementação do modelo proposto no Capítulo 4. Estes experimentos foram projetados para seja reproduzido algumas situações que podem ocorrer em ambientes de produção, como oscilação da carga e aumento intenso do número de clientes conectados.

O modelo proposto foi desenvolvido utilizando a linguagem Java, pois é consagrado pelo mercado e possui diversas bibliotecas padronizadas, podendo tornar a aplicação compatível com diversos SGBDs. A aplicação utilizada nestes experimentos é o PostgreSQL, na versão 9.5, um SGBD relacional tradicional amplamente utilizado pela área acadêmica e indústria. Será avaliado o provisionamento vertical de recursos computacionais em tempo de execução, alternando o número de núcleos alocados a uma MV dedicada a esta aplicação. Foram executados diversos testes analisando o comportamento do consumo de memória e seu provisionamento, entretanto não foi possível estabelecer uma estratégia eficiente utilizando o método proposto, nem de métodos baseado em regras. Isso ocorreu devido ao desempenho de um SGBD estar fortemente ligado aos seus parâmetros de uso de memória, sendo que pequenas alterações nestes parâmetros causam diferentes impactos no consumo de memória.

Ao se trabalhar com variação na quantidade de memória no XEN, ele adota a estratégia de *ballooning* [71], que realiza alterações na memória total de forma gradual, em tempo de execução, que dependendo da carga submetida na MV pode causar a chamada *memory pressure*, devido a falta momentânea de memória. Essa latência entre a ordem de alteração e a disponibilidade completa para a MV pode ocasionar problemas quando há picos de conexão não previstos, obrigando o sistema operacional (SO) da MV adotar medidas de contenção dos recursos e causar falhas para algumas conexões ou a parada completa do serviço. Outros valores poderiam ser otimizados, como tempo entre a amostragem ou o total mínimo livre, mas desta forma aumentaria o desperdício de memória ou um uso excessivo de recursos apenas para monitoramento. Galante e Bona [72] propõem adicionar um controle de provisionamento elástico em nível de programação, que pode-se ser analisado sua aplicação em SGBDs, podendo ser aplicado nos momentos anteriores a alocação de memória, podendo obter resultados significativos do que abordagens reativas para este tipo de provisionamento.

Este capítulo está organizado da seguinte forma: Na Seção 5.1 será mostrado o ambiente de teste criado, os detalhes da implementação e as cargas de trabalho usadas. Na Seção 5.2, será mostrado como os experimentos estão configurados e separados. Por fim, na Seção 5.3 será exibido e discutido os resultados obtidos, comparando entre as diversas configurações propostas.

5.1 Configuração dos experimentos

O ambiente foi projetado e configurado para que as interferências nos testes sejam mínimas, evitando que diferentes processos impactem no provisionador e na aplicação para que não ocorra oscilações entre a comunicação da aplicação com clientes e ao provisionador. A Figura 5.1 exibe como foi organizado o ambiente de testes.

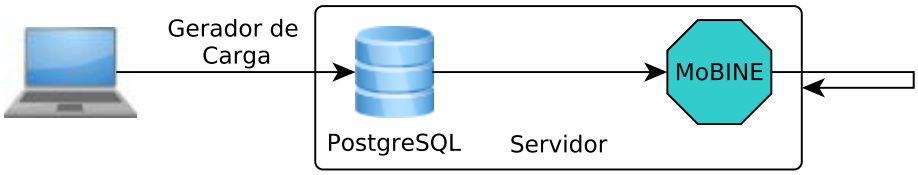


Figura 5.1: Organização do ambiente dos testes.

Uma estação de trabalho envia um conjunto de consultas a um SGBD, implantado em uma MV dedicada, gerando carga na aplicação. O provisionador é executado na máquina provedora da infraestrutura elástica, evitando oscilações de rede ocorram e monitoramento do SGBD não seja afetado, podendo assim efetuar alterações nos RCs quando julgar necessário. O SGBD escolhido para estes testes é o PostgreSQL 9.5, que está implantado em uma MV dedicada, virtualizada utilizando o XEN Hypervisor [73], de versão 4.4.2. Tanto a MV quanto o provisionador, estão sendo executados por um servidor dedicado para estes experimentos.

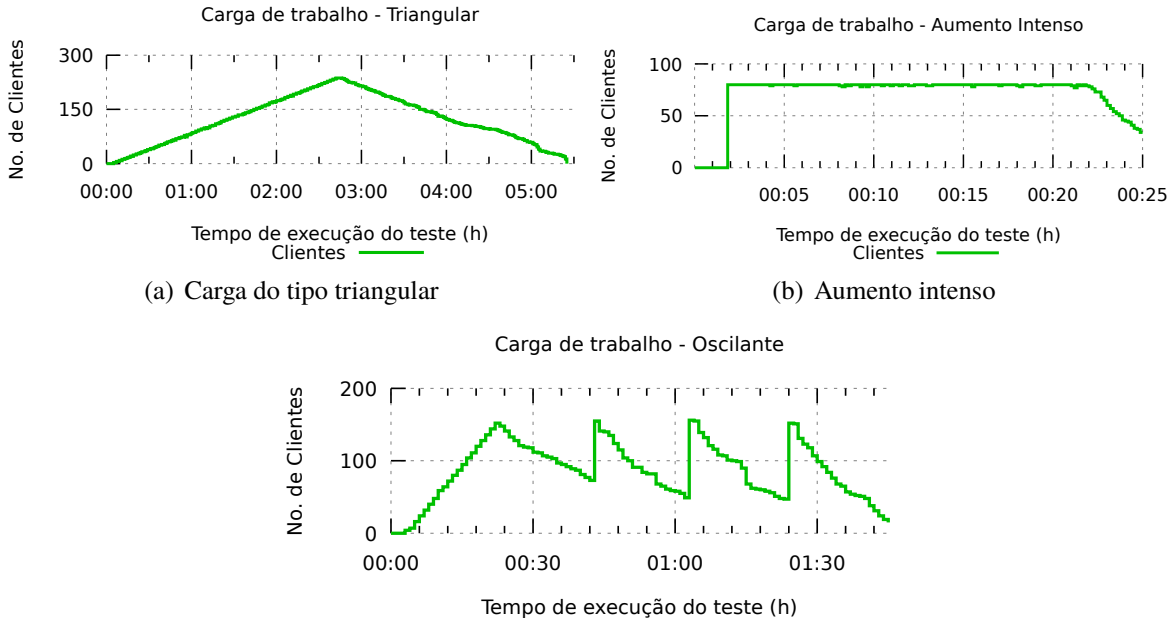
O servidor possui três processadores AMD Opteron 6136, totalizando 24 núcleos, 96 GB de memória e dois discos rígidos de 1 TB. A MV foi configurada com 40 GB de memória fixos, com a quantidade de processadores variando entre 2 a 16 núcleos e 250 GB de espaço em disco. Pelo SGBD escolhido ser do tipo relacional tradicional, foi dedicado um disco rígido para sua execução. A estação de trabalho possui um processador Core i5 com 4 GB de memória. Em todas as máquinas, físicas e virtuais é utilizado a distribuição Ubuntu-Server 14.04.4, com *Kernel* 3.13.0 – 77.

O hipervisor XEN foi mantido com suas configuração originais, visto que não é o objetivo desde trabalho obter o melhor desempenho do SGBD em ambientes virtualizados, mas demonstrar o impacto na quantidade de recursos causa no seu desempenho e uma estratégia de provisionamento. O provisionador é executado diretamente pelo SO da máquina servidora, para que não ocorram interferências no SGBD e que oscilações na rede de dados influencie os resultados.

5.2 Metodologia

Os experimentos foram projetados para avaliar diferentes quesitos relacionados a algumas estratégias de provisionamento e a utilização de recursos computacionais. Para isto, foram executados experimentos variando dois quesitos: (a) Carga de trabalho; (b) Estratégia de provisionamento. A primeira visa avaliar o provisionamento e consumo dos recursos com diferente número de clientes conectados e submetendo consultas ao SGBD. O segundo visa comparar diferentes abordagens de provisionamento, tendo como objetivo principal o menor consumo de recursos possível, desde que respeitada as regras. O SGBD foi mantido em sua configuração padrão.

Para mostrar o comportamento do provisionador e o consumo de recursos, perante os possíveis cenários de utilização do SGBD, foi estabelecido três diferentes cargas de trabalho, indicados pela Figura 5.2: (a) Crescente e decrescente; (b) Constante de início intenso; (c) Oscilante. A primeira visa mostrar um ambiente com aumento gradual do número de clientes e em seguida uma redução controlada para avaliar os momentos em que os recursos são alterados pelas diferentes estratégias de provisionamento. A segunda visa mostrar o comportamento em situações de carga constante, verificando o comportamento do provisionador. A terceira carga de trabalho visa validar o provisionamento, verificando o comportamento das estratégias se o consumo dos recursos torna-se coerente com a demanda.



(a) Carga do tipo triangular

(b) Aumento intenso

(c) Carga oscilante

Figura 5.2: Diferentes tipos de carga submetida ao SGBD durante os testes.

As estratégias de provisionamento utilizadas nestes experimentos são: (a) Tempo de Resposta - TR; (b) Uso de CPU - uCPU; (c) MoBINE. A primeira estratégia avalia o tempo de resposta como método de provisionamento, com uma janela amostral de 4 leituras. É feito a média aritmética desta janela e analisado se o tempo médio obtido neste intervalo está entre 1 e 2 segundos para um consumo coerente de recursos. Caso o valor seja inferior a 1, e os recursos alocados são superiores ao mínimo estabelecido, indica que a remoção de alguns recursos é necessária e caso o tempo seja superior a 2 segundos, é necessário que novos recursos sejam alocados para que o tempo médio seja reduzido. As operações elásticas alteram o número de núcleos da CPU, em uma unidade, adicionando ou removendo e é adicionado um atraso de 5 amostras para que novas ações elásticas sejam feitas, para evitar oscilações das operações elásticas.

A segunda avalia o uso de CPU, indicado pelo SO da MV, em porcentagem como métrica principal de provisionamento. Foi estabelecido empiricamente, que o uso de CPU entre o intervalo $[30\%; 70\%]$ indica o uso ideal dos recursos, sendo que uma utilização abaixo do intervalo acarreta em ações elásticas retirando recursos, enquanto acima é necessário adicionar novos recursos computacionais. As operações adicionam ou removem um núcleo de vCPU de cada vez.

A terceira estratégia, MoBINE, proposta no Capítulo 4, utiliza a classificação em 5 estados para a tomada de decisão das ações elásticas. A classificação nestes estados é baseada no

tempo de resposta de uma consulta, armazenada em uma janela de 4 amostras. Com a média e variância desta janela, é estabelecido o nível de desempenho da aplicação, indicado pelos estados e o provisionador estabelece quais as ações elásticas são necessárias. A Tabela 5.1 mostra os limiares escolhidos em cada estado.

Tabela 5.1: Valores de tempo de resposta e variância para o PRE (em segundos).

	Sobra	**Cruzeiro**	**Sob-Pressão**	**Estresse**	**Inesperado**
Tempo	< 1	$[1, 2]$	$]2, 5]$	$]5, 10]$	> 10
Variância	< 2	< 2	$[2, 5]$	$[5, 10]$	> 10
Operação - CPU	-1	0	+1	+1	+2

Estas estratégias foram implementadas em um protótipo escrito em Java, pois é uma linguagem consagrada e possui diversas bibliotecas disponíveis, reduzindo a complexidade no desenvolvimento deste protótipo. Ele foi implementado de forma modular, como especificado no Capítulo 4, sendo possível neste protótipo, alterar a tomada de decisão entre os diferentes tipos de estratégias. Desta forma, na Seção 5.3 está descrito os resultados obtidos e os pontos principais das execuções foram discutidos, realizando comparações entre estas abordagens.

5.3 Experimentos realizados

Foram realizados diferentes experimentos, variando a carga submetida ao SGBD e a estratégia de provisionamento. Foram realizadas amostras a cada 10 segundos e os dados obtidos foram agrupados por minuto para melhor visualização, sendo a regra de agrupamento escolhida a média aritmética simples, exceto para o número de vCPUs, que é considerado o maior encontrado no período do agrupamento.

Esta Seção está dividida em três categorias, uma para cada tipo de carga: Triangular, Intensa e Oscilante. Em cada seção será mostrado os resultados obtidos com cada um dos métodos de provisionamento, comparando as abordagens perante o uso dos recursos e o tempo de resposta obtido em cada situação.

5.3.1 Carga Triangular

Esta carga de trabalho, apresentada na Figura 5.2(a), tem como objetivo avaliar a eficiência das estratégias de provisionamento perante a cargas de trabalho crescente e decrescente. A carga é descrita da seguinte forma: Do início da execução até o momento 02:45h a carga é crescente até atingir o limite definido empiricamente de 237 clientes e, após isso, o total de clientes é reduzido de forma controlada. Considerando que cliente ser um fluxo de execução distinto e executar diferentes consultas com complexidades distintas, é definido uma função em que a cada instante de tempo é calculado o total de clientes necessário, fazendo adequações para que o número de clientes seja o calculado.

O método MoBINE, durante a carga crescente, manteve a maior parte do tempo abaixo do limiar de Cruzeiro com o tempo médio de resposta de $1,679$ segundos. Houve diversas leituras acima de 2 segundos, que alterações no número de vCPUs causaram a redução do tempo de resposta, mas como a carga é crescente nestes pontos, os aumentos acontecem novamente e novas medidas são necessárias.

A Figura 5.3 apresenta o comportamento do provisionador de vCPU e o tempo de resposta da aplicação durante as cinco horas de execução (aproximado). Durante a primeira hora, vemos que o tempo de resposta não ultrapassa o limite de Cruzeiro, de 2 segundos, e o número de vCPUs aumenta até 13 núcleos. Após isso, entre o período [01:00; 01:30] houve uma queda do tempo abaixo do limite inferior de Cruzeiro, retirando vCPUs e causando um novo aumento no TR, economizando recursos.

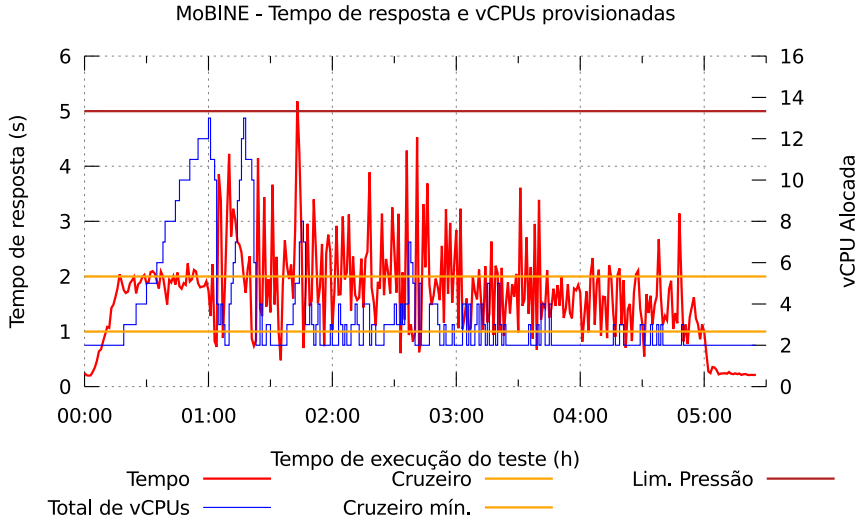


Figura 5.3: Tempo de resposta e o provisionamento de vCPU do MoBINE, para carga Triangular.

Em seguida, nota-se que houve um aumento acima do limiar de Cruzeiro, causando novas ações para adicionar novos vCPUs, indicando que as remoções feitas anteriormente foram além do necessário, causando uma instabilidade no SGBD, como mostra a Figura 5.4. Após isso, no instante 02:30h, houve novamente uma nova oscilação no provisionamento, mas menos intensa e o comportamento seguiu de forma considerada estável, variando entre 2 e 4 o núcleos de vCPU provisionadas.

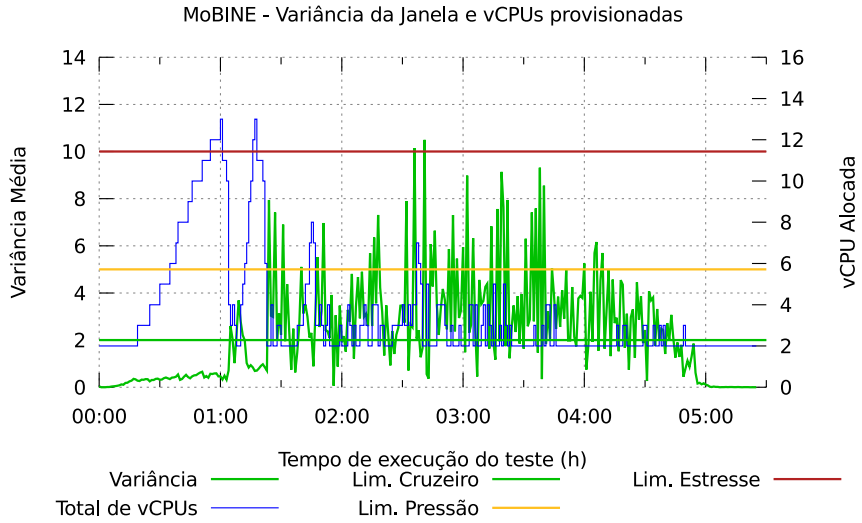


Figura 5.4: Variância e o provisionamento de vCPU do método MoBINE, para carga Triangular.

A Figura 5.4 mostra a variância da janela e os limiares de cada estado atribuídas para esta variável. Podemos ver que ela no inicio do teste permanece abaixo do limiar de Cruzeiro,

indicando um sistema estável até a primeira hora do teste. Após isso, a variância da janela oscilou de zero até 10 segundos, ficando a maior parte do tempo entre [2; 6], classificando entre os estados de Pressão e Estresse.

Na Tabela 5.2 podemos ver os resultados do provisionamento pelo método MoBINE. A média do tempo de resposta obtido foi de $1,679 \pm 1,872$ segundos, revelando que durante a maior parte da execução, o SGBD respondeu dentro do intervalo esperado. Com a média da variância da janela em $2,295 \pm 2,169$ segundos, vemos que a maior parte do tempo a aplicação ficou no classificada em Sob-Pressão, indicando que na maior parte do tempo, o SGBD não foi classificado como estável.

Tabela 5.2: Tabela analítica do método MoBINE, para carga Triangular.

MoBINE	**Média**	**Mínimo**	**Máximo**	**Desvio Padrão**
Tempo	1,679	0,199	5,179	0,872
Variância	2,295	0,000	10,495	2,169
Total de CPU	3,512	2,000	13,000	2,630

Com uma máxima de $5,179$ segundos, o estado de Estresse foi atingido apenas uma vez próximo a segunda hora de execução, indicando que o provisionamento naquele momento foi insuficiente, apresentando um tempo de resposta não condizente com o objetivo. A média de vCPUs alocadas foi de $3,512 \pm 2,630$ unidades, alternando entre 2 e 13 unidades.

Com o método TR, que utiliza apenas a média do tempo de resposta para provisionar vCPUs, a Figura 5.5 apresenta o tempo médio de resposta com o número de núcleos alocados. O comportamento de modo geral foi semelhante ao encontrado no método MoBINE, entretanto, durante cenários de tempos de resposta baixos, o comportamento do provisionador é conservador, evitando alterações drásticas nos recursos, o que causa um aumento da latência de alteração.

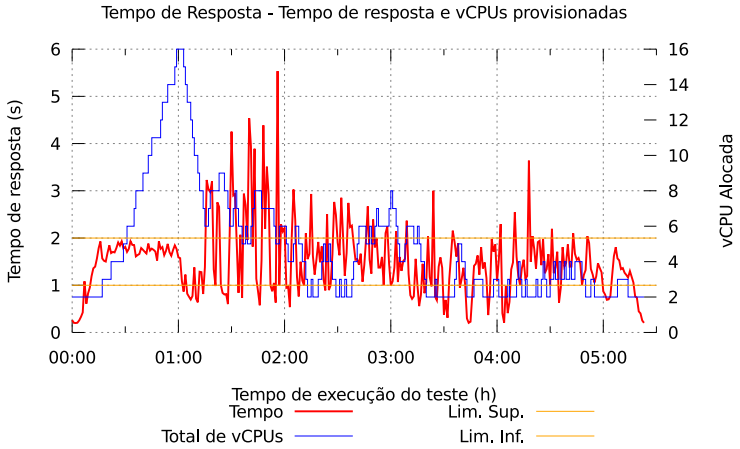


Figura 5.5: Tempo de resposta e o provisionamento de vCPU do método TR, para carga Triangular.

A Tabela 5.3 mostra algumas estatísticas da execução deste método que obteve $1,513 \pm 0,753$ segundos de tempo médio de resposta, com máxima de $5,535$ segundos. Para isso, foram necessários $4,799 \pm 3,220$ núcleos para manter o SGBD respondendo com uma variância média de $1,493 \pm 1,546$, indicando um SGBD estável dentro do especificado.

Tabela 5.3: Tabela analítica do método TR, para carga Triangular.

Tempo de Resposta	Média	Mínimo	Máximo	Desvio Padrão
Tempo	1,513	0,200	5,535	0,753
Variância	1,493	0,000	7,782	1,546
Total de CPU	4,799	2,000	16,000	3,220

O método uCPU apresentou um comportamento diferenciado perante as outras abordagens, o que garantiu um tempo de resposta abaixo do intervalo objetivo, como mostra a Tabela 5.4. O tempo médio obtido foi de $0,532 \pm 0,401$ segundos, tendo como máximo $2,052$ segundos, os menores valores entre os três métodos. Com a variância média da janela de $0,171 \pm 0,316$, podemos afirmar que este método apresentou resultados de desempenho acima do esperando, no entanto, foram provisionados em média, $14,747 \pm 2,482$ núcleos, com um valor máximo de 16, indicando que na maior parte do tempo, foram utilizados entre 3 a 4 vezes mais núcleos que os outros métodos, como a Figura 5.6.

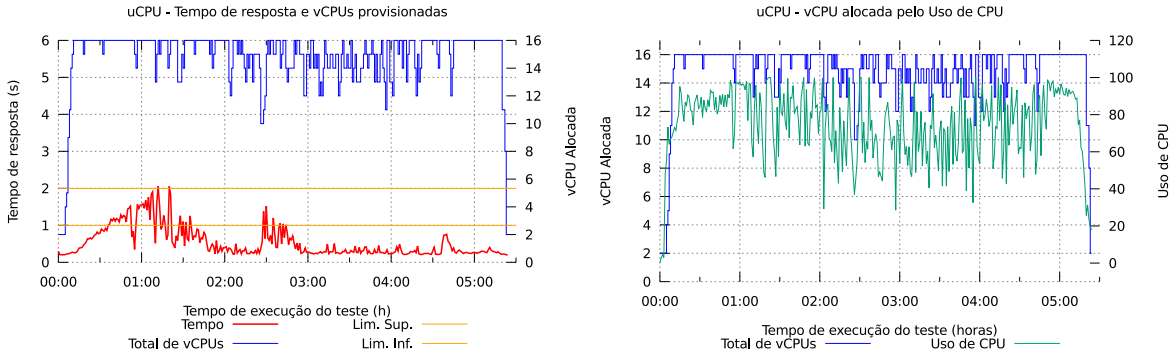
Tabela 5.4: Tabela analítica do método uCPU, para carga Triangular.

uCPU	Média	Mínimo	Máximo	Desvio Padrão
Tempo	0,532	0,197	2,052	0,401
Variância	0,171	0,000	2,536	0,316
Total de CPU	14,747	2,000	16,000	2,482
Uso de CPU	74,196	0,000	100%	19,458

Isso ocorreu devido a métrica utilizada ser o uso de CPU não variar conforme o número de clientes conectados, mas aos processos criados que a CPU deve executar, que mesmo com uma quantidade baixa de clientes, o uso de CPU chega ao máximo, como indicado no gráfico da Figura 5.6(b). Com isso, o método baseado nesta métrica, aumenta o número de núcleos para que os processos sejam divididos entre os núcleos, reduzindo o total médio utilizado. O uso médio de CPU foi de $74,196\% \pm 19,458\%$, indicando que em grande parte do tempo, o provisionador adicionou núcleos, mesmo que o tempo de resposta seja baixo.

Desta forma, a estratégia uCPU apresentou melhores resultados em velocidade e estabilidade, ao custo de utilizar a maioria dos recursos disponíveis e os métodos MoBINE e TR apresentaram resultados semelhantes de provisionamento e tempo de resposta, como indica a Figura 5.7. Podemos notar que, quando o número de clientes está dentre o intervalo [30; 120], durante as primeiras horas do teste, ocorre a maior alocação de vCPU, causado pelo aumento do tempo de resposta do SGBD. Isto é causado devido ao SGBD utilizar parte do processamento para criação de *caches* e *buffers* internos e tratamento das conexões dos clientes, causando um impacto no tempo de resposta.

Durante o restante da execução, o comportamento das abordagens do MoBINE e TR foram semelhantes, com enfase que o método MoBINE utilizou-se de $3,512 \pm 2,630$ núcleos, uma quantidade menor de vCPUs se comparado com o método TR, que utilizou de $4,799 \pm 3,220$. Entretanto, ao avaliarmos a velocidade do SGBD, o método TR obteve médias melhores, como o tempo de resposta de $1,513 \pm 0,753$ segundos, mas não muito distantes se comparados ao do MoBINE com $1,979 \pm 0,872$ segundos.



(a) Tempo de resposta e o provisionamento de vCPU da carga

(b) Uso de CPU e Provisionamento de vCPU

Figura 5.6: Tempo de resposta e uso de vCPU, com o provisionamento, do método uCPU, para carga Triangular.

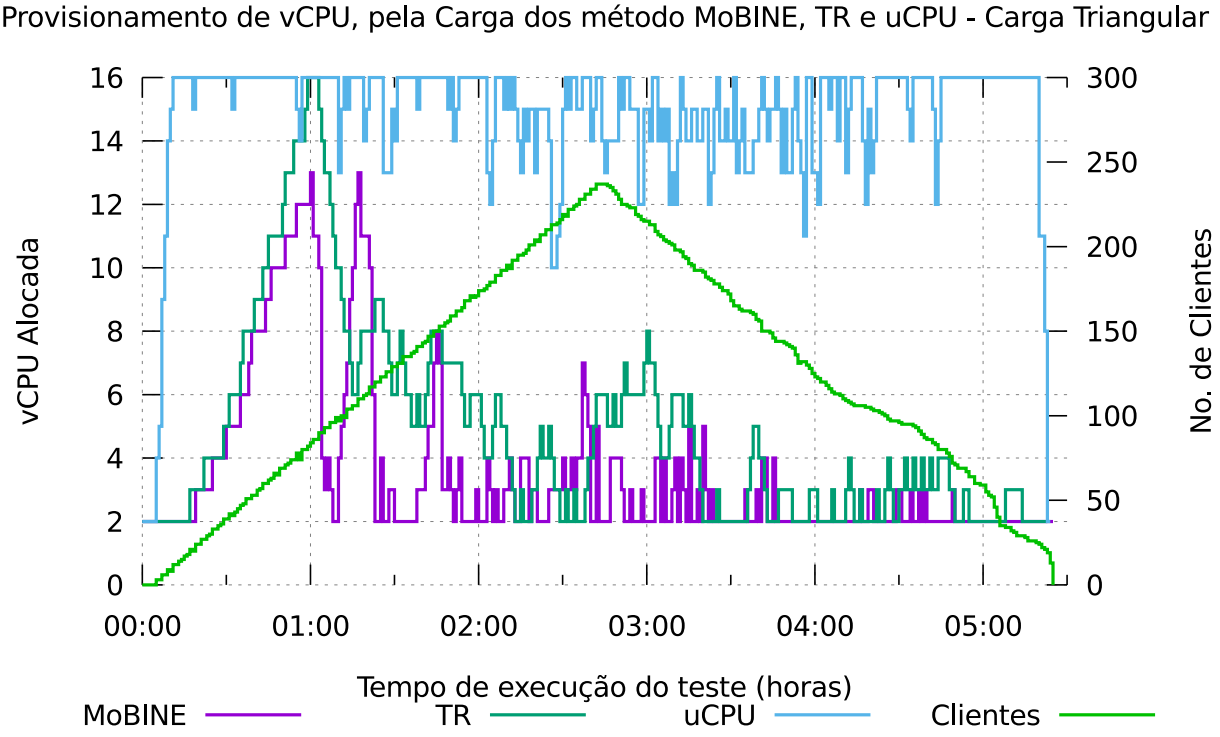


Figura 5.7: Provisionamento de vCPU pelos método MoBINE, TR e uCPU e do número de clientes durante o tempo de execução com carga Triangular.

5.3.2 Carga Intensa

Esta carga de trabalho foi projetada para verificar o comportamento dos métodos de provisionamento com carga constante, conforme mostrado na Figura 5.2(b). A execução é feita com aproximadamente oitenta clientes, definidos empiricamente, e a carga de trabalho é aplicada durante aproximadamente 25 minutos.

Utilizando o método MoBINE, podemos ver o comportamento do provisionador na Figura 5.8 e os resultados analíticos na Tabela 5.5. Pelo número de clientes solicitando conexões ser alto num curto espaço de tempo, o sistema operacional e SGBD priorizam as abertura das conexões, causando impacto no tempo de resposta das consultas, como mostra o pico de

5, 713 segundos no início do gráfico, o máximo encontrado neste teste. O tempo de resposta foi reduzindo conforme novos vCPUs foram alocados até o tempo de resposta atingir valores abaixo do intervalo objetivo, causando redução do número de vCPUs.

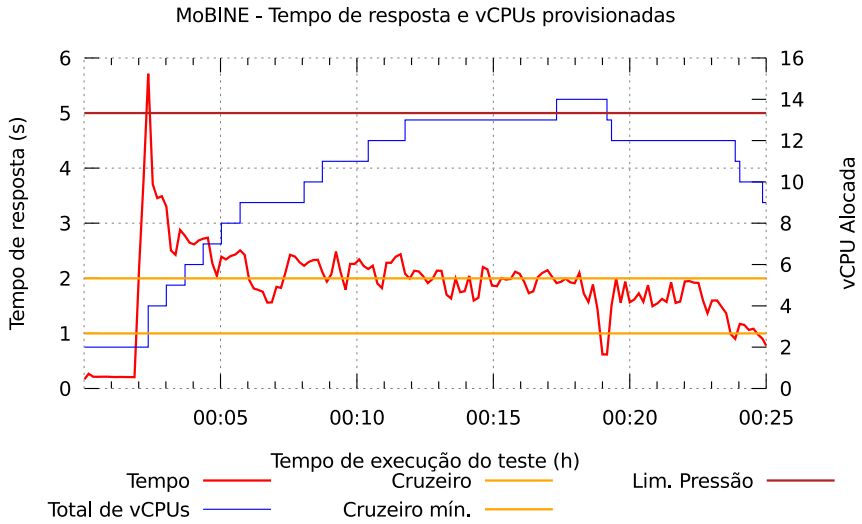


Figura 5.8: Tempo de resposta e número de vCPUs provisionadas pelo método MoBINE, para carga Intensa.

O SGBD respondeu com velocidade média de $1,654 \pm 0,784$ segundos, podendo ser considerada praticamente estável e com uma variância média da janela de $0,493 \pm 1,090$, indicando um SGBD estável. Foram provisionados em média $8,185 \pm 4,960$ núcleos de forma gradual durante toda a execução.

Tabela 5.5: Resultados analíticos do método MoBINE com a carga de teste Intensa.

MoBINE	**Média**	**Mínimo**	**Máximo**	**Desvio Padrão**
Tempo	1,654	0,158	5,713	0,784
Variância	0.4930	0,000	11,2900	1,090
Total de CPU	8,185	2,000	14,000	4,960

Com o método TR, obtemos o comportamento mostrado na Figura 5.9, em que o tempo de resposta oscilou entre $0,202$ e $4,890$ segundos, com média de $1,425 \pm 0,899$. Estes valores são muito próximos ao obtido pelo MoBINE, entretanto os vCPUs foram provisionados de forma distinta, utilizando no máximo 4 núcleos, contra 14 do MoBINE e obteve valores melhores de tempo de resposta.

Durante toda a execução, podemos notar que quando o tempo de resposta atinge valores acima do limite superior, novos núcleos são adicionados, tendo como reação a queda do tempo de resposta. Entretanto com valores inferiores ao mínimo, o provisionador remove estes novos núcleos e ocorre um novo aumento no tempo de resposta. Com estes aumentos serem acima do limite novamente, novos vCPUs são adicionados, causando uma oscilação maior no tempo de resposta do SGBD.

A Tabela 5.6 mostra o resultado analítico desta execução, que se obteve um consumo médio de $2,606 \pm 0,653$ núcleos, uma quantidade inferior ao provisionado pelo modelo MoBINE,

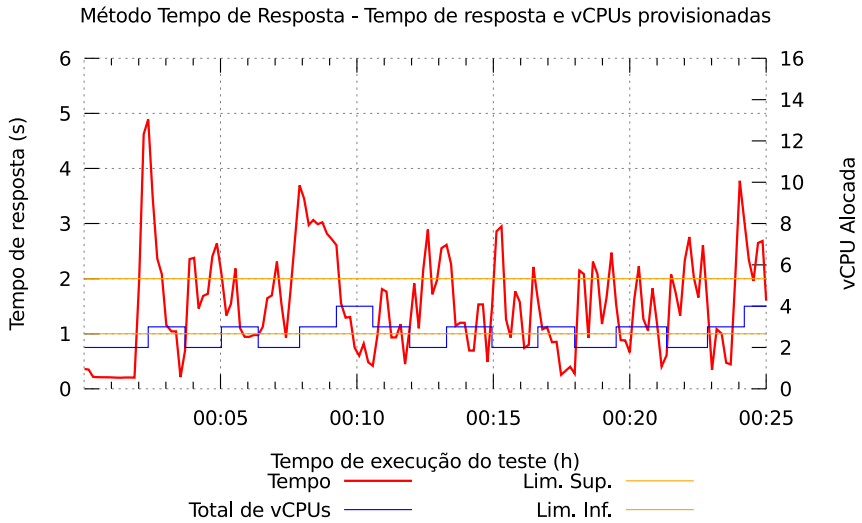


Figura 5.9: Tempo de resposta e número de vCPUs provisionados pelo método TR, para carga Intensa.

com um menor tempo de resposta, mas com uma pequena instabilidade maior para os clientes, mostrado pela variância da janela, que ficou em média $1,521 \pm 1,559$.

Tabela 5.6: Resultados analíticos do método TR com a carga Intensa.

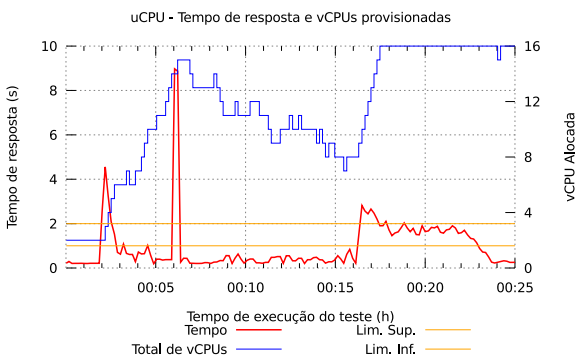
Tempo de Resposta	**Média**	**Mínimo**	**Máximo**	**Desvio Padrão**
Tempo	1,425	0,202	4,890	0,899
Variância	1,521	0,000	7,320	1,559
Total de CPU	2,606	2,000	4,000	0,653

Com o método de uCPU temos um comportamento semelhante ao encontrado na carga Triangular, em que o uso de CPU não oscila diretamente conforme o número de clientes. A Figura 5.10(a) mostra o comportamento do provisionador, em que no início, quando o SGBD recebe as conexões dos clientes, o tempo de resposta aumenta semelhante aos outros métodos e o uso de CPU também, indicado pela Figura 5.10(c). Logo após os primeiros 5 minutos de teste, temos uma leitura de $8,974$ segundos no gráfico de tempo de resposta, em que os dados analisados durante o teste não foram suficientes para expressar o causador da sua existência.

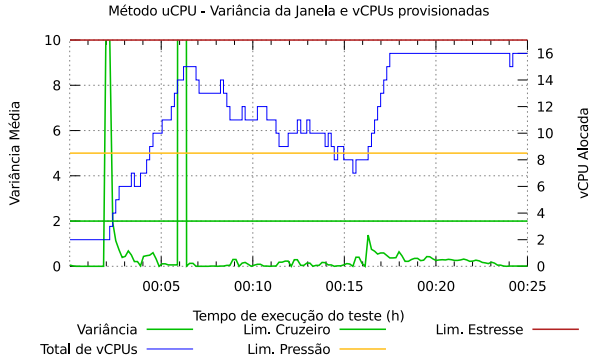
Até os 15 minutos, o comportamento geral do teste seguiu o semelhante ao encontrado na Carga Triangular, mantendo o tempo de resposta abaixo do intervalo. Após isso, seguido de algumas reduções de vCPUs, o tempo de resposta ultrapassou os 2 segundos de limite máximo, mesmo tendo sido alocados mais núcleos que o método TR. Após isso, mesmo adicionando novas vCPUs, o tempo permaneceu dentro do intervalo objetivo, só reduzindo ao final do teste.

Os resultados analíticos deste método estão na Tabela 5.7, em que novamente os uCPU apresentou-se como método que provisiona um menor tempo de reposta, com $0,939 \pm 1,206$ segundos e com o maior uso de vCPU, com média de $11,409 \pm 7,360$ núcleos. Entretanto, nesta execução apresentou-se como o método mais instável, com variância média de $2,574 \pm 19,497$ segundos, com uma máxima de $172,149$ segundos devido ao pico anômalo ocorrido.

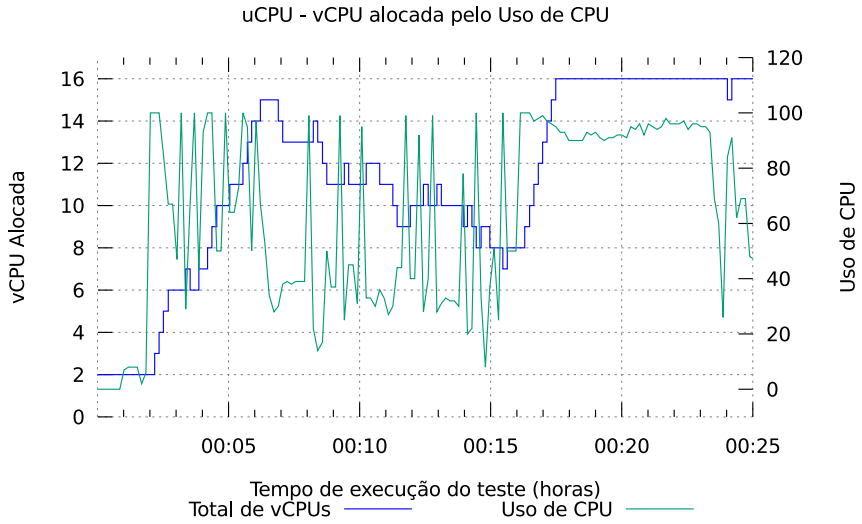
Na Figura 5.11 apresenta o provisionamento de vCPU dos três métodos analisados. MoBINE provisionou o número de vCPUs de forma gradual, enquanto uCPU aumentou de forma



(a) Tempo de resposta e número de vCPUs provisionadas pelo método uCPU.



(b) Variância do tempo de resposta e Provisionamento de vCPU



(c) Uso de CPU e Provisionamento de vCPU

Figura 5.10: Tempos de resposta e uso de CPU do método uCPU, para carga Intensa.

Tabela 5.7: Resultados analíticos do método uCPU com a carga de teste Intensa.

uCPU	Média	Mínimo	Máximo	Desvio Padrão
Tempo	0,939	0,195	8,974	1,206
Variância	2,574	0,000	172,149	19,497
Total de CPU	11,409	2,000	16,000	4,360

intensa e o método TR oscilou em sua maior parte do tempo entre duas e três unidades, apenas em um momento houve quatro unidades alocadas. O uCPU apresentou um comportamento conflitante, pois no início do teste houve a adição de vCPUs e após alguns minutos foram removidas e em seguida novamente alocadas até o máximo permitido, com a carga do sistema constante. Após o aumento não houve reduções até o final da execução do teste.

O método TR apresentou o melhor comportamento perante as abordagens, visto que utilizou a menor quantidade de vCPUs e manteve o tempo de resposta dentro do intervalo determinado, oscilando pouco o número de vCPUs, embora oscilando mais que o MoBINE.

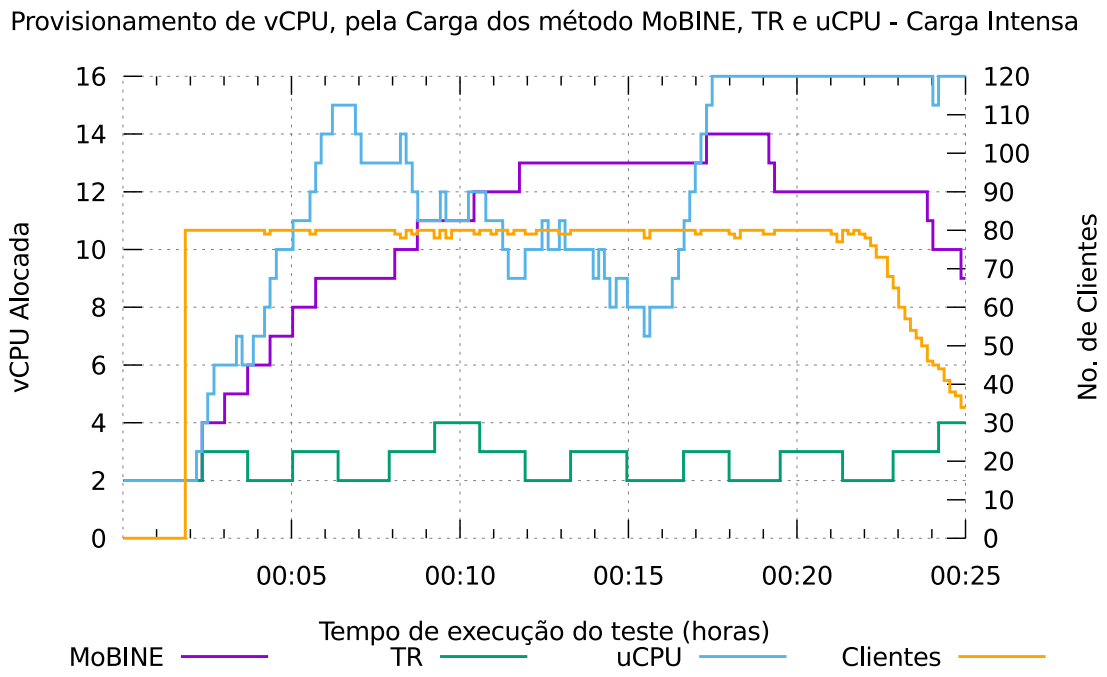


Figura 5.11: Provisionamento de vCPU pelos método MoBINE, TR e uCPU e do número de clientes durante o tempo de execução com carga Intensa.

5.3.3 Carga Oscilante

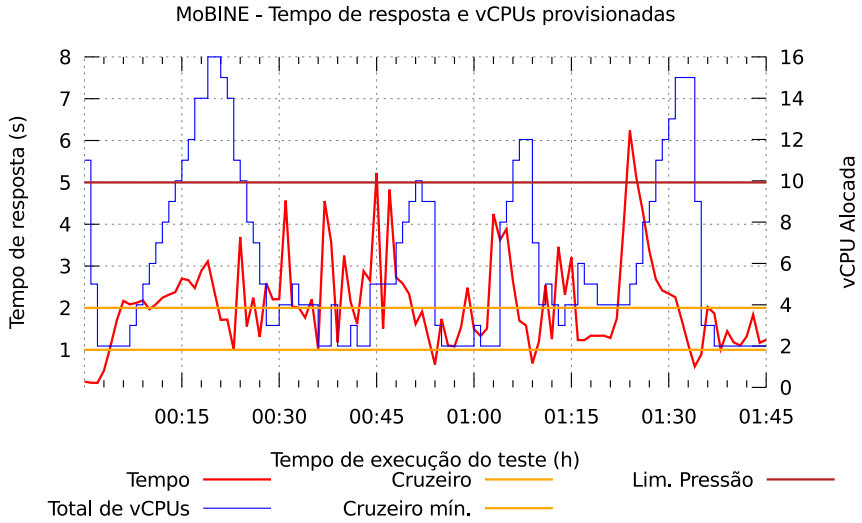
A carga oscilante foi definida para validar a aplicação das diferentes técnicas de provisionamento em um ambiente de produção, com o número de clientes variando ao longo do tempo, como mostra a Figura 5.2(c). O número máximo de clientes foi de 156 conectados simultaneamente e durante o período oscilatório, houve uma mínima de 47 clientes, com uma média geral de 82 clientes conectados.

O método MoBINE apresentou-se condizente com a carga aplicada, oscilando o número de CPUs provisionada seguindo a tendência do número de clientes, conforme mostra a Figura 5.12(a), provisionando em média $5,819 \pm 4,101$ núcleos. A maior quantidade provisionada, foi a máxima permitida de 16 vCPUs, como indica a Tabela 5.8. O tempo de resposta médio foi de $2,059 \pm 1,150$ segundos, com a máxima de $6,240$ segundos, indicando que grande parte do tempo o SGBD foi classificado no estado Sob-Pressão, faltando recurso.

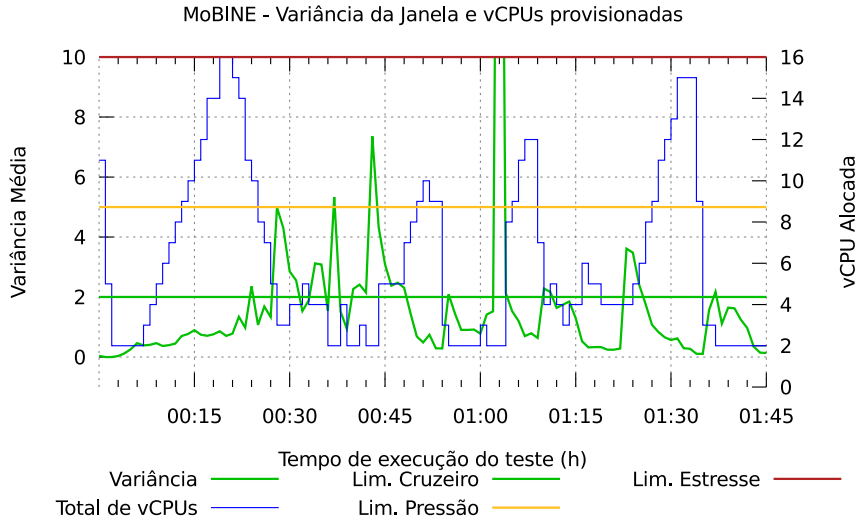
No quesito estabilidade, o SGBD apresentou uma variância média da janela de $1,570 \pm 3,132$ segundos, indicando um sistema estável em grande parte do tempo, sendo classificada como Cruzeiro. Entretanto, houve momentos em que ocorreram variações mais intensas, como na 01:01h de execução, em que foi atingido a variância máxima de $31,383$ segundos, sendo o SGBD classificado como Fora da especificação, adotando medidas drásticas, aumentando em duas unidades de vCPU.

Este método apresentou resultados próximos ao especificado, oscilando o provisionamento de vCPUs conforme a carga em que o SGBD foi submetido. O tempo de resposta, apesar da baixa variação, apresentou valores acima do esperado e em alguns momentos a classificação no estado de Sob-Pressão foi atingida, indicando que ocorreu atrasos no provisionamento de recursos.

O método TR apresentou um resultado semelhante ao MoBINE, provisionando o número de vCPUs na mesma tendência da demanda, conforme mostra a Figura 5.13(a). O tempo de resposta médio foi de $1,985 \pm 1,579$, indicando que o método, na maior parte do tempo, manteve



(a) Tempo de resposta e provisionamento de vCPU



(b) Variância do tempo de reposta e provisionamento de vCPU

Figura 5.12: Tempos de resposta, variância e número de vCPUs provisionadas pelo método MoBINE, para carga Oscilante.

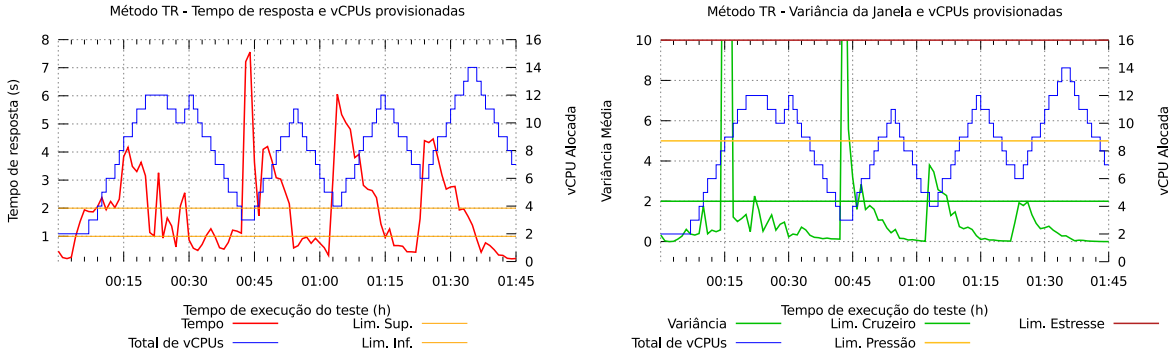
o SGBD respondendo dentro do intervalo especificado, ocorrendo algumas oscilações, como o máximo de 7, 559 segundos em 00:45h, como mostra a Tabela 5.9.

A variância da janela durante a execução do teste, apresentou um comportamento constante, com exceção de quatro picos de variação, um em cada momento em que a carga do sistema é aumentada, sendo o terceiro e o quarto de menor intensidade devido a maior quantidade de núcleos perante o aumento de carga anterior. A variância média obtida foi de $7,971 \pm 3,124$, com o máximo de 34, 435 segundos, presente no primeiro pico. Foram provisionados, em média, $7,971 \pm 3,124$ núcleos durante a execução deste teste, sendo o máximo provisionado de 14 núcleos.

O uCPU novamente obteve um tempo de resposta menor, comparado aos outros dois métodos, utilizando uma maior quantidade de vCPUs. O tempo médio obtido nesta execução foi de $0,370 \pm 0,317$ segundos, indicando que o SGBD responde com velocidade durante todo o teste, tendo como máximo 1, 934 segundos.

Tabela 5.8: Resultados analíticos do método MoBINE com carga Oscilante.

MoBINE	Média	Mínimo	Máximo	Desvio Padrão
Tempo	2,059	0,199	6,240	1,150
Variância	1,570	0,000	31,583	3,132
Total de CPU	5,819	2,000	16,000	4,101

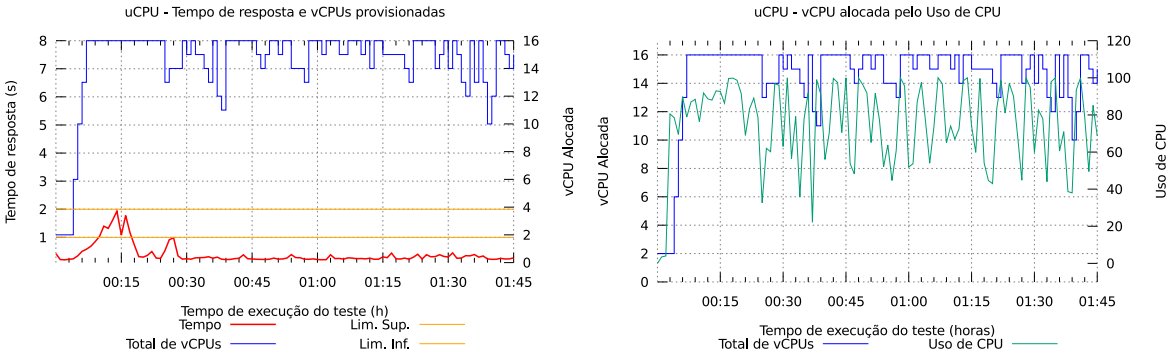


(a) Tempo de resposta e provisionamento de vCPU (b) Variância do tempo de reposta e provisionamento de vCPU

Figura 5.13: Tempos de resposta, variância e número de vCPUs provisionadas pelo método TR, para carga Oscilante.

Tabela 5.9: Resultado analítico do método TR, para carga Oscilante.

Tempo de Resposta	Média	Mínimo	Máximo	Desvio Padrão
Tempo	1,985	0,197	7,559	1,579
Variância	1,605	0,000	34,435	5,034
Total de CPU	7,971	2,000	14,000	3,124



(a) Tempo de resposta e provisionamento de vCPU (b) Variância do tempo de reposta e provisionamento de vCPU

Figura 5.14: Provisionamento de vCPU, tempo de resposta e variância do método uCPU, para carga Oscilante.

Com a variância, mostrada na Tabela 5.10, de $0,073 \pm 0,169$ indica que a aplicação se manteve estável durante toda a execução, com um pico máximo de $0,781$. Foram provisionados $14,573 \pm 2,816$ núcleos em média, indicando que na maior parte do tempo, o SGBD tinha disponível grande parte dos processadores, independente da carga que era lhe submetido, como mostra a Figura 5.14.

O provisionamento se manteve próximo ao máximo estabelecido, de 16 núcleos, devido ao uso de CPU, indicado na Figura 5.14(b), se manter em grande parte acima de 70%.A Tabela 5.10 mostra o resultado analítico dessa execução, em que podemos ver o uso de CPU médio de $72,833\% \pm 23,698\%$, que explica o provisionamento deste método.

Tabela 5.10: Resultado analítico do método uCPU, para carga Oscilante.

uCPU	**Média**	**Mínimo**	**Máximo**	**Desvio Padrão**
Tempo	0,370	0,200	1,934	0,317
Variância	0,07	0,000	0,781	0,169
Total de CPU	14,573	2,000	16,000	2,816
Uso de CPU	72,833	0,000	100,000	23,698

Na Figura 5.15 mostra o total de vCPUs provisionadas e o número de clientes ao longo da execução deste teste. uCPU apresenta, assim como nas outras cargas de trabalho, um aumento intenso logo no início da execução, mantendo acima de dez unidades alocadas.

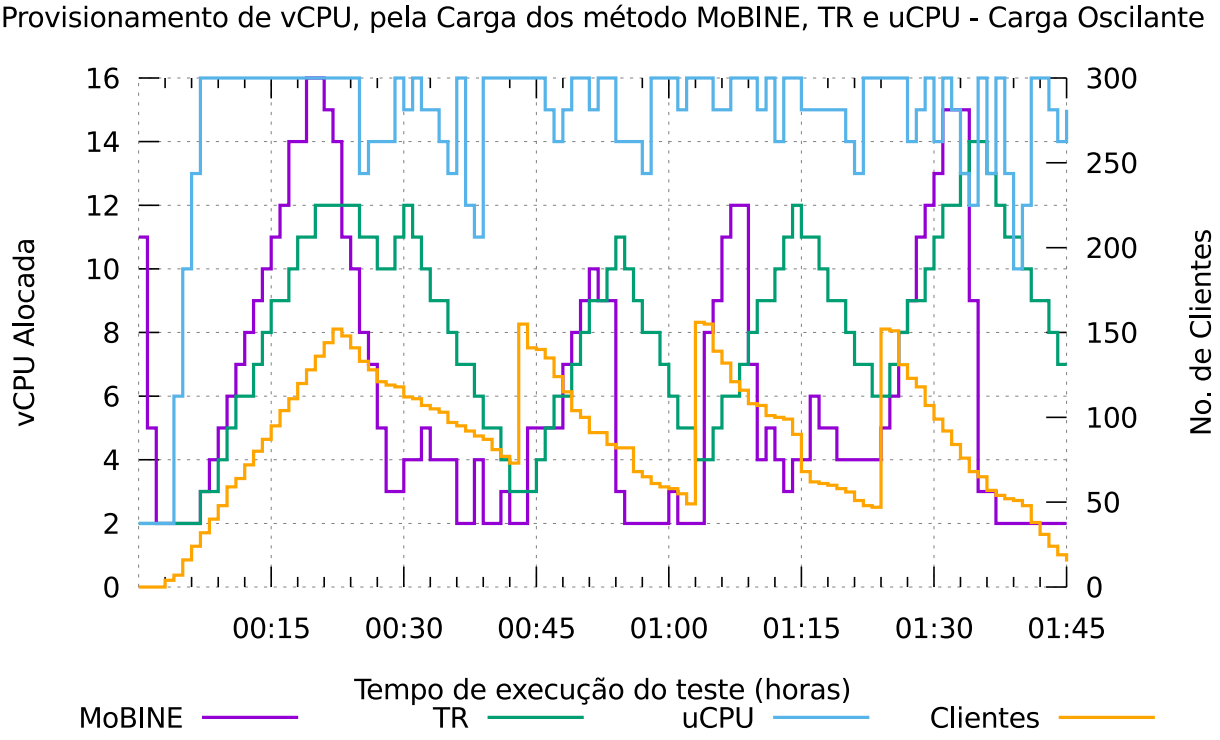


Figura 5.15: Provisionamento de vCPU pelos método MoBINE, TR e uCPU e do número de clientes durante o tempo de execução com carga Oscilante.

O MoBINE provisiona vCPUs conforme a carga de trabalho que é submetida, em que a cada aumento do número de clientes, aumenta-se a quantidade de vCPUs alocadas. Quando

ocorrem as reduções do número de clientes, o provisionamento retira vCPUs de modo mais agressivo que as outras abordagens, devido ao fato do tempo de resposta obter leituras inferiores ao intervalo.

O método TR oscila a quantidade de vCPUs seguindo as mesmas tendências que o MoBINE, entretanto as ações elásticas ocorrem com uma latência maior de provisionamento. Esta latência causa um aumento na variância e no prejudica o tempo de resposta, visto que quanto maior a latência de provisionamento, mais leituras com valores fora do especificado ocorrem, causando sintomas como lentidão ou instabilidade sentidas pelos clientes.

Desta forma, os três métodos apresentam comportamentos distintos perante cargas de trabalho diferentes. Foge do escopo deste trabalho, análises com diferentes configurações do SGBD, visto que este tipo de aplicação é sensível a alteração dos seus parâmetros internos, pelo fato de utilizar diferentes estratégias de otimização do uso da memória e dos *caches* disponível.

Alterações na configuração do SO ou MV também trazem impactos no desempenho do SGBD perante nestes tipos de ambientes, como realizar a pinagem de CPU[1], compartilhar ou dedicar discos rígidos ou memória elástica. Configurar estes parâmetros podem impactar de forma significativa o desempenho do SGBD, e consequentemente dos provisionadores baseados em variáveis temporais.

[1]Neste contexto, *pinagem* de CPU se traduz em dedicar núcleos de CPU físico para núcleos de CPU virtuais.

Capítulo 6

Considerações Finais

Este trabalho teve como objetivo apresentar uma nova abordagem para a exploração da elasticidade em aplicações cliente-servidor, com ênfase para sistemas gerenciadores de bancos de dados, sob a motivação de que os mecanismos presentes no estado-da-arte para este tipo de sistemas, apresentam abordagens diferentes e em outros contextos. Por isto, a exploração da elasticidade em nível vertical, para um sistema altamente influenciável por estes recursos é interessante visto os resultados obtidos.

Nesta proposta, a carga no sistema é estimada pelo tempo de resposta de um SGBD e representada em níveis de estresse utilizando a máquina de estados ENE. Com esta classificação e algumas informações sobre os recursos, ações de elasticidade podem ser realizadas para que o SGBD responda com rapidez e estabilidade, dentro do escopo especificado.

Esta máquina de estados ENE foi baseada nas definições da *Database State Machine* em que este identifica os estados de um SGBD baseado em métricas utilizando o total de transações por segundo (TPS) atendidas pelo SGBD e aquela utiliza do tempo de resposta de uma requisição (no contexto de banco de dados, uma requisição é transação). Com esta informação, cálculos estatísticos são realizados com o histórico recente, estabelecendo métricas principais de desempenho, e aliado a regras pré-definidas, o estado é inferido.

Uma aplicação pode ser representada, quando analisado seu desempenho perante os recursos computacionais utilizados, em seis estados: Aquecimento, Cruzeiro, Sob-Pressão, Estresse, Falha Iminente e Sobra de recursos. Em um ambiente elástico ideal, tendo como objetivo a alocação apenas de RCs necessários, uma aplicação permanece sempre no estado de Cruzeiro, que indica um ótimo gerenciamento de recursos, alocando e removendo-os conforme a demanda. Em ambientes reais de produção, oscilações bruscas na carga de trabalho, a existência da latência de detecção e de provisionamento, podem causar uma piora no desempenho, levando a aplicação para fora do especificado até que seja disponibilizado novos recursos.

Os resultados apresentados por este provisionador mostram que o controle de elasticidade baseado em níveis de estresse é uma alternativa promissora quando comparado a outros métodos que empregam métricas mais simples, como tempo de resposta e uso de recursos. Com a carga Triangular, este método apresentou o maior tempo de resposta e a maior variação, mas inferior ao máximo desejado, consumindo o menor número de recursos, indicando que neste cenário foi vantajoso esta abordagem.

Com a carga Intensa, o método TR apresentou o melhor desempenho no geral, pois o tempo de resposta ficou dentro dos limiares estabelecidos, consumindo a menor quantidade de recursos, e o MoBINE obteve valores semelhantes em desempenho, mas consumindo o três vezes mais recursos. No cenário de carga oscilante, tanto o MoBINE quanto TR ficaram próximos ao desempenho esperado, mas o MoBINE apresentou em média um menor consumo de recursos,

com a menor variação. Dentre todos os testes, o método uCPU apresentou o melhor desempenho, mas consumindo o máximo de recursos possível, sendo uma estratégia não recomendada para este tipo de aplicação com este tipo de carga.

Para melhorar este modelo, é necessário que seja melhor ampliado os estudos no consumo de recursos por um SGBD, comparando diferentes sondas de trabalho e configurações do provisionador. O *benchmark* TPC-H foi usado neste trabalho por utilizar-se de consultas de alto consumo de processamento e memória, para aproveitar a utilização da elasticidade vertical. Utilizar outros *benchmarks* foram avaliados, mas descartados devido ao seu objetivo não estressar diretamente estes recursos. O TPC-B, por exemplo, utiliza de simulação de transações bancárias, possuindo uma transação que altera alguns dados presentes no SGBD, que multiplicada pelo total de clientes, criam condições *race-condition*, em que acabam por não utilizar todos os recursos disponíveis na máquina, mas perdem o desempenho devido a alta concorrência.

Por isto, este modelo é capaz de provisionar recursos corretamente, desde que a carga de trabalho e as sondas envidas sejam coerentes com o que é usado pelos clientes, pois ao diferenciar as consultas, os impactos que uma causam as outras é diferentes. A configuração da aplicação também pode interferir no bom provisionamento, visto que um SGBD é totalmente dependente dos recursos alocados e da configuração interna, principalmente na memória utilizada ao se processar diferentes consultas.

Por isto, a abordagem proposta neste trabalho pode ser empregada em Banco de Dados como Serviço (DBaaS), pois do ponto de vista do provedor, a elasticidade garante um melhor uso dos recursos de computação, fornecendo economia de escala e permitindo que mais usuários possam ser atendidos simultaneamente, uma vez que os recursos liberados por um usuário podem instantaneamente ser alocados por outro. Da perspectiva do usuário, a elasticidade garante que o desempenho do banco de dados esteja sempre conforme o especificado no acordo de nível de serviço, independentemente da carga de trabalho recebida.

Referências Bibliográficas

[1] RIGHI, R. d. R. Elasticidade em cloud computing: conceito, estado da arte e novos desafios. *Revista Brasileira de Computação Aplicada*, v. 5, n. 2, p. 2–17, 2013.

[2] MEIRA, J. A.; ALMEIDA, E. C. D.; TRAON, Y. L. A State Machine for Database Non-functional Testing. In: *IDEAS 2014 : 18th International Database Engineering & Applications Symposium*. Porto, Portugal: [s.n.], 2014.

[3] VAQUERO, L. M. et al. A break in the clouds: Towards a cloud definition. *SIGCOMM Computer Communication Review*, v. 39, n. 1, p. 50–55, 2008.

[4] GALANTE, G.; BONA, L. C. E. d. A survey on cloud computing elasticity. In: *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*. [S.l.: s.n.], 2012.

[5] MAO, M.; HUMPHREY, M. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. New York, NY, USA: [s.n.], 2011.

[6] HAN, R. et al. Lightweight resource scaling for cloud applications. In: *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*. [S.l.: s.n.], 2012. p. 644–651.

[7] DUTREILH, X. et al. From data center resource allocation to control theory and back. In: *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. [S.l.: s.n.], 2010.

[8] ELMASRI, R. A.; NAVATHE, S. B. *Fundamentals of Database Systems*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 1999.

[9] ELMORE, A. et al. Towards an elastic and autonomic multitenant database. In: *NetDB 2011 - 6th International Workshop on Networking Meets Databases Co-located with SIGMOD 2011*. [S.l.: s.n.], 2011.

[10] MINHAS, U. F. et al. Elastic scale-out for partition-based database systems. In: *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering Workshops*. Washington, DC, USA: IEEE Computer Society, 2012. (ICDEW '12), p. 281–288. ISBN 978-0-7695-4748-0.

[11] DAS, S.; AGRAWAL, D.; ABBADI, A. E. Elastras: An elastic transactional data store in the cloud. In: *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*. Berkeley, CA, USA: USENIX Association, 2009. (HotCloud'09). Disponível em: <http://dl.acm.org/citation.cfm?id=1855533.1855540>.

[12] ELMORE, A. J. et al. Zephyr: Live migration in shared nothing databases for elastic cloud platforms. In: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: ACM, 2011. (SIGMOD '11), p. 301–312.

[13] SAKR, S. et al. CloudDB AutoAdmin: Towards a truly elastic cloud-based data store. In: Ian Foster, Louise Moser, Jia Zhang (Ed.). *Proceedings of the IEEE 9th International Conference on Web Services (ICWS '11)*. Washington DC, USA: IEEE Computer Society, 2011. p. 732–733.

[14] DAS, S. et al. Albatross: Lightweight elasticity in shared storage databases for the cloud using live data migration. *Proc. VLDB Endow.*, VLDB Endowment, v. 4, n. 8, p. 494–505, maio 2011. ISSN 2150-8097. Disponível em: <http://dx.doi.org/10.14778/2002974.2002977>.

[15] CURINO, C. et al. Relational Cloud: A Database Service for the Cloud. In: *5th Biennial Conference on Innovative Data Systems Research*. Asilomar, CA: [s.n.], 2011.

[16] SERAFINI, M. et al. Accordion: Elastic scalability for database systems supporting distributed transactions. *Proc. VLDB Endow.*, VLDB Endowment, v. 7, n. 12, p. 1035–1046, ago. 2014. ISSN 2150-8097. Disponível em: <http://dx.doi.org/10.14778/2732977.2732979>.

[17] GALANTE, G.; BONA, L. C. E. A survey on cloud computing elasticity. In: *Proceedings of the International Workshop on Clouds and eScience Applications Management*. [S.l.]: IEEE, 2012. (CloudAM'12), p. 263–270.

[18] SULLIVAN, D. *Conversation with Eric Schmidt hosted by Danny Sullivan*. 2006. `http://www.google.com/press/podium/ses2006.html`. Acessado em julho de 2014.

[19] AMAZON EC2. 2014. `http://aws.amazon.com/pt/ec2/`. Acessado em julho de 2014.

[20] GALANTE, G. *Explorando a Elasticidade em nível de programação no desenvolvimento e na execução de aplicações científicas*. Tese (Doutorado) — Universidade Federal do Paraná, 2014.

[21] HERBST, N. R.; KOUNEV, S.; REUSSNER, R. Elasticity in cloud computing: What it is, and what it is not. In: *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*. [S.l.: s.n.], 2013.

[22] VIRTUALIZATION and Cloud Computing. 2013. `http://www.intel.com/content/dam/www/public/us/en/documents/guides/cloud-computing-virtualization-building-private-iaas-guide.pdf`. Acessado em novembro de 2014.

[23] MELL, P.; GRANCE, T. The NIST definition of cloud computing (draft). In: . [s.n.], 2010. Disponível em: <http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_cloud-definition.pdf>.

[24] KAUFMAN, C.; VENKATAPATHY, R. *Windows Azure Security Overview*. 2014. `http://go.microsoft.com/?linkid=9740388`. Acessado em agosto de 2014.

[25] GOOGLE Apps for Business. 2015. `https://apps.google.com/`. Acessado em agosto de 2015.

[26] DROPBOX. 2014. https://www.dropbox.com. Acessado em agosto de 2014.

[27] OWENS, D. Securing elasticity in the cloud. *Queue*, v. 8, n. 5, p. 10:10–10:16, 2010.

[28] ARMBRUST, M. et al. A view of cloud computing. *Communications ACM*, v. 53, n. 4, 2010.

[29] VAQUERO, L. M.; RODERO-MERINO, L.; BUYYA, R. Dynamically scaling applications in the cloud. *SIGCOMM Computer Communication Review*, v. 41, n. 1, p. 45–52, 2011.

[30] MINHAS, U. F. et al. Elastic scale-out for partition-based database systems. In: *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering Workshops*. [S.l.: s.n.], 2012.

[31] LORIDO-BOTRAN, T.; MIGUEL-ALONSO, J.; LOZANO, J. A. A review of auto-scaling techniques for elastic applications in cloud environments. *J. Grid Comput*, v. 12, n. 4, p. 559–592, 2014.

[32] URGAONKAR, B. et al. Agile dynamic provisioning of multi-tier internet applications. *ACM Transactions on Autonomous and Adaptive Systems*, v. 3, n. 1, p. 1:1–1:39, 2008.

[33] MOORE, L. R.; BEAN, K.; ELLAHI, T. Transforming reactive auto-scaling into proactive auto-scaling. In: *Proceedings of the 3rd International Workshop on Cloud Data and Platforms*. [S.l.: s.n.], 2013.

[34] HEINZE, T. et al. Auto-scaling techniques for elastic data stream processing. In: *Data Engineering Workshops (ICDEW), 2014 IEEE 30th International Conference on*. [S.l.: s.n.], 2014.

[35] ASSUNCAO, M. D. de; COSTANZO, A. di; BUYYA, R. Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In: *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing*. [S.l.: s.n.], 2009.

[36] MI, H. et al. Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers. In: *Services Computing (SCC), 2010 IEEE International Conference on*. [S.l.: s.n.], 2010.

[37] POSTGRESQL. 2014. http://www.postgresql.org. Acessado em outubro de 2014.

[38] MYSQL Database. 2014. http://www.mysql.com/products/. Acessado em outubro de 2014.

[39] ORACLE Database. 2014. https://www.oracle.com/database/index.html. Acessado em outubro de 2014.

[40] GORDON, K. What is big data? *ITNOW*, v. 55, n. 3, p. 12–13, 2013. Disponível em: <http://itnow.oxfordjournals.org/content/55/3/12.abstract>.

[41] HBASE. 2014. http://hbase.apache.org/. Acessado em outubro de 2014.

[42] HYPERTABLE. 2014. http://hypertable.org/. Acessado em outubro de 2014.

[43] CATTELL, R. Scalable sql and nosql data stores. *SIGMOD Record*, v. 39, n. 4, p. 12–27, 2011.

[44] NAHEMAN, W.; WEI, J. Review of nosql databases and performance testing on hbase. In: *Mechatronic Sciences, Electric Engineering and Computer (MEC), Proceedings 2013 International Conference on*. [S.l.: s.n.], 2013.

[45] ASLETT, M. *How will the database incumbents respond to NoSQL and NewSQL?* 2014. `http://cs.brown.edu/courses/cs227/archives/2012/papers/newsql/aslett-newsql.pdf`. Acessado em outubro de 2014.

[46] DEWITT, D. J.; LEVINE, C. Not just correct, but correct and fast: a look at one of jim gray's contributions to database system performance. *SIGMOD Record*, v. 37, n. 2, p. 45–49, 2008. Disponível em: <http://dblp.uni-trier.de/db/journals/sigmod/sigmod37.html>.

[47] TPC. 2014. `http://www.tpc.org/`. Acessado em outubro de 2014.

[48] MEIRA, J. A. *MODEL-BASED STRESS TESTING FOR DATABASE SYSTEMS*. Tese (Doutorado) — Universidade Federal do Paraná and Universidade de Luxemburgo, 2014.

[49] SILBERSCHATZ, A.; KORTH, H.; SUDARSHAN, S. *Sistema de banco de dados*. [S.l.]: CAMPUS - RJ, 2006.

[50] HAERDER, T.; REUTER, A. Principles of transaction-oriented database recovery. *ACM Computing Surveys*, v. 15, n. 4, p. 287–317, 1983.

[51] HAN, J. et al. Survey on nosql database. In: *Pervasive Computing and Applications (ICPCA), 2011 6th International Conference on*. [S.l.: s.n.], 2011.

[52] LI, Y.; MANOHARAN, S. A performance comparison of sql and nosql databases. In: *Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference on*. [S.l.: s.n.], 2013.

[53] STONEBRAKER, M. New opportunities for newsql. *Communications ACM*, ACM, v. 55, n. 11, p. 10–11, 2012.

[54] GILBERT, S.; LYNCH, N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, v. 33, n. 2, p. 51–59, 2002.

[55] BREWER, E. Cap twelve years later: How the "rules"have changed. *Computer*, v. 45, n. 2, p. 23–29, 2012.

[56] ABADI, D. Consistency tradeoffs in modern distributed database system design: Cap is only part of the story. *Computer*, IEEE Computer Society Press, v. 45, n. 2, 2012.

[57] TUDORICA, B.; BUCUR, C. A comparison between several nosql databases with comments and notes. In: *Roedunet International Conference (RoEduNet), 2011 10th*. [S.l.: s.n.], 2011.

[58] MONGODB. 2014. `http://www.mongodb.org/`. Acessado em outubro de 2014.

[59] CASSANDRA. 2014. `http://cassandra.apache.org/`. Acessado em outubro de 2014.

[60] MEMCACHEDB. *MemCacheDB*. 2014. `http://memcachedb.org/`. Acessado em 20 out. 2014.

[61] COUCHDB. 2014. `http://couchdb.apache.org/`. Acessado em outubro de 2014.

[62] LEAVITT, N. Will nosql databases live up to their promise? *Computer*, 2010.

[63] HECHT, R.; JABLONSKI, S. Nosql evaluation. In: *International Conference on Cloud and Service Computing*. [S.l.: s.n.], 2011.

[64] VIJAYKUMAR, S.; SARAVANAKUMAR, S. Implementation of nosql for robotics. In: *Emerging Trends in Robotics and Communication Technologies (INTERACT), 2010 International Conference on*. [S.l.: s.n.], 2010.

[65] NUODB. 2014. `http://www.nuodb.com/`. Acessado em outubro de 2014.

[66] VOLTDB. 2015. `http://voltdb.com/`. Acessado em janeiro de 2015.

[67] HSTORE. 2015. `http://hstore.cs.brown.edu/`. Acessado em janeiro de 2015.

[68] JAIN, R. K. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley, 1991. Disponível em: <http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0471503363>.

[69] LIMA, M. R. d.; SUNYE, M. S. et al. *Execução distribuída de benchmarks em sistemas de bancos de dados relacionados*. Dissertação (Mestrado) — Universidade Federal do Paraná, 2009.

[70] MEIRA, J. A. *Uma metodologia incremental de teste de estresse de Banco de Dados Transacional de grande escala*. Dissertação (Mestrado) — Universidade Federal do Paraná, 2010.

[71] LIU, H. et al. Hotplug or ballooning: A comparative study on dynamic memory management techniques for virtual machines. *IEEE Trans. Parallel Distrib. Syst.*, v. 26, n. 5, p. 1350–1363, 2015.

[72] GALANTE, G.; BONA, L. C. E. A programming-level approach for elasticizing parallel scientific applications. *J. Syst. Softw.*, Elsevier Science Inc., New York, NY, USA, v. 110, n. C, p. 239–252, dez. 2015. ISSN 0164-1212.

[73] XEN Project. 2014. `http://www.xenproject.org/`. Acessado em agosto de 2014.