UNIVERSIDADE FEDERAL DO CEARÁ
DEPARTAMENTO DE COMPUTAÇÃO
MESTRADO E DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

VICTOR AGUIAR EVANGELISTA DE FARIAS

UMA ABORDAGEM PARA A MODELAGEM DE DESEMPENHO E DE ELASTICIDADE PARA BANCOS DE DADOS EM NUVEM

FORTALEZA, CEARÁ

2016

VICTOR AGUIAR EVANGELISTA DE FARIAS

UMA ABORDAGEM PARA A MODELAGEM DE DESEMPENHO E DE ELASTICIDADE PARA BANCOS DE DADOS EM NUVEM

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Área de concentração: Banco de Dados

Orientador: Prof. Dr. Javam de Castro Machado

Co-Orientador: Prof. Dr. Flávio Rubens de Carvalho Sousa

FORTALEZA, CEARÁ

2016

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca de Ciências e Tecnologia

F238a Farias, Victor Aguiar Evangelista de.
Uma abordagem para a modelagem de desempenho e de elasticidade para bancos de dados em nuvem / Victor Aguiar Evangelista de Farias. – 2016.
37 p.: il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Departamento de Computação, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2016.
Área de Concentração: Banco de dados.
Orientação: Prof. Dr. Javam de Castro Machado.
Coorientação: Prof. Dr. Flávio Rubens de Carvalho Sousa.

1. Computação em nuvem. 2. Modelagem computacional. 3. Banco de dados. I. Título.

CDD 005

VICTOR AGUIAR EVANGELISTA DE FARIAS

UMA ABORDAGEM PARA A MODELAGEM DE DESEMPENHO E DE ELASTICIDADE PARA BANCOS DE DADOS EM NUVEM

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação. Área de concentração: Banco de Dados

Aprovada em: __/__/____

BANCA EXAMINADORA

Prof. Dr. Javam de Castro Machado
Universidade Federal do Ceará - UFC
Orientador

Prof. Dr. Flávio Rubens de Carvalho Sousa
(Co-Orientador)

Prof. Dr. José Maria da Silva Monteiro Filho
Universidade Federal do Ceará - UFC

Prof. Dr. Daniel Cardoso Moraes de Oliveira
Universidade Federal Fluminense - UFF

Aos Meus Pais.

AGRADECIMENTOS

Aos meu pais, Francisco Aguiar e Clara Maria, pelo apoio incondicional dado durante todas as etapas da minha vida possibilitando alcançar esta vitória.

Ao meu orientador e mentor, Javam Machado, pelos conselhos, orientações e oportunidades concedidas que foram fundamentais para o sucesso desta dissertação.

Ao meu coorientador Flávio Sousa, pelo acompanhamento, considerações, paciência e companheirismo ao longo deste trabalho.

Ao professor Gilvan Maia pelo acompanhamento próximo e proativo desde o começo da minha jornada científica.

Aos profs. João Paulo Pordeus e Leonardo Moreira pelas contribuições dadas durante o curso deste trabalho.

Aos profs. Daniel de Oliveira e José Maria Monteiro pela disponibilidade de participar da minha banca de mestrado e pelas valiosas sugestões no aprimoramento desta dissertação.

À minha namorada, Lara Campos, pela compreensão, paciência e apoio nos momentos mais delicados.

Ao LSBD por ter fornecido uma estrutura adequada para o desenvolvimento da minha dissertação e apoio financeiro necessário para eventos científicos.

Aos amigos colaboradores do ARIDa e do LSBD por todos os momentos compartilhados, por todas as contribuições e por todo o apoio durante esse período.

RESUMO

A computação em nuvem é um paradigma de computação emergente e bem sucedido que oferece serviços por demanda. Com o crescimento exponencial da quantidade de dados utilizados pelas aplicações atuais, os bancos de dados NoSQL, que são sistemas inerentemente distribuídos, têm sido usados para gerenciar dados na Nuvem. Nesse cenário, é fundamental que os provedores de serviços em nuvem garantam a Qualidade de Serviço (QoS) por meio do cumprimento do contrato *Service Level Agreement* (SLA) enquanto reduz os custos operacionais relacionados a *overprovisioning* e *underprovisioning*. Mecanismos de QoS podem se beneficiar fortemente de modelos de desempenho preditivos que estimam o desempenho para uma dada configuração do sistema NoSQL e da carga de trabalho. Com isso, estratégias de elasticidade podem aproveitar esses modelos preditivos para fornecer meios de adicionar e remover recursos computacionais de forma mais confiável. Este trabalho apresenta uma abordagem para modelagem de desempenho genérica para banco de dados NoSQL em termos de métricas de desempenho baseadas no SLA capaz de capturar o efeitos não-lineares causados pelo aspectos de concorrência e distribuição. Adicionalmente, é apresentado um mecanismo de elasticidade para adicionar e remover nós sistema NoSQL baseado em modelos de desempenho. Resultados de avaliação experimental confirmam que a modelagem de desempenho estima as métricas de forma acurada para vários cenários de carga de trabalho e configurações do sistema. Por fim, a nossa estratégia de elasticidade é capaz de garantir a QoS enquanto utiliza os recursos de forma eficiente.

Keywords: Computação em nuvem, modelagem de desempenho, elasticidade, NoSQL

ABSTRACT

Cloud computing is a successful, emerging paradigm that supports on-demand services. With the exponential growth of data generated by present applications, NoSQL databases which are inherently distributed systems have been used to manage data in the cloud. In this scenario, it is fundamental for cloud providers to guarantee Quality of Service (QoS) by satisfying tho Service Level Agreement (SLA) contract while reducing the operational costs related to both overprovisioning and underprovisioning. Thus QoS mechanisms can greatly benefit from a predictive model that estimates SLA-based performance metrics for a given cluster and workload configuration. Therewith, elastic provisioning strategies can benefit from these predictive models to provide a reliable mechanism to add and remove resources reliably. In this work, we present a generic performance modeling for NoSQL databases in terms of SLA-based metrics capable of capturing non-linear effects caused by concurrency and distribution aspects. Moreover we present a elastic provisioning mechanism based on performance models. Results of experimental evaluation confirm that our performance modeling can accurately estimate the performance under a wide range of workload configurations and also that our elastic provisioning approach can ensure QoS while using resources efficiently.

Keywords: Cloud Computing. perfomance modeling. elasticity. NoSQL

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

A computação em nuvem é um paradigma para computação orientada a serviços que tem experimentado notável sucesso e aderência. Disponibilidade, cobrança baseada no uso e escalabilidade são as maiores razões para tal êxito. Os sistemas de *software* são artefatos de computador que operam sobre dados, resultando em modificações ou em novo dados. Portanto, nota-se que, como a maioria das aplicações em nuvem é orientada a dados, os sistemas de gerenciamento de banco de dados (SGBD) que proveem serviços de dados para essas aplicações tornam-se componentes tipicamente críticos na pilha de *software* em nuvem (ELMORE et al., 2011).

Nesse ambiente, os sistemas de bancos de dados NoSQL aparecem como alternativa em relação aos sistemas relacionais tradicionais. Esses sistemas são horizontalmente escaláveis, distribuídos e armazenam dados de forma não-relacional. Sistemas NoSQL são capazes de armazenar e indexar grandes conjuntos de dados e, em geral, sem esquema fixo, servindo uma grande quantidade de requisições concorrentes. Nesses sistemas, a escalabilidade é possível principalmente por meio de dois mecanismos: fragmentação, ou seja, o particionamento horizontal sobre arquiteturas que não compartilham dados (e.i. *shared-nothing*) para reduzir gargalos e aumentar a disponibilidade; e replicação, ou seja, a cópia completa ou parcial de dados para aumentar a capacidade de leitura. Usando esses mecanismos, as implementações de SGBDs NoSQL podem melhorar seu desempenho ao ajustar a quantidade de recursos alocados em função da quantidade de requisições, tipo de consulta e latência da rede (CRUZ et al., 2013).

Usuários de serviços em nuvem esperam que os provedores de nuvem assegurem a qualidade de serviço (*Quality of Service*, QoS) usando contratos denominados de acordo de nível de serviço (*Service Level Agreement*, SLA). Tais contratos baseiam-se principalmente no cumprimento de determinados aspectos de desempenho e de disponibilidade inerentes às aplicações. Todavia, os provedores geralmente baseiam o SLA apenas na disponibilidade dos serviços. Entretanto, é crucial que os provedores também ofereçam QoS que considerem efetivamente os aspectos de desempenho a fim de melhorar a experiência do usuário final das aplicações. Assim, ao se propor mecanismos de QoS, administradores de SGBDs se deparam com questões como:

- *Planejamento de capacidade.* A capacidade atual do sistema é suficiente para garantir a QoS para a carga de trabalho corrente? Deve-se adicionar ou remover capacidade para atender o SLA ou economizar custos de infraestrutura?
- *Diagnóstico de Carga de Trabalho.* Quais classes de requisições da carga de trabalho degradam mais o desempenho? Que pares de classes tem a interação mais conflituosa por contenção de dados? e
- *Avaliação das decisões de projeto.* Como identificar se o modelo de distribuição, modelo de dados, particionamento dos dados e e outras configurações do banco são adequadas para a carga de trabalho que é submetida nele?

Nesse contexto, uma modelagem da desempenho pode responder esses tipos de questões. Os SGBDs em nuvem integram ambientes de processamento de dados que executam concorrentemente cargas de trabalhos heterogêneas. Por isso, é importante que uma abordagem de modelagem do desempenho tenha a habilidade de estimar os impactos de execução concorrentes de requisições em um carga de trabalho em evolução ao longo do tempo (DUGGAN et al., 2011). O desempenho de um SGBD distribuído pode não ser linearmente correlacionado com a quantidade de recursos alocados. Trabalhos teóricos mostram que aspectos de concorrência e a distribuição podem degradar o desempenho de modo não-linear o que configura um desafio para abordagens de modelagem do desempenho (GRAY et al., 1996).

Esse problema tem sido amplamente atacado para SGBDs relacionais centralizados (DUGGAN et al., 2011; MOZAFARI et al., 2013). Todavia, tem recebido pouca atenção no contexto de SGBDs NoSQL em nuvem. Nesse contexto, algoritmos de aprendizado de máquina permitem criar modelo preditivos usando dados de exemplos ou experiências passadas (ALPAYDIN, 2014). Esses algoritmos podem ser utilizados para modelar o desempenho de forma genérica e simples em relação às modelagens analíticas que consideram as peculiaridades de um sistema específico.

Desse modo, mecanismos podem se confiar fortemente em modelos de desempenho para manter QoS com confiabilidade. Em particular, este trabalho utiliza a modelagem de desempenho para tratar a elasticidade. Com isso, auxilia os provedores de nuvem a garantir de forma adequada suas obrigações, descritas nos SLA, enquanto maximizam a utilização da infraestrutura subjacente alocada.

A elasticidade permite que os provedores de serviços adicionem e removam recursos computacionais disponíveis para as aplicações, sem interrupção do serviço, de modo a lidar com as variações na demanda, também denominada na literatura como carga de trabalho (COUTINHO et al., 2014; SOUSA; MACHADO, 2012), de tal sorte a evitar *underprovisioning*, que acarretam em penalidades de SLA, e *overprovisioning* que implica em custo desnecessários de infraestrutura (SANTOS et al., 2013). Serviços sem estado, como as aplicações *web* típicas e os servidores de aplicação, podem ser escalados horizontalmente sob demanda com a simples alocação ou remoção de recursos computacionais para prover o serviço. Por sua vez, os serviços com estado, como os SGBDs são mais difíceis de serem escalados. Isso ocorre porque, em um contexto mais amplo, esses serviços estão encarregados de manter a consistência dos dados que manipulam ao longo de diferentes registros, tabelas e bases de dados (SOUSA; MACHADO, 2014). Além disso, o ambiente em nuvem apresenta alta variação de desempenho que dificulta a alocação ótima de recursos (SCHAD et al., 2010). Nesse cenário, mecanismos de provisionamento dinâmico que se baseiam em modelos de desempenho e lidam com a variação intrínseca do desempenho em nuvem podem oferecer um provisionamento mais confiável.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

O objetivo geral desse trabalho é propor uma solução para lidar com a modelagem do desempenho e o provisionamento elástico de recursos de bancos de dados NoSQL em nuvem.

1.2.2 Objetivos Específicos

Os seguintes objetivos específicos foram estabelecidos para alcançar o objetivo geral deste trabalho:

- Investigar a possibilidade de utilizar técnicas de aprendizado de máquina (regressão) para modelagem do desempenho de bancos de dados NoSQL;
- Definir uma abordagem de provisionamento elástico baseado em modelos de desempenho; e
- Avaliar a qualidade da modelagem do desempenho e a efetividade da estratégia de provisionamento elástico.

1.3 CONTRIBUIÇÕES

Como o principal resultado dessa dissertação, tem-se as seguintes contribuições:

- Técnica baseada em algoritmos de aprendizado de máquina para modelagem de desempenho, considerando os aspectos de distribuição, de SGBDs NoSQL;
- Técnica para ajuste automático e dinâmico de recursos de acordo com um problema de otimização matemática baseados na modelagem do desempenho.

1.3.1 Produção Científica

As contribuições científicas apresentadas neste trabalho possibilitaram as seguintes publicações:

- FARIAS, VICTOR A. E. ; SOUSA, FLAVIO R. C. ; MAIA, J. G. R. ; MACHADO, J. C. ; GOMES, J. P. P. . Elastic Provisioning for Cloud Databases with Uncertainty Management. In: ACM Symposium on Applied Computing, 2016, Pisa. ACM Symposium on Applied Computing, 2016.
- FARIAS, VICTOR A. E. ; SOUSA, FLAVIO R. C. ; MACHADO, J. C. . Auto Escalonamento Proativo para Banco de Dados em Nuvem. In: SBBD, 2014, Curitiba. Simpósio Brasileiro de Banco de Dados, 2014.
- FARIAS, V. A. E.; MOREIRA, L. O. ; SOUSA, F. R. C. ; MAIA, J. G. R. ; ; MACHADO, J. C. ; SANTOS, G. A. C. . A Machine Learning Approach for SQL Queries Response Time Estimation in the Cloud. In: SBBD, 2013, Recife. XXVIII Simpósio Brasileiro de Banco de Dados, 2013.

Embora as publicações listadas abaixo não estejam diretamente vinculadas com o tema central desta dissertação, vários conceitos arquiteturais e relativos às melhores práticas para aplicações distribuídas foram desenvolvidos e utilizados no presente trabalho:

- da Silva, T. L. C., Neto, A. C. A., Magalhães, R. P., de Farias, V. A., de Macêdo, J. A., Machado, J. C. Towards an Efficient and Distributed DBSCAN Algorithm Using MapReduce. In Enterprise Information Systems. Springer International Publishing, 2014, pp. 75-90.
- Neto, A. C. A., da SILVA, T. L. C., de FARIAS, V. A. E., MACÊDO, J. A. F., de CASTRO MACHADO, J. . G2P: A Partitioning Approach for Processing DBSCAN with MapReduce. In Web and Wireless Geographical Information Systems. Springer International Publishing, 2015, pp. 191-202
- MOREIRA, L. O. ; FARIAS, V. A. E. ; SOUSA, F. R. C. ; SANTOS, G. A. C. ; MAIA, J. G. R. ; MACHADO, J. C. . Towards improvements on the quality of service for multi-tenant RDBMS in the cloud. In: Proceedings of the Sixth International Workshop on Cloud Data Management (CloudDB '14), 2014, Chicago. IEEE 30th International Conference on Data Engineering Workshops (ICDEW), 2014. p. 162-169.

1.4 ESTRUTURA DA DISSERTAÇÃO

Esta dissertação está organizada da seguinte forma: No Capítulo 2 são apresentados conceitos e definições fundamentais sobre computação em nuvem, SLA e banco de dados NoSQL. O Capítulo 3 ressalta e discute os trabalhos relacionados. Em seguida, o Capítulo 4 apresenta a abordagem proposta para a modelagem do desempenho e para elasticidade. O Capítulo 5 apresenta os resultados obtidos por um conjunto de experimentos realizados considerando diferentes cenários e métricas. Finalmente, o Capítulo 6 conclui o trabalho apresentando um resumo dos resultados alcançados e mostrando direções de pesquisa futuras.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 COMPUTAÇÃO EM NUVEM

A Computação em Nuvem está se tornando uma das palavras-chaves da indústria de Tecnologia da Informação (TI). A nuvem computacional é uma metáfora para a Internet ou infraestrutura de comunicação entre os componentes arquiteturais, representando uma abstração que oculta a complexidade da infraestrutura que suporta a execução de aplicações. Cada parte dessa infraestrutura é provida como um serviço e estes são normalmente alocados em centros de dados, utilizando hardware compartilhado para computação e armazenamento (BUYYA et al., 2009).

A computação em nuvem é uma evolução dos serviços e produtos de TI sob demanda, também chamada de *Utility Computing* (BRANTNER et al., 2008). O objetivo da *Utility Computing* é fornecer componentes básicos como armazenamento, processamento e largura de banda de uma rede como ”mercadoria" através de provedores especializados com um custo reduzido por unidade utilizada. Usuários de serviços baseados em *Utility Computing* não precisam se preocupar com escalabilidade, pois a capacidade de armazenamento fornecida é praticamente infinita. A *Utility Computing* propõe fornecer disponibilidade total, isto é, os usuários podem ler e gravar dados a qualquer tempo, sem serem bloqueados; os tempos de resposta são quase constantes e não dependem do número de usuários simultâneos, do tamanho do banco de dados ou de qualquer parâmetro do sistema. Os usuários não precisam se preocupar com *backups*, pois se os componentes falharem, o provedor é responsável por substituí-los e tornar os dados disponíveis em tempo hábil por meio de réplicas (BRANTNER et al., 2008).

Uma razão importante para a construção de novos serviços baseados em *Utility Computing* é que provedores de serviços que utilizam serviços de terceiros pagam apenas pelos recursos que recebem, ou seja, pagam pelo uso. Assim, dispensam-se investimentos iniciais em infraestrutura de TI. Além disso, o custo cresce de forma linear e previsível, de acordo com o uso. Dependendo do modelo do negócio, é possível que o provedor de serviços repasse o custo de armazenagem, computação e de rede para os usuários finais, já que é realizada a contabilização do uso (BINNIG et al., 2009).

Existem diversas propostas para definir o paradigma da computação em nuvem (VAQUERO et al., 2009). O *National Institute of Standards and Technology* (NIST) argumenta que a computação em nuvem é um paradigma em evolução e apresenta a seguinte definição: *"Computação em nuvem é um modelo que possibilita acesso, de modo conveniente e sob demanda, a um conjunto de recursos computacionais configuráveis (por exemplo, redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente adquiridos e liberados com mínimo esforço gerencial ou interação com o provedor de serviços"* (MELL; GRANCE, 2011). Segundo o NIST, a computação em nuvem é composta por cinco características essenciais, três modelos de serviço e quatro modelos de implantação, detalhados a seguir. O leitor interessado pode encontrar mais informações sobre os principais sistemas de computação em nuvem em (SOUSA et al., 2009).

2.1.1 Características Essenciais

As características essenciais da computação em nuvem são vantagens que as suas soluções oferecem, listadas a seguir:

- *Self-service sob demanda*. O usuário pode adquirir *unilateralmente* recurso computacional, como tempo de processamento no servidor ou armazenamento na rede, na medida em que necessite e sem precisar de interação humana com os provedores de cada serviço;
- *Amplo acesso*. Recursos são disponibilizados por meio da rede e acessados através de mecanismos padronizados que possibilitam o uso por plataformas do tipo *thin client*, tais como celulares, *laptops* e PDAs;
- *Pooling de recursos*. Os recursos computacionais do provedor são organizados em uma base (*pool*) para servir múltiplos usuários usando um modelo multi-inquilino (*multi-tenant*), com diferentes recursos físicos e virtuais, dinamicamente atribuídos e ajustados de acordo com a demanda dos usuários. Esses usuários não precisam ter conhecimento da localização física dos recursos computacionais utilizados, podendo somente especificar a localização em um nível mais alto de abstração, tais como o país, o estado ou o centro de dados;
- *Elasticidade rápida*. Recursos podem ser adquiridos de forma rápida e elástica, em alguns casos automaticamente, caso haja a necessidade de escalar com o aumento da demanda, e liberados, na retração dessa demanda. Para os usuários, os recursos disponíveis para utilização parecem ser ilimitados, podendo ser adquiridos em qualquer quantidade e a qualquer momento; e
- *Medição de serviço*. Sistemas em nuvem automaticamente controlam e otimizam o uso de recursos por meio de uma capacidade de medição. A automação é realizada em algum nível de abstração apropriado para o tipo de serviço, tais como armazenamento, processamento, largura de banda e contas de usuário ativas. O uso de recursos pode ser monitorado e controlado, possibilitando transparência para o provedor e o usuário do serviço utilizado.

Algumas destas características, em conjunto, definem exclusivamente a computação em nuvem, distinguido-a de outros paradigmas. Segundo NIST (MELL; GRANCE, 2011), a elasticidade rápida de recursos, o amplo acesso e a medição de serviço são características básicas para compor uma solução de computação em nuvem.

2.1.2 Modelos de Serviços

O ambiente de computação em nuvem é composto de três modelos de serviços. Esses modelos são importantes, pois delineam um padrão arquitetural para soluções de computação em nuvem:

- *Software como um Serviço (Software as a Service, SaaS)*. O modelo de SaaS proporciona sistemas de *software* com propósitos específicos que são disponibilizados para os usuários por meio da Internet e acessíveis a partir de vários dispositivos do usuário por meio de uma interface *thin client* como, por exemplo, um navegador Web. No SaaS, o usuário não administra ou controla a infraestrutura subjacente, incluindo rede, servidores, sistema operacional, armazenamento ou mesmo as características individuais da aplicação, exceto

configurações específicas. Como exemplos de SaaS podemos destacar os serviços de *Customer Relationship Management* (CRM) da Salesforce e o Google Docs;

- *Plataforma como um Serviço (Platform as a Service, PaaS)*. O modelo de PaaS fornece sistema operacional, linguagens de programação e ambientes de desenvolvimento para as aplicações, auxiliando a implementação de sistemas de software. Assim como no SaaS, o usuário não administra ou controla a infraestrutura subjacente, mas tem controle sobre as aplicações implantadas e, possivelmente, as configurações de aplicações hospedadas nesta infraestrutura. *Google App Engine* (CIURANA, 2009) e *Microsoft Azure* (AZURE, 2012) são exemplos de PaaS; e
- *Infraestrutura como um Serviço (Infrastructure as a Service, IaaS)*. A IaaS torna mais fácil e acessível o fornecimento de recursos, tais como servidores, rede, armazenamento e outros recursos de computação fundamentais para construir um ambiente de aplicação sob demanda, que podem incluir sistemas operacionais e aplicativos. Em geral, o usuário não administra ou controla a infraestrutura da nuvem, mas tem controle sobre os sistemas operacionais, armazenamento, aplicativos implantados e, eventualmente, seleciona componentes de rede, tais como *firewalls*. O Amazon *Elastic Cloud Computing* (EC2) (ROBINSON, 2008) e o Eucalyptus (LIU et al., 2007) são exemplos de IaaS.

2.1.3 Modelos de Implantação

Há diferentes tipos de modelos de implantação para os ambientes de computação em nuvem quanto ao acesso e à disponibilidade. A restrição ou abertura de acesso depende do processo de negócios, do tipo de informação e do nível de visão desejado:

- *Nuvem privada*. A infraestrutura de nuvem é utilizada exclusivamente por uma organização, sendo esta nuvem local ou remota. A nuvem é administrada pela própria organização ou por terceiros;
- *Nuvem pública*. A infraestrutura de nuvem é disponibilizada para o público em geral, sendo acessada por qualquer usuário que conheça a localização do serviço;
- *Nuvem comunitária*. Fornece uma infraestrutura compartilhada por uma comunidade de organizações com interesses em comum; e
- *Nuvem híbrida*. A infraestrutura é uma composição de duas ou mais nuvens, que podem ser do tipo privado, público ou comunidade e que continuam a ser entidades únicas, mas conectadas por meio de tecnologia proprietária ou padronizada que permite a portabilidade de dados e de aplicações.

2.2 ELASTICIDADE

Nos ambientes de computação em nuvem, a elasticidade é um dos pontos chave para desenvolver serviços com QoS, pois permite adicionar ou remover recursos, sem interrupções e em tempo de execução para lidar com a variação da carga. Um SGBD é elástico se ele ajusta automaticamente a quantidade de recursos para a carga de trabalho atual, ou seja, adiciona novos recursos se o sistema não consegue lidar com a carga de trabalho atual ou remove recursos

desnecessários (SOUSA; MACHADO, 2014).

A elasticidade é frequentemente associada com a escala do SGBD, mas existe uma sutil diferença entre elasticidade e escalabilidade. A escalabilidade é definida como a capacidade de um sistema satisfazer um requisito de aumento da carga de trabalho pela adição de uma quantidade proporcional de recursos. Isso é parte do que a elasticidade precisa, pois escalabilidade perfeita não garante elasticidade perfeita. A elasticidade permite ao sistema reduzir os recursos quando a carga de trabalho diminui, enquanto a escalabilidade considera apenas situações de aumento da carga de trabalho (COUTINHO et al., 2014). A variação da carga de trabalho pode colocar o sistema em dois tipos de situações indesejadas:

- *Underprovisioning*: O sistema não consegue atender de forma adequada a carga de trabalho por falta de capacidade acarretando em violações do SLA e lentidões na aplicação.
- *Overprovisioning:* O sistema dispõe capacidade demasiadamente grande para carga de trabalho atual o que resulta em recursos pouco utilizados e, consequentemente, em gastos desnecessários de infraestrutura.

As principais técnicas para implementar elasticidade são a provisionamento vertical e o provisionamento horizontal, descritas a seguir:

- Provisionamento vertical: O redimensionamento permite a ajuste da quantidade de recursos de acordo com a carga de trabalho. Por exemplo, pode-se incrementar ou decrementar a quantidade de CPU nas máquinas virtuais de forma automática, garantindo a qualidade e reduzindo os custos.
- Provisionamento horizontal: Técnicas de replicação são usadas para melhorar a disponibilidade, o desempenho e a escalabilidade em diversos ambientes. Assim, a replicação pode ser utilizada para implementar a elasticidade do ajuste automaticamente da quantidade de número de réplicas do serviço para a carga de trabalho atual. Elasticidade é alcançada por meio de provisionamento automático, que adiciona novas réplicas de serviços caso o sistema não consiga lidar com a carga de trabalho atual ou remove réplicas desnecessárias.

A estratégia de elasticidade proposta nesse trabalho faz uso da técnica de replicação de serviços. Em nosso caso, novas réplicas de serviços são instanciadas em novas máquinas virtuais para adicionar capacidade ao sistema.

2.3 ACORDO DE NÍVEL DE SERVIÇO

Existem muitos modelos gerais de SLA em nuvem (FITO et al., 2010), (MALKOWSKI et al., 2010) (SCHNJAKIN et al., 2010), assim como há, também, modelos específicos para serviços de banco de dados (YANG et al., 2009) (XIONG et al., 2011b) (CHI et al., 2011) (SOUSA et al., 2012). De acordo com (CHI et al., 2011), as métricas de SLA para serviços de banco de dados em nuvem devem otimizar o sistema, tratar aspectos relevantes para o gerenciamento de dados e contemplar as características do modelo de computação em nuvem.

SLAs devem refletir o valor econômico, as exigências de serviço do usuário, descrever as condições comuns de negócios, tais como métricas de avaliação, contabilidade e questões

jurídicas, prazos de contrato, bem como os aspectos técnicos de desempenho e de disponibilidade (MALKOWSKI et al., 2010). Contudo, na maioria dos casos, apenas de aspectos técnicos são abordados.

Com isso, é possível identificar uma estrutura geral para o SLA: informações sobre as partes envolvidas, parâmetros do SLA, métricas utilizadas para calcular os parâmetros do SLA, algoritmos para calcular os parâmetros do SLA, objetivo de nível de serviço (SLO) e quais ações a serem tomadas em caso de violação do acordo (SCHNJAKIN et al., 2010).

De acordo com (SOUSA et al., 2012), o SLA para Serviço de Banco de Dados em Nuvem é composto por informações das partes envolvidas, métricas do SLA, SLOs, algoritmos para calcular as métricas do SLA e penalidades. Informações sobre as partes envolvidas referem-se ao contrato entre o provedor e o cliente. As métricas do SLA estão relacionadas aos itens a serem monitorados, como, por exemplo, tempo de resposta e vazão. Por sua vez, o SLO contém os limites pré-definidos para um determinado parâmetro, como, por exemplo, tempo de reposta inferior a 5 ms. Para cada parâmetro é definida uma forma de calculá-lo (o tempo médio em um intervalor de tempo, por exemplo) e as penalidades referentes às ações em caso de não conformidade dos SLOs (geralmente uma multa).

Assim, é comum encontrar na literatura diferentes métricas para o SLA, dentre as quais pode-se destacar:

- *Tempo de reposta*. O tempo de resposta, em milissegundos, para cada consulta, durante um período de tempo t;
- *Vazão*. O rendimento mínimo, em transações por segundo, durante um período de tempo t;
- *Disponibilidade*. A fração máxima de consultas rejeitadas ao longo de um período de tempo t; e
- *Violações por segundo*. Quantidade máxima de requisições que violam o contrato SLA, ou seja, consultas rejeitadas ou consultas cujo tempo de resposta ultrapassou o limite pré-definido, por segundos.

Além dos aspectos supramencionados, alguns trabalhos abordam a relação entre custos e lucro. O SLA orientado ao lucro apresenta um funcionamento confiável dos sistemas, pois o provedor está motivado para prestar o serviço com uma qualidade elevada. Nos casos onde o provedor não é capaz de atender o SLA, este é incentivado a continuar a prestar o serviço até obter o lucro. Para definir o lucro, geralmente o SLA considera três parâmetros: receita, custo operacional e penalidades. A receita ou preço é o valor pago pelo cliente ao provedor para cumprir um SLA S_i de um determinado serviço. O custo operacional são os gastos do provedor para a execução de um serviço com um SLA S_i especificado, como, por exemplo, o custo com a infraestrutura. Dessa forma, o lucro consiste na receita obtida menos a soma dos custo mais as penalidades, conforme descrito na seguinte fórmula (SOUSA et al., 2012).

$$Lucro = Receita - (Custo + Penalidades) \tag{2.1}$$

A penalidade é o valor pago pelo provedor ao cliente, caso o SLA S_i não seja

cumprido. Por exemplo, no Google AppEngine [1], Microsoft Azure [2] ou Amazon S3 [3], se a disponibilidade ficar abaixo de $99,9\%$, em seguida, os clientes recebem um crédito no serviço de acordo com a qualidade do serviço e proporcional à receita.

Da mesma forma, o tempo de resposta é fundamental para garantir a qualidade do serviço e pode acarretar penalidades em alguns modelos de serviços (XIONG et al., 2011b). Em algumas abordagens, a penalidade é a razão entre a soma de todas as consultas violadas pelo total de consultas multiplicada pela receita do sistema, de acordo com a fórmula abaixo.

$$Penalidades = \frac{\sum consultas\_violadas}{\sum consulta} * Receita \tag{2.2}$$

Com isso, pode-se definir uma função de satisfação para o SLA, conforme apresentado abaixo. A função é atendida, caso o SLA S_i seja satisfeito, ou seja, todos os SLOs do SLA S_i sejam satisfeitos. Caso contrário, a função não é atendida.

$$FSS(Si) = \begin{cases} 1 & \text{se SLA } S_i \text{ é satisfeito} \\ 0 & \text{se SLA } S_i \text{ é violado} \end{cases} \tag{2.3}$$

2.4 BANCOS DE DADOS NOSQL

2.4.1 Definições Preliminares

Sistemas de Gerenciamento de Banco de Dados (SGBDs)[4] são candidatos potenciais para a implantação em nuvem. Isso ocorre porque, em geral, as instalações destes sistemas são complexas e envolvem uma grande quantidade de dados, ocasionando um custo elevado, tanto em *hardware* quanto em *software*. Para muitas empresas, especialmente para *startups* e médias empresas, o pagamento baseado no uso do modelo de computação em nuvem, juntamente com o suporte para manutenção do hardware é muito atraente (SOUSA; MACHADO, 2014).

Por décadas, o SGBDs relacionais têm sido considerados uma solução adequada para prover persistência e recuperação de dados. Contudo, a necessidade crescente por escalabilidade e diferentes requisitos das aplicações trouxeram à tona novos desafios para os tradicionais SGBDs relacionais. Além disso, recentemente, a nova geração de SGBDs de alto desempenho e baixo custo tem desafiado a dominância dos SGBDs relacionais. Esse novo sistemas são chamados de NoSQL (*Not only SQL*), significando que há apoio para além das linguagens de consulta baseadas no SQL (Structured Query Language).

Em oposição às transações ACID dos SGBDs relacionais, os SGBDs NoSQL seguem o teorema CAP (BROWNE, 2009) e suas transações obedecem os princípios BASE (*Basically Available, Soft state, Eventually consistent*) (PRITCHETT, 2008). De acordo com o teorema CAP, um SGBD distribuído pode escolher apenas duas entre as três propriedades seguintes:

[1] http://code.google.com/appengine/sla.html
[2] http://www.microsoft.com/windowsazure/sla
[3] http://aws.amazon.com/s3-sla/
[4] SGBD refere-se um uma classe geral de armazenamento de dados, incluindo sistemas não-relacionais.

Consistência, Disponibilidade e tolerância a partições. E, também, a maioria dos sistemas com transações ACID decide comprometer o requerimento de consistência estrita para ganhar maior disponibilidade e escalabilidade. Em particular, esses sistemas adotam uma política de consistência relaxada chamada consistência eventual.

Dois sistemas NoSQL, o BigTable (CHANG et al., 2008) proposto pelo Google e o DynamoDB (DECANDIA et al., 2007) suportado pela Amazon, inspiraram o desenvolvimento de outro novos sistemas como o MongoDB (INC, 2013), HBase (HBASE, 2013) e o Cassandra (PROJECT, 2013). Nós apresentaremos mais detalhes sobre o MongoDB nese capítulo, pois ele será usado em nossos experimentos.

Esse novos SGBDs alegam que são elásticos mas não proveem mecanismos para monitorar o ambiente e tomar decisões de adicionar ou remover recursos com base nas métricas de monitoramento como uso de CPU, uso de memória, vazão e latência. Por isso, é necessário adicionar outros componentes sobre o SGBD para se ter um sistema mais completo que é o que chamamos de Banco de Dados em Nuvem. Esses sistemas são compostos por: (i) um Gerenciador de Instâncias; (ii) um Gerenciador de SGBD; e (iii) um *pool* de instâncias que estão atualmente em execução ou estão disponíveis para serem usadas. O Gerenciador de Instâncias, composto por um Monitor e por um módulo de Tomada de Decisão, é responsável por colher métricas da *pool* de instâncias para que, com base nessas métricas, tome decisões quanto a instanciar ou parar máquinas virtuais na *pool*. O Gerenciador de SGBD é o ponto de acesso para o qual as requisições são enviadas. Dependendo do SGBD que está em uso, pode não existir um nó central que receba as requisições e as distribua na *pool*. Nesses sistemas, as requisições podem ser enviadas a qualquer instância na *pool*. A *pool* de instâncias é tida como um conjunto de instâncias de Máquina Virtual com o SGBD instalado, as quais podem ser iniciadas ou paradas pelo Gerenciador de Instâncias.

2.4.2 MongoDB

O MongoDB (INC, 2013), cujo nome é uma referência ao adjetivo "imenso"(*"humongous"*) é um SGBD NoSQL orientado a documentos de código aberto escrito em C++. Embora o MongoDB não seja relacional, ele implementa várias características de SGBDs relacionais como ordenação, indexação secundária e consultas por intervalo. Além disso, MongoDB também oferece *schema* dinâmico, replicação de dados para aumentar disponibilidade, auto-fragmentação, *Map/Reduce*, agregação e processamento de dados.

Diferentemente dos SGBD relacionais, o MongoDB não organiza seus dados em tabelas com colunas e linhas. Como alternativa, esse sistema guarda os dados em documentos que consistem em *arrays* associativos de escalares, valores, listas ou *arrays* associativos aninhados. Os documentos do MongoDB são naturalmente serializados como objetos *Javascript Object Notation* (JSON). Na prática, esses documentos são armazenados internamente usando uma codificação binária do JSON, denominada BSON (*Binary* JSON).

Em nosso trabalho, nós usamos o MongoDB em seu modo replicação. Ele funciona no esquema Mestre/Escravo, onde um *cluster* contém somente um mestre e os escravos possuem

um cópia completa da base de dados. Os clientes podem submeter requisições de leitura para qualquer nó, porém as leituras são enviadas sempre ao mestre. Para propagar as atualizações, o MongoDB armazena as atualizações em um *log*, de modo que os escravos coletam as entradas do *log* em *batch* e aplicam as operações necessárias. Adicionalmente, os escravos não permitem ler dados que estão sofrendo modificações.

O *driver* do MongoDB possui algumas estratégias de distribuição de carga para o balanceamento de requisições de leituras entre os nós do *cluster*. Em nosso trabalho, nós usamos a estratégia do MongoDB que se baseia em na escolha aleatória: inicialmente, o *driver* MongoDB elege os nós candidatos ao recebimento da operação de leitura; feito isso, um nó candidato é escolhido aleatoriamente para receber a requisição.

2.5 AVALIAÇÃO DE DESEMPENHO *(BENCHMARK)*

2.5.1 Avaliação de Desempenho de SGBDs na Nuvem

Tradicionalmente, o objetivo do *benchmark* é avaliar o desempenho de um dado sistema sob um carga de trabalho em particular e ajudar o provedor do serviço a ajustar o sistema. Os exemplos mais proeminente de avaliação de SGBDs, como também o SGBD com outras aplicações atreladas a ele como servidores de aplicação e servidores web, é a família TPC. Os *bechmark* ambientes de TPC focam em testar SGBDs transacionais que garatem as propriedades ACID.

Uma característica importante de um SGBD em nuvem é sua capacidade de se adaptar à variação da carga de trabalho. E, como dito anteriormente, a maioria dos novos sistemas em nuvem comumente não garantem as propriedades ACID, tem modelos de dados distintos, sacrificam a consistência pela disponibilidade oferecendo formas mais fracas de consistência. Logo, ferramentas de *benchmark* devem considerar as especificidades da computação em nuvem e dos novos SGBDs.

A ferramente mais referenciada que leva em consideração aspectos de avaliar SGBDs na nuvem é o YCSB (COOPER et al., 2010).

2.5.2 YCSB

O framework YCSB (COOPER et al., 2010) consiste em criar uma carga de trabalho gerada por clientes onde é possível explorar aspectos importantes do desempenho. Por exemplo, é possível criar cargas predominantemente compostas por leituras ou por escritas, além de existirem tipos de diferente de leituras e escritas. Outo ponto importante do framework YCSB é a extensibilidade: além das cargas de trabalho pré-definidas, o gerador de carga de trabalho facilita a definição de novas cargas de trabalhos personalizadas e é fácil adaptar o cliente para testar novos serviços de dados.

A arquitetura do YCSB é projetada de modo a prover uma camada de abstração para se adaptar à *API* de um serviço de dados específico. Para adicionar suporte a novos sistemas,

poucos métodos devem ser implementados, criando um interface para o esse novo SGBD. No momento da escrita desse trabalho, existem 17 interfaces para SGBDs diferentes.

As cargas de trabalho no *framework* YCSB são compostas por um conjunto de parâmetros como número de operações por segundo, número de *threads* de clientes, porcentagem de operação de cada tipo. A porcentagens de operações definem qual é porção das requisições submetidas ao SGBD testado que são de um determinado tipo. O tipos de operações são: *read*, *scan*, *insert* e *update*. Operações de *read* correspondem a consultas por igualdade que retornam apenas um tupla. Operações de *scan* são consultas por intervalo que recuperam um número aleatório, entre 1 e o parâmetro definido pelo usuário *maxscanlength*, de tuplas. E, finalmente, requisições *insert* consistem na inserção de novas tuplas no banco e requisições *update* correspondem a operações de atualização. Para ilustrar, mostramos abaixo exemplos de como são escritas as requisições *read* e *scan* em formato JSON para o MongoDB.

{'$query' : { '_id ': ' user1961247787207082852 ' }}
{'$query': {'_id': {'$gte': u'user3819172086670194905'}}}

Além dos parâmetro de carga de trabalho, também é possível definir parâmetros sobre a base de dados gerada para experimentos. Por exemplo, é possível modificar o tamanho da base de dados em número de tuplas, o número de campos em cada tupla e tamanho em *bytes* de cada campos.

Adicionalmente, o *framework* YCSB dispõe de um ferramenta de relatório de latência onde ele grava um *log* do experimento em disco contendo três informações sobre cada requisição: 1) Tipo da requisição (*read*,*scan*,*insert* ou *update*), 2) *timestamp* do momento da subimssão da consulta e 3) latência. Usamo essa ferramenta em nosso trabalho para reportar os resultados experimentais.

2.6 MODELAGEM DE DESEMPENHO

Administradores de Sistemas de Gerenciamento de Bancos de Dados concorrentes e distribuídos se deparam constantemente com questões como:

- *Planejamento de capacidade.* A capacidade atual do sistema é suficiente para garantir a QoS para a carga de trabalho corrente? Deve-se adicionar ou remover capacidade para atender o SLA ou economizar custos de infraestrutura?
- *Diagnóstico de Carga de Trabalho.* Quais classes de requisições da carga de trabalho degradam mais o desempenho? Que pares de classes tem a interação mais conflituosa por contenção de dados? e
- *Avaliação das decisões de projeto.* Como identificar se o modelo de distribuição, modelo de dados, particionamento dos dados e e outras configurações do banco são adequadas para a carga de trabalho que é submetida nele?

Nesses pontos, um análise preditiva sobre o desempenho considerando os recursos disponíveis, a configuração do sistema e a carga de trabalho atual é de fundamental importância. E, nesse trabalho, expressamos o desempenho em termos de métricas de desempenho de baixo *overhead* baseadas no SLA: (i) tempo de resposta médios de transações por segundo; (ii)

violações do SLA por segundo; e (iii) vazão. Formalmente, definimos um modelo de desempenho como uma função $M(cluster, carga\_de\_trabalho) \Rightarrow Métrica$ onde $cluster$ é um conjunto de parâmetros que descreve a estado do *cluster* que se quer obter predições, $carga\_de\_trabalho$ é um conjunto de parâmetros que descreve a carga de trabalho que está sendo submetida no sistema e $Métrica$ é um valor numérico de retorno da função M correspondendo a uma das métricas de desempenho citadas anteriormente.

Esse problema já foi muito estudado no contexto de SGBDs relacionais (DUGGAN et al., 2011; MOZAFARI et al., 2013). Mas tem recebido pouco atenção no contexto de SGDBs NoSQL em nuvem. Nesse sentido, SGBDs NoSQL são sistemas horizontalmente escaláveis, armazenadores de dados não-relacionais que surgem como alternativas a abordagens tradicionais. Eles servem uma grande quantidade de requisições concorrentes. E, nesses sistemas, a escalabilidade é possível, em geral, através da fragmentação, *i.e.*, partições horizontais sobre arquiteturas *shared-nothing* ou através usando de replicação para aumentar a disponibilidade e vazão dos dados. Usando essas estratégias, várias implementações de SGDBs NoSQL são capazes de ajustar seu desempenho (vazão, inserção de dados, latência) de acordo com a quantidade de recursos alocada (número de nós trabalhadores) (TSOUMAKOS et al., 2013). Isso também justifica a necessidade do modelo M tem parâmetros de *cluster* como entrada.

Os autores em (GRAY et al., 1996) alertam sobre os problemas de desempenho para manter a consistência dos dados de SGBDs que usam da estratégia de replicação, seja total ou parcial (dos fragmentos). Nessa estratégia um objeto é replicado em n nós distintos, mas só existe um nó que é dono desse objeto (Mestre). Em geral, as atualizações de um dado são feitas pelo dono dele (Mestre) ou podem ser efetuadas por outros nós desde que peçam ao Mestre e, a partir disso, as atualizações são propagadas de modo *lazy*, *i.e.*, são propagadas de modo assíncrono em transações separadas da transação que originou essa atualização. Com isso, o trabalho (GRAY et al., 1996) alega que o gargalo nesse cenário é a ocorrência de *deadlocks* distribuídos. E a quantidade de *deadlocks* pode ser aproximada por:

$$taxa\_de\_deadlock \approx \frac{(TPS \times Nós) \times updates^4}{4 \times tamanho\_DB^2} \tag{2.4}$$

Onde TPS é o número de transações por segundo, $Nós$ é o número de nós no *cluster*, $updates$ é o número de atulizações por transação e $tamanho\_DB$ é número de objetos na base de dados. A Equação 2.4 sugere que exista uma quantidade quadrática de *deadlocks* em relação a TPS e $Nós$, i.e., a ocorrência de *deadlocks* degrada o desempenho de forma quadrática em relação ao aumento do tamanho do *cluster* e da volume da carga de trabalho. Note que vários SGBDs NoSQL usam replicação em seu projeto e oferecem consistência forte como Redis (LABS, 2015), HBase (HBASE, 2013) e BigTable (CHANG et al., 2008).

Adicionalmente, estudamos também *cargas de trabalhos heterogêneas*, *i.e.*, cargas de trabalhos compostas por requisições com complexidades diferentes de demandas de E/S também diferentes. Isso implica que porções da carga de trabalho vão impactar mais no desempenho que outras porções. Também é interessante citar que o compartilhamento de recursos causa interações complexas entre as requisições da carga de trabalho (MOZAFARI et al., 2013). Essas

cargas de trabalhos fazem oposição a cargas de trabalhos homogêneas onde as requisições tem complexidade aproximadamente iguais o que é irreal.

2.7 APRENDIZADO DE MÁQUINA

Aprendizado de máquina é um uma área da inteligência artificial dedicada a estudar técnicas de programar computadores de modo a aprender um determinado fenômeno usando dados de exemplos ou experiências passadas (ALPAYDIN, 2014). Para ser inteligente, um sistema em um ambiente em mudança deve ter a habilidade de aprender. Se um sistema pode aprender a se adaptar a tais mudanças, o projetista do sistema não precisa prever ou prover soluções para todas a situações possíveis. Em vez disso, o sistema deve ser capaz de inferir hipóteses sobre um conjunto de dados ou experiência, o que resulta em modelos preditivos que generalização o dado na esperança de se obter conclusões corretas.

O objetivo do aprendizado de máquina raramente é replicar o comportamento do conjunto de treinamento mas é fazer predições para novos casos, o que significa em gerar a saída certa para uma instância de entrada fora do conjunto de treinamento. O quão bom um modelo treinado sobre o um conjunto de treinamento prediz o valor de saída correto para novas instâncias é chamado de *generalização* (ALPAYDIN, 2014).

Essa abordagem que dispõe dessa habilidade de aprender de experiências passadas em muito útil em situações práticas pois é, em geral, é mais fácil obter dados de de experiências sobre o problema que ter boas suposições teóricas sobre as leis que governam o sistema em que o dado foi gerado (ZHANG et al., 1998).

Consideremos o exemplo de reconhecer dígitos escritos à mão ilustrados na Figura 1. Cada dígito corresponde a uma imagem de 28x28 pixels e, por isso, pode ser representada por um *vetor de características x* de 784 números reais e cada posição de vetor correponde a uma *característica*. O objetivo é construir um programa que recebe um vetor x e produza a identidade do dígito: $0, 1, 2, .., 9$ como saída. Esse problema é complexo pois existe uma alta variabilidade de maneiras de se escrever um número à mão. Esse problema poderia ser atacado usando regras manualmente escolhidas ou heurísticas baseadas nas formas das linhas. Mas, na prática, essa abordagem produz uma quantidade enorme de regras e muitas exceções a essas regras o que pode inviabilizar essa abordagem.

Resultados melhores e mais simples podem ser obtidos a partir de abordagens baseadas em técnicas de aprendizado de máquina. Nessa abordagem, exige-se um conjunto de n dígitos em formato de vetor de características $\{x_1,x_2,...,x_n\}$ chamado de *conjunto de treinamento* e é usado para ajustar os parâmetros de um modelo adaptativo de aprendizado. As categorias (0,1,2,..,9) dos dígitos são conhecidas previamente, i.e., foram previamente rotuladas manualmente. Expressamos a categoria de cada dígito usando o *vetor alvo t* onde cada posição de t representa a categoria, *valor alvo* ou *valor de saída* de um dígito no conjunto de treinamento.

O resultado da execução de um algoritmo de aprendizado é um função ou *modelo preditivo* $y(x)$ que recebe x como entrada e gera a saída ou categoria para x. A forma precisa de y é determinada durante a *fase de treinamento* baseado no conjunto de treinamento. Assim, uma

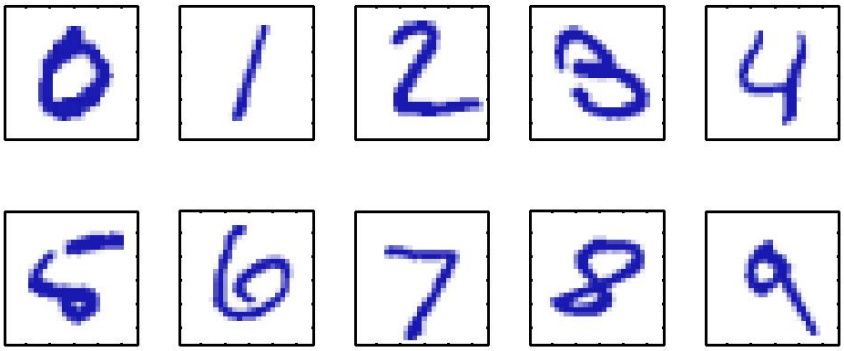


Figura 1 – Exemplos de dígitos escritos à mão. (Fonte: (BISHOP, 2006))

vez que treinamos o modelo, é possível gerar a identidade para novas imagens de dígitos o que é chamados de *conjunto de treinamento*. E a habilidade de categorizar corretamente esses novos exemplos é chamado de generalização como citado anteriormente. Modelos com alta capacidade de generalização são ditos como modelos *acurados*. Em particular, a habilidade de se categorizar um novo exemplo x que se encontra fora dos intervalos do conjunto de treinamento se chama *extrapolação*.

Aplicações que o conjunto de treinamento compreende os vetores de características e os seus valores alvos são conhecidos como problemas de aprendizagem supervisionada. No caso do reconhecimento de dígitos, onde é necessário atribuir uma categoria entre um número finito e discreto de categorias, são chamados de *problemas de classificação*. Por sua vez, se o valor de saída for um valor contínuo, então esse problema é chamado de *problema de regressão*.

Em outro problemas, o conjunto de treinamento consiste no no conjunto de vetores de características X sem o vetor alvo. Esses problemas são chamados de problemas de *aprendizado não-supervisionado* e compreende várias tarefas como: achar grupos de dados mais similares que é chamado de *clusterização* ou determinar a distribuição dos dados que também é conhecida como *estimativa de densidade* (BISHOP, 2006).

Nesse trabalho, consideramos o problema de modelar o desempenho de um Banco de Dados NoSQL como um problema de regressão. Pois as métricas de desempenho são contínuas (tempo de resposta médio) ou discretas e não-finitas (vazão, violações por segundo). Além disso, o uso de algoritmos de aprendizado de máquina mantém a nossa solução mais genérica pois não é necessário usar peculiaridades de um SGBD para conseguir modelar seu desempenho como a abordagens analíticas para modelagem do desempenho.

Em nosso solução, testamos várias técnicas de regressão, mas nem todos obtiveram bons resultados e, nesse trabalho, analisaremos apenas as seguintes técnicas de regressão:

- *Regressão linear com regularização*. É um dos métodos mais simples conhecidos na literatura. Esse métodos gera modelos que capturam correlações lineares entre as características e os valores alvo através do métodos dos mínimos quadrados. Para se achar correlações polinomiais, é possível aplicar um transformação polinomial nas características (BISHOP, 2006).

- *Gradient Boosting Machine* (OLSHEN et al., 1984) (FRIEDMAN; MEULMAN, 2003). Esse preditor constrói iterativamente um conjunto de modelos de aprendizado base. Cada modelo base é construído com base no resíduo do último preditor e esse erro é otimizado usando o algoritmo de gradiente descendente. Segundos os autores, este método é capaz de capturar relações não-lineares entre as características e os valores alvo. Mas, como esse método é baseado em árvores, ele tende a ter baixo poder de extrapolação (LOH et al., 2007).

2.8 DENSIDADES DE PROBABILIDADES

As métricas estudadas nesse trabalho podem ser tratadas como variáveis discretas ou contínuas. Para efeitos de genericidade, consideramos que as métricas são contínuas. Desse modo, introduziremos alguns conceitos básicos do densidades de probabilidades de variáveis contínuas. Esses conceitos serão usadas principalmente nas seções de gerenciamento da incerteza desse trabalho.

Densidade de probabilidade: Se a probabilidade de uma variável x estar entre os intervalos (x,$x+\delta x$) for dado por $p(x)\delta x$ para $\delta x \to 0$ então $p(x)$ é chamada de *densidade de probabilidade* sobre x. Isso é ilustrada na Figura 2. A probabilidade de x estar no intervalo (a,b) é dado por:

$$p(x \in (a,b)) = \int_b^a p(x)dx \tag{2.5}$$

E, como probabilidades são não-negativas e como os valores de x estão situados no domínio real, a densidade de probabilidade $p(x)$ deve satisfazer duas condições:

$$p(x) \geq 0 \tag{2.6}$$

$$\int_{-\infty}^{\infty} p(x)dx = 1 \tag{2.7}$$

Função de distribuição acumulada: A probabilidade de x estar no intervalo ($-\infty$,z) é dada pela função de distribuição acumulada, ilustrada na Figura 2, definida por:

$$P(z) = \int_{-\infty}^{z} p(x)dx \tag{2.8}$$

Função de distribuição acumulada inversa: A função inversa $P^{-1}(y)$ da Função de distribuição acumulada P que retorna um valor z que representa o intervalo ($-\infty$,z) em que a variável x está com probabilidade y.

Em nosso trabalho, usamos a *Distribuição Gaussiana* para se modelar o desvio-padrão das métricas de desempenho. A distribuição Gaussiana também conhecida como distribuição Normal é amplamente usada para modelar distribuições de variáveis contínuas. No

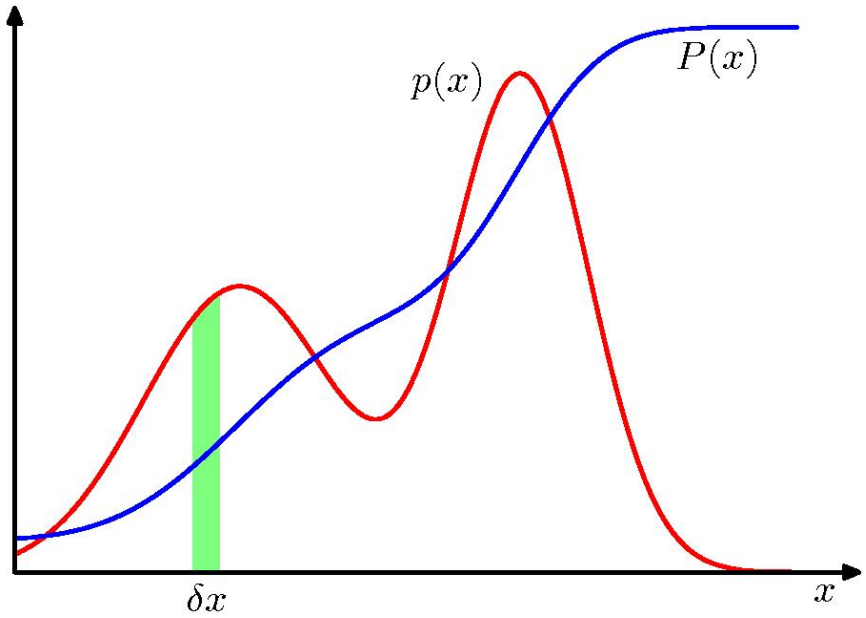


Figura 2 – Densidade de probabilidade $p(x)$ e Função de distribuição acumulada $P(X)$. (Fonte: (BISHOP, 2006))

caso unidimensional, a densidade de probabilidade da distribuição Gaussiana pode ser escrita da seguinte forma:

$$\mathcal{N}(\mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right) \tag{2.9}$$

Onde μ é a média que representa a localidade da distribuição Gaussiana e σ^2 é a variância que denotada a largura da distribuição Gaussiana.

2.9 CONCLUSÕES PRELIMINARES

Este capítulo apresentou o arcabouço necessário para construir a modelagem do desempenho. Foi definido os aspectos mais importantes sobre computação em nuvem: os seus modelos de serviço e implantação, elasticidade e acordo de nível de serviço assim como aspectos relacionados a banco de dados NoSQL e *benchmark*. Também foram apresentados técnicas de aprendizado de máquina, tais como aprendizado supervisionado (ex: classificação e regressão) e não-supervisionado (ex: clusterização e estimativa da densidade). Além disso, foram introduzidas funções de densidade de probabilidade necessárias para parte de gerenciamento de incerteza.

3 TRABALHOS RELACIONADOS

3.1 INTRODUÇÃO

Este capítulo apresenta trabalhos relacionados que propõem modelagens de desempenho e elasticidade para bancos de dados de modo geral. Nesse sentido, são descritos os trabalhos mais expressivos, considerando seus pontos fortes e suas limitações. Ao final deste capítulo é realizada uma análise comparativa entre os trabalhos expostos.

Para abordagens de modelagem de desempenho para bancos de dados NoSQL, exige-se flexibilidade no atendimento a uma variedade de modelos de distribuição e modelos de dados que disponíveis no mundo NoSQL. Cada um desses aspectos influencia no desempenho e estas abordagens devem capturar tais especificidades para gerar modelos de desempenho acurados. Foram estudados trabalhos que tratam o desempenho para requisições isoladas submetidas ao banco de dados e para cargas de trabalhos onde se consideram aspectos de concorrência entre as requisições que compartilham o recurso simultaneamente.

No contexto de abordagens para elasticidade de bancos de dados NoSQL foram analisados trabalhos que implementam estratégias para tratar alguns dos seguintes aspectos: (i) capacidade de adicionar e remover recursos computacionais; (ii) genericidade em relação ao SGBD NoSQL subjacente; e (iii) sensibilidade às variações da carga de trabalho, tanto de volume como de composição das consultas.

3.1.1 Estratégia proposta em (GANAPATHI et al., 2009)

Essa estratégia trata a predição de múltiplas métricas de desempenho e uso de recursos na execução de consultas SQL. A predição se baseia somente em informações disponíveis antes da execução da consulta, como o código e o plano de consulta gerado pelo otimizador de consultas. Especificamente, as métricas abordadas são: uso de CPU, memória e disco. Os experimentos mostram que algumas consultas são executadas em tempo da ordem de milissegundos enquanto outras, que são comumente utilizadas como apoio em processos de tomada de decisão, levam horas para terminar suas execuções.

Os autores baseiam sua abordagem em algoritmos de aprendizado de máquina devido à excessiva complexidade de modelos analíticos. Uma série de técnicas de aprendizado de máquina, como regressão linear, *k-means*, *Principal Component Analysis* (PCA), *Canonical Correlation Analysis* (CCA) e *Kernel Canonical Correlation Analysis* (KCCA) foram analisadas. Dentre essas técnicas, os autores julgaram o KCCA mais adequado devido à sua capacidade de encontrar e projetar os dados nas dimensões de maior correlação entre conjuntos de dados multidimensionais, além de permitir o uso de funções *kernel* para transformação do espaço.

Dois conjuntos de dados multidimensionais são gerados nessa abordagem. O primeiro deles é chamado de matriz de características, que corresponde aos dados sobre as consultas. O segundo conjunto, chamado de matriz de valores alvos, corresponde às métricas de desempenho. Estes dois conjuntos de dados são construídos a partir de experimentos, no quais uma

série de consultas SQL é executada e o conjunto de características é extraído a partir do texto SQL e do plano de consulta. As métricas de desempenho são observadas na execução dessas consultas. Assim, a fase de treinamento, como mostrado na Figura 3, consiste em projetar esse dois conjuntos de dados por meio do KCCA com uma função *kernel* gaussiana em um espaço de maior correlação.

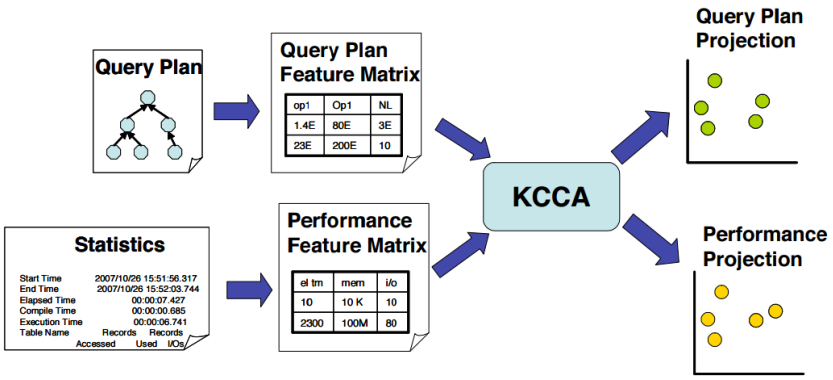


Figura 3 – Fase de Treinamento. (Fonte: (GANAPATHI et al., 2009))

A fase de predição, ilustrada na Figura 4, tem como objetivo estimar o conjunto de métricas para uma nova consulta SQL. A predição é realizada em três passos:

1. As características do texto SQL e do plano de consulta são extraídas para geração de um vetor de características. Esse vetor é projetado no novo espaço de características através do KCCA;
2. A técnica *K-nearest neighbor* é usada para encontrar as projeções de consultas vizinhas mais similares assim como para achar as projeções de métricas de desempenho como maior similaridade; e
3. É realizado um mapeamento inverso das projeções dos vetores de desempenho mais similares no espaço original com o objetivo de obter o vetor de métricas preditas.

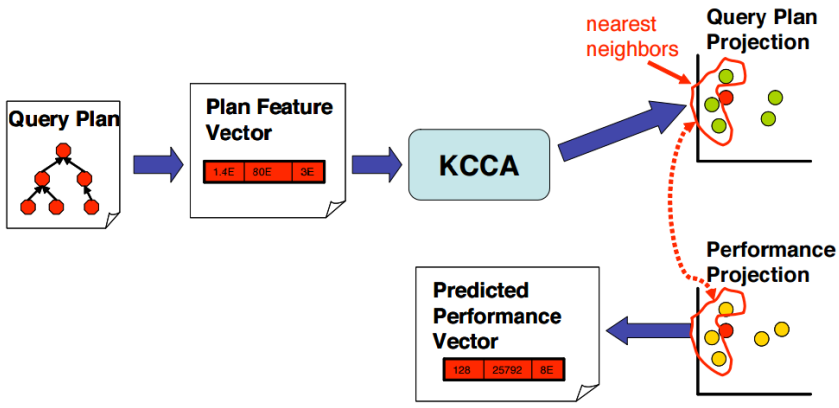


Figura 4 – Fase de predição.(Fonte: (GANAPATHI et al., 2009))

A principal limitação desse trabalho é a falta de tratamento de execuções concorrentes de consultas. Além disso, apenas o contexto de bancos de dados relacionais é considerado. Este trabalho é exclusivamente uma abordagem de modelagem de desempenho e, por isso, não considera o provisionamento de recursos.

3.1.2 DBSeer

O DBSeer (MOZAFARI et al., 2013) visa predizer métricas de desempenho para bancos de dados relacionais centralizados. Nessa abordagem são coletadas estatísticas do SGBD para construir os modelos de desempenho *offline* de acordo com a carga de trabalho executada. Esses modelos estimam o desempenho por meio da avaliação do uso de recursos computacionais (CPU, disco, RAM, *cache*, *locks*), considerando as consultas contidas na carga de trabalho e os recursos alocados.

Duas classes de modelos são propostas:

1. *Modelos black-box*. Fazem o mínimo de suposições sobre o SGBD subjacente e usam métodos de aprendizado de máquina, no caso, regressão, para predizer o desempenho utilizando como base experiências passadas; e
2. *Modelos white-box*. Consideram os componentes do SGBD para a realização de predições mais precisas. Mais especificamente, são construídos modelos para E/S de disco, contenção de *lock* e utilização de memória para o SGBD.

A maior contribuição desse trabalho consiste na análise de cargas de trabalhos OLTP altamente concorrentes e suas interações complexas. O DBSeer é dividido em 3 principais etapas (Figura 5): (1) Coleta de logs, na qual são coletados estatísticas do SGBD em execução no seu estado de produção; (2) Pré-processamento, onde os *logs* são agregados e combinados e as consultas SQL são agrupadas segundo seu padrão de acesso, gerando assim as classes de carga de trabalho; (3) Modelagem, na qual são construídos modelos *black-box* e *white-box* para predizer a utilização de recursos (CPU, RAM, E/S de disco e *locks*). Como principais desvantagens, o DBSeer considera apenas SBGBDs relacionais centralizados e não aborda aspectos de provisionamento.

3.1.3 PROMPT

PROMPT (DIDONA; ROMANO, 2014) é uma estratégia de modelagem de desempenho para bancos de dados chave-valor transacionais parcialmente replicados. PROMPT combina modelagem analítica *white-box* com técnicas de aprendizado de máquina *black-box* com o objetivo de aproveitar o melhor das duas metodologias: baixo tempo de treinamento, alto poder de extrapolação e portabilidade entre infraestruturas de nuvem heterogêneas.

O modelo analítico é responsável por capturar os efeitos da localidade dos dados e da contenção de dados na vazão considerando a mudança da localidade dos dados ao se escalar o sistema. Para isso, o modelo analisa o controle de concorrência e o modelo de replicação de um banco de dados NoSQL em memória e transacional Infinispan. Já as técnicas de aprendizado de

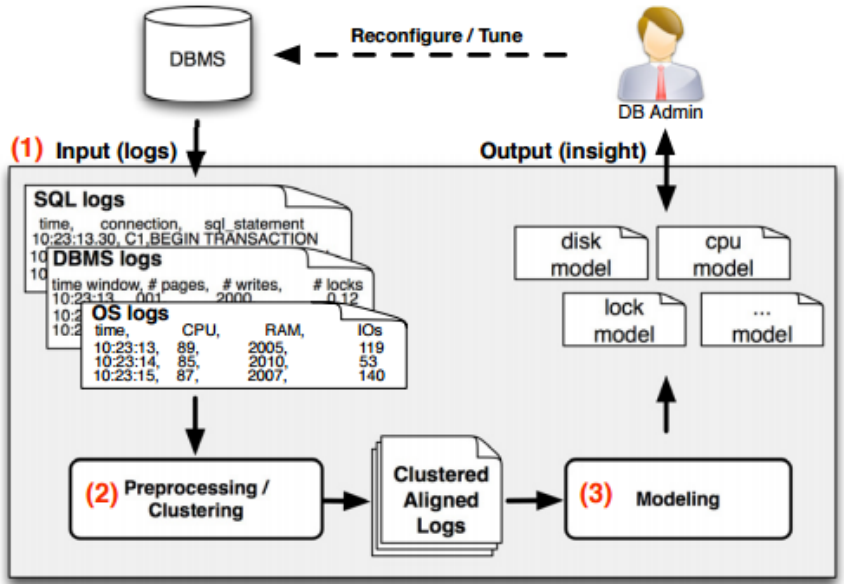


Figura 5 – Visão geral do DBSeer. (Fonte: (MOZAFARI et al., 2013))

máquina são empregadas para prever o tempo de resposta de operações que fazem uso intensivo da rede. Os autores introduzem a noção de múltiplos modelos mais simples de aprendizado de máquina para melhorar a acurácia de um modelo *black-box* mais complexo, onde cada modelo simples captura diferentes cenários de carga de trabalho e configurações do sistema.

O PROMPT apresenta limitações, pois a modelagem de desempenho é construída sobre algumas premissas do SGBD subjacente (Infinispan): (i) SGBD é baseado em uma função HASH para determinar a localidade de um item; (ii) SGBD tem consistência e isolamento fracos para ganhar escalabilidade; e (iii) SGBD implementa uma variante não-serializável do algoritmo de controle de concorrência multiversão que nunca bloqueia ou aborta transações sobre operações de leitura. Isso posto, contribuição torna-se muito específica dentro do contexto de SGBDs NoSQL.

3.1.4 SmartSLA

O SmartSLA (XIONG et al., 2011a) é o único trabalho relacionado que ataca o problema de provisionamento elástico de recursos e modelagem do desempenho em um banco de dados em nuvem. Os autores propõem uma estratégia de alocação e de elasticidade por meio de um modelo de desempenho do sistema. O trabalho considera o contexto de multi-inquilino. Assim, a estratégia busca compartilhar e alocar máquinas virtuais em máquinas físicas visando atender os requisitos de SLA de cada cliente enquanto maximiza os lucros do provedor de nuvem. Caso os requisitos de SLA de um cliente não sejam atendidos com a reorganização da alocação das máquinas virtuais, a estratégia aloca novas réplicas com o objetivo de melhorar a QoS.

A modelagem do sistema é puramente baseada em técnicas de aprendizado máquina, ou seja, métodos de regressão, para prever o custo de penalidades de SLA, proporcional à quantidade de violações de SLA, para uma dada configuração do sistema. O modelo tem como

entrada a *share* de CPU da máquina virtual, a quantidade de memória, o número de réplicas e o volume de transações ao sistema, conforme apresenta a Figura 6. Para a construção desse modelo, os autores executaram experimentos iniciais para medir o desempenho do sistema sob diversas configurações de máquinas virtuais e para gerar o conjunto de treinamento. O *benchmark* usado é projetado para SGBDs relacionais TPC-W. Foram testados três métodos: regressão linear, árvore de regressão e abordagens *boosting*.

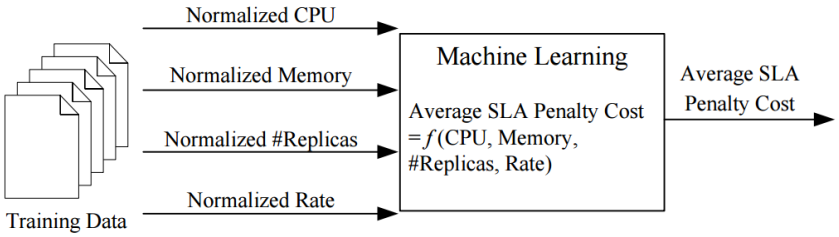


Figura 6 – Modelo do sistema em SmartSLA. (Fonte: (XIONG et al., 2011a))

A realocação dinâmica de recursos é feita em dois níveis:

1. CPU e memória. Esse nível tem como objetivo dividir de maneira ótima os recursos entre os clientes que compartilham os mesmos recursos, i.e., dividir os *share* de CPU e de memória das máquinas virtuais que compartilham a mesma máquina física. Para isso, os autores propõem um problema de otimização que visa minimizar as penalidades de SLA com base no modelo de desempenho do sistema; e
2. Réplicas de banco de dados. Nesse nível é analisado o ajuste do número de réplicas para um determinado cliente com o intuito de reduzir o custo total causado por penalidades de SLA, o custo de infraestrutura e o custo de provisionamento.

Esse trabalho trata da distribuição do SGBD mas não captura as especificidades da carga de trabalho, pois utiliza apenas um parâmetro, o volume de requisições, para descrevê-la, tornando-a muito simples.

3.1.5 MeT

MeT consiste de uma estratégia de elasticidade para bancos de dados NoSQL que adiciona ou remove recursos, assim como reconfigura o ambiente de modo heterogêneo para atender à demanda de requisições e para mitigar o problema de *hotspots* (CRUZ et al., 2013).

A arquitetura do MeT é composta por três componentes principais:

1. Monitor. Colhe estatísticas importantes sobre o *cluster* em estado de produção, passando, periodicamente, essas informações ao tomador de decisões;
2. Tomador de decisões. Núcleo da abordagem que é capaz de decidir quantos nós devem ser adicionados ou removidos em caso de inatividade ou sobrecarga, respectivamente, do sistema; e
3. Atuador. Implementa as ações de provisionamento que foram calculadas pelo tomador de decisões.

O algoritmo de decisão utilizado para adição e remoção de nós do sistema é baseado em regras estáticas e é executado periodicamente. São definidos limiares superiores e inferiores para as métricas de monitoramento com o objetivo de verificar se o *cluster* está em estado subótimo. Caso seja identificado que o *cluster* está inativo, um nó é removido. Uma visão geral do MeT é apresentada na Figura 7

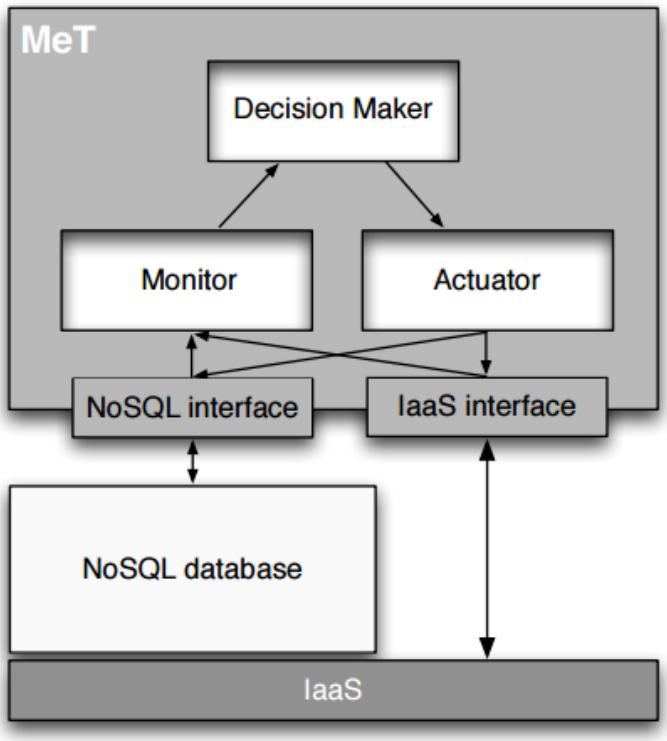


Figura 7 – Arquitetura MeT. (Fonte: (CRUZ et al., 2013))

No caso de sobrecarga, MeT adota um estratégia de adição quadrática de nós para responder rapidamente ao aumento no volume da carga de trabalho, de modo a atingir a uma situação aceitável em um número logarítmico de iterações. Uma vez que uma sobrecarga é detectada no sistema, a estratégia adiciona um nó ao sistema. Caso uma sobrecarga ainda seja detectada nas iterações seguintes, dois nós são adicionados na segunda iteração, quatro nós na terceira iteração e assim por diante, até que o *cluster* atinja um estado aceitável. Essa estratégia, em muitos casos, provisiona um número de nós maior do que o necessário (*overprovisioning*), o que implica em custos dispensáveis de infraestrutura. Além disso, os autores não consideram a composição das cargas de trabalho nem sua variação ao longo do tempo.

3.1.6 Tiramola

Em (TSOUMAKOS et al., 2013; KONSTANTINOU et al., 2011; KONSTANTINOU et al., 2012), os autores apresentam o Tiramola, um sistema que provê uma camada entre o provisionamento automático de recursos ao nível de infraestrutura, ou seja, máquinas virtuais,

para alcançar a elasticidade do banco de dados NoSQL. Tiramola permite a expansão ou a contração do *cluster* virtualmente para qualquer banco de dados NoSQL que esteja equipado com funcionalidades de adição ou remoção de recursos. Uma visão geral do Tiramola é apresentada na Figura 8.

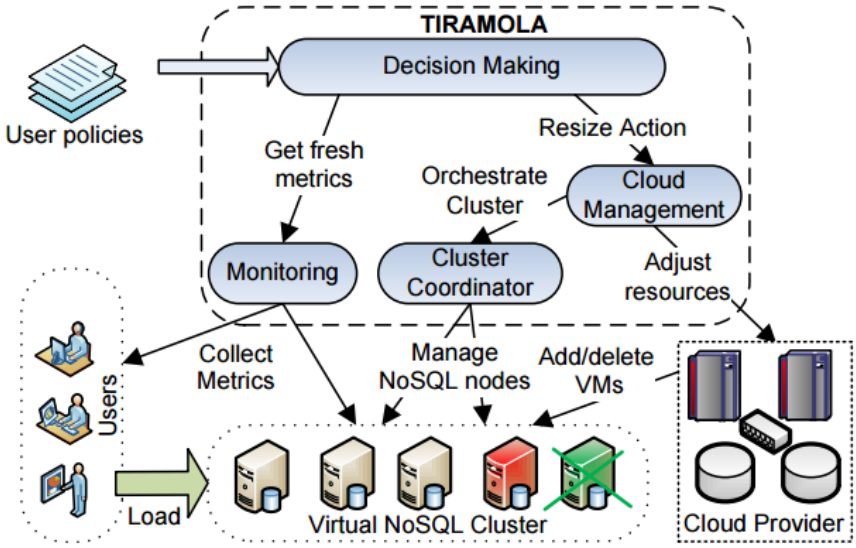


Figura 8 – Arquitetura Tiramola. (Fonte: (KONSTANTINOU et al., 2011))

O Tiramola é dividido em 4 módulos:

1. Monitoramento. Este o módulo faz uso da ferramenta de monitoramento distribuído e escalável Ganglia (MASSIE et al., 2004), que reporta estatísicas do *cluster* via uma API XML;
2. Gerenciador de Nuvem. Interage com a nuvem para criação e remoção de máquinas virtuais por meio da ferramenta *euca2tools*;
3. Coordenador de cluster. Este módulo é capaz de fazer uso das máquina virtuais recém instanciadas e incorporá-las no *cluter* NoSQL por meio das APIs disponibilizadas pelo SGBD; e
4. Tomada de decisões. Responsável pelo redimensionamento adequado do *cluster* de acordo com a carga de trabalho sendo executada, com o *cluster* instanciado e com a política de otimização definida pelo usuário.

O processo de tomada de decisão é definido como um Processo de Decisão de Markov (PDM), que identifica a ação mais vantajosa para o usuário de acordo com o estado atual do sistema. O usuário é responsável por fornecer a função de recompensa, que traduz os requisitos de QoS de cada aplicação.

As ações de redimensionamento são modeladas como um PDM (S, A, P, γ, r). $S = \{S_1, S_2, S_3, ..., S_m\}$ são os estado do processo, onde S_i representa um *cluster* com i nós ativos. As ações $A(s)$ indicam as ações que podem ser tomadas ao se trocar de estado. São definidas 3 tipos de ações: $no-op$ (não executar ação), $add-i$ (adicionar i nós) e $remove-i$ (remover i

nós). P é a matriz de probabilidades de se transitar de estado dentro do PDM. $r(s)$ é a função de recompensa que representa quão beneficente é estar no estado s através de um valor numérico que quantifica os ganhos e os custo do estado s. Em particular, o ganho considerado é a latência de resposta das requisições do cliente e os custos correspondem ao custos de infraestrutura em se alocar máquinas virtuais.

Essa abordagem consegue se manter genérica para qualquer tipo de banco de dados, uma vez que o PDM consegue se adaptar de forma online. Todavia, esse trabalho se limita a analisar apenas o volume de requisições que são submetidas ao sistema por unidade de tempo, ou seja, o tipo e a complexidade das requisições não são analisados.

A abordagem exposta em (KASSELA et al., 2014) estende o (KONSTANTINOU et al., 2011) por meio da melhoria na análise da carga de trabalho. Esse trabalho modifica o módulo de tomada de decisão para suportar cargas de trabalho com tipos diferentes de requisições, tais como leitura e escrita, e para analisar a interferência entre essas operações, causadas por *caching* e *locking* no SGBD. Para isso, a função de recompensa é modificada para contar com a vazão e com a latência das requisições de leitura e de escrita.

Embora (KASSELA et al., 2014) seja capaz de tratar separadamente operações de leitura e de escrita e suas interações, as requisições de leitura e escrita podem ter complexidades e demandas diferentes de E/S de disco. Por exemplo, a carga de trabalho pode conter consultas por igualdade que retornam apenas uma tupla. Por outro lado, consultas por intervalo retornam um número maior de tuplas que, por consequência, acarretam uma demanda maior de disco e de rede.

3.2 DISCUSSÃO

Os trabalhos apresentados nessa seção propõem estratégias de elasticidade e modelagem de desempenho para Bancos de Dados. Elucidamos quais tipos de Bancos de Dados cada trabalho trata e qual é a estratégia usada para atingir seus objetivos.

O nosso trabalho é tratar elasticidade e modelagem do desempenho para Bancos de Dados NoSQL em Nuvem. Nossa estratégia de modelagem do desempenho é baseada em técnicas de aprendizado de máquina e trata a execução simultânea de consultas que compartilham em uma carga de trabalho heterogênea, *i.e.*, uma carga de trabalho com requisições de leitura e escrita com complexidades diferentes, ou seja, consultas por igualdade e consultas por intervalo. Por sua vez, nossa estratégia de elasticidade é baseada em um problema de otimização matemática que faz uso de modelos de desempenho para computar a alocação de recursos ótima para a carga de trabalho corrente. Também analisamos cargas de trabalhos heterogêneas no cálculo da alocação de recursos para a carga corrente. Essa característica torna a estratégia mais confiável pois a carga pode mudar de composição ao longo do tempo (ex: proporção de requisições complexas aumenta) e nossa estratégia consegue lidar com esse problema.

Na seção 3.1, foram apontadas os pontos fortes e as falhas de cada trabalho. No contexto de modelagem do desempenho, os Bancos de Dados NoSQL têm recebido pouca atenção. Apenas o trabalho (DIDONA; ROMANO, 2014) aborda um tipo de Banco de Dados

NoSQL, os Bancos chave-valor transacionais parcialmente replicados. Além disso, os trabalhos falham em pelo menos um dos seguintes quesitos: (i) não tratam cargas de trabalhos; (ii) não tratam cargas de trabalho heterogêneas; ou (iii) não tratam Bancos de Dados em Nuvem. Em relação aos trabalhos que tratam elasticidade, todos os trabalhos que citamos na seção 3.1 abordam Bancos de Dados NoSQL em nuvem mas também apresentam pelo menos um dos seguintes pontos negativos: (i) não tratam cargas de trabalhos heterogêneas; ou (ii) o tratamento da carga é demasiadamente simples.

A Tabela 1 traz um resumo comparativo com as características de cada trabalho relacionado e da presente proposta.

Trabalho	**Abordagem**	**Bancos de Dados**	**Estratégia de Modelagem**	**Estratégia de Elasticidade**	**Análise de Cargas de Trabalhos**
Ganapathi et al.	Modelagem do Desempenho	Relacional Centralizado	Aprendizado	Não se aplica	Não, apenas consultas isoladas
Mozafari et al.	Modelagem do Desempenho	Relacional Centralizado	Aprendizado e analítico	Não se aplica	Heterogêneas
Didona; Romano et al.	Modelagem do Desempenho	Chave-valor transacionais parcialmente replicados	Aprendizado e analítico	Não se aplica	Heterogêneas
Xiong et al.	Modelagem do Desempenho e elasticidade	Relacional Distribuído	Aprendizado	Otimização Matemática	Homogêneas
Cruz et al.	Elasticidade	NoSQL	Não se aplica	Regras estáticas	Homogêneas
Konstantinou et al.	Elasticidade	NoSQL	Não se aplica	Aprendizado por reforço	Homogêneas
Kassela et al.	Elasticidade	NoSQL	Não se aplica	Aprendizado por reforço	Heterogêneas
Farias et al.	Modelagem do Desempenho e elasticidade	NoSQL	Aprendizado	Otimização matemática	Heterogêneas

Tabela 1 – Análise Comparativa entre os Trabalhos Relacionados

3.3 CONCLUSÃO

Os principais trabalhos relacionados ao tema desta dissertação foram apresentados neste Capítulo. Descrevemos as principais características de cada trabalho, assim como seus pontos positivos e negativos. Por fim, comparamos a nossa proposta com aquelas presentes nos trabalhos relacionados. Muitos dessas propostas não tratam cargas de trabalhos heterogêneas ou não abordam Bancos de Dados NoSQL em nuvem, o que justifica a proposição do trabalho dessa dissertação.

4 ABORDAGEM PARA MODELAGEM DE DESEMPENHO E ELASTICIDADE PARA BANCOS DE DADOS NOSQL

4.1 INTRODUÇÃO

Os bancos de dados NoSQL estão integrando ambientes distribuídos de processamento de dados que executam cargas de trabalhos concorrentes e heterogêneas compostas por requisições. Ao mesmo tempo, esses sistemas precisam satisfazer as expectativas de desempenho definidas no SLA. Nesse ambiente, a previsibilidade de desempenho pode ajudar a evitar potenciais violações do SLA, *i.e.*, a habilidade de estimar o impacto da execução concorrente de requisições em uma carga de trabalho em execução (DUGGAN et al., 2011). Trabalhos teóricos (GRAY et al., 1996) sugerem que aspectos de concorrência e distribuição podem degradar o desempenho de maneira não-linear em relação ao tamanho do *cluster* e ao volume de requisições. Além disso, o ambiente em nuvem apresenta alta variação de desempenho (SCHAD et al., 2010). Por isso, são necessários mecanismos para gerenciar a incerteza advinda da alta variação de desempenho para se oferecer uma modelagem de desempenho mais completa.

O provisionamento de recursos é um ponto chave para permitir que os provedores de serviço atendam ao SLA enquanto maximizam a utilização da infraestrutura subjacente alocada. Provedores de nuvem precisam evitar tanto o *underprovisioning*, que provoca lentidão no serviço, como o *overprovisioning* que acarreta em gastos desnecessários de infraestrutura (SANTOS et al., 2013). Técnicas de provisionamento automático são projetadas para controlar tanto flutuações no volume de requisições da carga de trabalho quanto mudanças em sua composição, visando evitar penalidades contratuais. Nesse cenário, mecanismos de provisionamento automático baseados em modelos de desempenho seriam capazes de oferecer uma alocação de recursos confiável.

Deste modo, este trabalho propõe uma abordagem de modelagem de desempenho baseada no SLA para sistemas de bancos de dados NoSQL. Essa abordagem leva em consideração cargas de trabalho heterogêneas e é capaz de capturar os efeitos não-lineares dos aspectos de concorrência e distribuição sobre o desempenho. Esta abordagem também oferece um mecanismo para modelar a variação do desempenho.

Este trabalho também propõe uma estratégia de provisionamento elástico automático baseado em modelos de desempenho, que adiciona e remove recursos em tempo de execução de acordo com as características da carga de trabalho corrente. Adicionalmente, essa abordagem de provisionamento automático faz proveito do gerenciamento da incerteza produzidos pelos modelos de desempenho. A seguir são apresentadas as técnicas desenvolvidas em cada abordagem.

4.2 VISÃO GERAL

Este trabalho lida com cargas de trabalhos heterogêneas, i.e., cargas de trabalhos que possui tipos de requisições diferentes (leitura e escrita) e complexidades diferentes (consultas por igualdade, consultas por intervalo, atualizações e inserções). Nós agrupamos manualmente as

requisições da carga de trabalho em classes. Cada classe contém requisições que são do mesmo tipo (leitura ou escrita) e possuem complexidade semelhante. Em particular, a carga de trabalho usada é gerada pelo *benchmark* YCSB.

O YCSB produz quatro tipos de requisições: *read*, *scan*, *update* e *insert*. Desse modo, foi executado um agrupamento manual das requisições do YCSB em quatro classes, de modo que cada classe corresponde a um tipo de requisição. Note-se que, apesar deste trabalho considerar uma carga de trabalho específica, as discussões sobre cargas de trabalho ao longo desse trabalho podem ser entendidas de forma mais genérica. Este trabalho foi dividido em duas abordagens:

1) **Modelagem do desempenho:** constrói um modelo ou função de desempenho que recebe as características da carga de trabalho e as características do *cluster* como entrada para estimar o desempenho do serviço nesse cenário por meio de algoritmos de aprendizado de máquina.

2) **Provisionamento:** monitora a carga de trabalho e calcula, dinâmica e automaticamente, a alocação ótima de acordo com as características da carga de trabalho corrente ao longo da execução do sistema.

4.3 MODELAGEM DO DESEMPENHO

O objetivo desta etapa é construir um modelo ou função de desempenho *M* que recebe as características da carga de trabalho e do *cluster* e estima o desempenho do serviço nesse cenário por meio de algoritmos de aprendizado de máquina. Este trabalho utiliza três métricas de desempenho: vazão (quantidade de requisições executadas por segundo), tempo de resposta médio (tempo de resposta médio das requisições executadas a cada segundo) e violações por segundo (quantidade de requisições cujo tempo de resposta ultrapassa limita pré-definido no SLA). Estas métricas são intuitivas do ponto de vista dos *stakeholders* e refletem, de um modo geral, a experiência de uso da aplicação pelo usuário. Não foram utilizadas métricas relativas a custos de infraestrutura. Contudo, aspectos de custos podem ser calculadas utilizando o modelo proposto por (SOUSA et al., 2012).

As características extraídas da carga de trabalho são relativas às classes da carga de trabalho (*read*, *scan*, *insert* e *update*). Para cada classe, foi extraída uma característica (sec 4.3.1.2) que representa o volume de requisições dessa classe que é gerado pelo *benchmark*. Vale ressaltar que, nesse caso, o termo característica se refere àquele comumente usado no campo de aprendizado de máquina que, em geral, é chamado de *feature*.

Além disso, o modelo *M* considera o tamanho do *cluster* pois tamanhos maiores de *clusters* indicam maior poder computacional e, consequentemente, indicam melhora no desempenho. Adicionalmente, este trabalho tem como objetivo ser genérico em relação ao SGBD utilizado. Sistemas NoSQL são projetados para executar de forma distribuída usando um grande *pool* de recursos. O modelo de particionamento e gerenciamento de dados são específicos do SGBD utilizado. Por isso, foi extraída apenas a caraterística do tamanho do *cluster*, a quantidade de nós trabalhadores, visto que está é uma das únicas características que

são comuns a todos sistemas NoSQL e que tem grande impacto no desempenho.

O desempenho de um banco NoSQL distribuído pode ser afetado por diversos fatores. Nesse trabalho foram tratados três aspectos que contribuem para a alteração do desempenho:

1. Complexidade das consultas. Cada requisição exige um volume diferente de operações de disco. Isso faz com que, para retornar o resultado em tempo hábil, uma classe de requisições demande mais recursos computacionais do que outra classe;
2. Concorrência. A execução concorrente de requisições pode causar uma interferência por contenção de lock (MOZAFARI et al., 2013) o que acarreta na degradação do desempenho por esperas e *deadlocks* entre as classes de requisições da carga de trabalho; e
3. Distribuição. Muitos bancos NoSQL empregam estratégias de replicação para aprimorar a disponibilidade e confiabilidade. Contudo, isso implica no custo adicional de manter a consistência. Em geral, estratégias de consistência usam *locks* distribuídos o que causa esperas e *deadlocks* distribuídos. O trabalho (GRAY et al., 1996) sugere que a quantidade de *deadlocks* aumenta quadraticamente em relação ao número de nós no sistemas e ao volume de requisições.

O modelo de desempenho M é construído utilizando algoritmos de aprendizado de máquina. Especificamente, o problema de correlacionar um métrica de desempenho a partir das características da carga de trabalho e do tamanho do *cluster* é tratado como um problema de regressão, pois a variável alvo (métrica) a ser predita é uma variável contínua. O conjunto de dados de treinamento das técnicas de aprendizado de máquina é gerado em um ambiente de teste controlado, onde é possível executar várias configurações de carga de trabalho sob demanda, o que caracteriza a abordagem de modelagem de desempenho como *offline*. Nossa abordagem é dividida em três grandes etapas como ilustrado na Figura 9:

1. *Geração do Conjunto de Dados de Desempenho*. Técnicas de aprendizado de máquina necessitam de um conjunto de observações (*i.e.* conjunto de treinamento) de um determinado fenômeno para construir um modelo preditivo. Essa etapa visa produzir um conjunto treinamento base para ser usado posteriormente em técnicas de aprendizado de máquina e divido em duas subetapas: Medição do desempenho e Filtragem e Agregação;
2. *Transformação das Características*. Como citado anteriormente, o comportamento do desempenho pode estar correlacionado de forma não-linear com as características da carga de trabalho e do tamanho do *cluster*. Essa fase apresenta uma manipulação das características do conjunto de dados base para melhorar a qualidade dos modelos preditivos produzidos por técnicas de aprendizado de máquina; e
3. *Seleção e Avaliação de Modelos*. Técnicas de aprendizado de máquina apresentam vários parâmetros a serem ajustados. Essa fase se concentra na escolha da melhor configuração de parâmetros e em mostrar como se mede a acurácia de um modelo preditivo. A saída dessa fase é o modelo de desempenho acurado para uma dada métrica e desempenho.

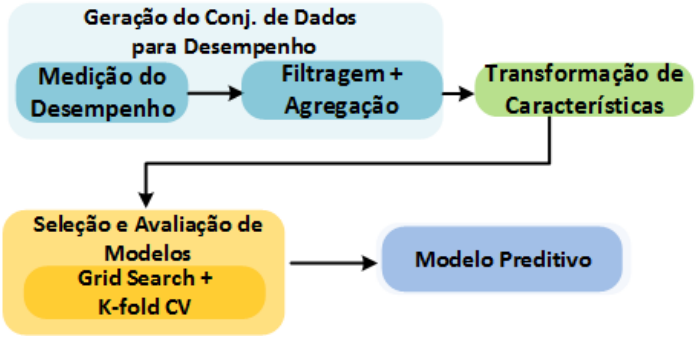


Figura 9 – Visão geral da proposta.

4.3.1 Geração do Conjunto de Dados de Desempenho

O conjunto de treinamento desempenha um papel crucial na acurácia de técnicas de aprendizado de máquina. Conjuntos de treinamento com pequena quantidade de amostras ou que apresentem intervalos limitados de características podem gerar modelos preditores com baixo poder de generalização e extrapolação. Além disso, um conjunto de treinamento de desempenho pode conter ruído, o que induz as técnicas de aprendizado de máquina a criar modelos preditivos menos acurados.

A geração desse conjunto de dados é dividida em duas subetapas: (1) Medição do Desempenho e (2) Agregação e filtragem.

4.3.1.1 Medição do desempenho

Para se entender os impactos das características da carga de trabalho e do tamanho do *cluster* sobre o desempenho, foram executados experimentos iniciais para medir o desempenho do SGBD sob diversos cenários. Esses experimentos são executados em um ambiente de teste separado do sistema de produção, de modo *offline*, utilizando o *benchmark* YCSB.

O YCSB oferece vários parâmetros que permitem mudar o padrão de acesso, o volume e a composição da carga de trabalho. Dentre todos parâmetros, foram selecionados os parâmetros que mais contribuem para a variação do desempenho e que apresentam dinâmicas similar a cargas de trabalho reais. Os parâmetros variados foram os seguintes: *target*, um inteiro que representa a quantidade de requisições por segundo geradas pelo *benchmark* e *mix*, um vetor $[p_{read}, p_{scan}, p_{update}, p_{insert}]$ onde cada elemento representa a porcentagem total de requisições de cada tipo de operação. O parâmetro p_{read} representa a porcentagem de requisições de leitura por igualdade, p_{scan} representa a porcentagem de requisições de leitura por intervalo, p_{update} representa a porcentagem de requisições de atualização e p_{insert} representa a porcentagem de requisições de inserção. Note-se que *target* e *mix* afetam de forma direta o volume e a composição da carga de trabalho, respectivamente.

Dessa forma, é necessário definir o conjunto de parâmetros relativos à carga de trabalho e ao tamanho do *cluster* que serão empregados nos testes. Vale lembrar que o intervalo usado para teste de cada parâmetro impacta diretamente na capacidade de generalização do

modelo gerado. Por exemplo, no caso dos experimentos serem realizados com parâmetros *target* cujos valores pertençam ao intervalo $(4000, 6000)$, espera-se que o modelo seja menos acurado para valores de *target* menores que 4000 e maiores que 6000. A Tabela 2 mostra os valores de teste selecionados para cada parâmetro.

Categoria	**Parâmetro**	**Valores**
Carga de trabalho	p_{read}	0%,20%,40%,60%,80%,100%
	p_{scan}	0%,20%,40%,60%,80%,100%
	p_{insert}	0%,20%
	p_{update}	0%,20%
	target	1000,2000,3000,4000,5000, 6000,7000,8000,9000,10000
Cluster	tamanho	2,3,4

Tabela 2 – Tabela de valores de cada parâmetro para experimentos

Executamos um experimento com duração de 360 segundos para cada combinação de parâmetros da Tabela 2. Note que nem todas as combinações são possíveis pois, por se tratarem de valores percentuais, $p_{read} + p_{scan} + p_{update} + p_{insert} = 1$. Dessa forma, foi adotado um total de 20 combinações de $mix = [p_{read}, p_{scan}, p_{update}, p_{insert}]$. Além disso, existem 10 possibilidades para *target* e 3 possibilidades para tamanho do *cluster*. Logo, tem-se o total $20x10x3 = 600$ experimentos.

Em cada experimento, é gerado um arquivo de *log* que contém informações relativas às requisições processadas pelo sistema. A cada segundo foram coletadas as métricas de desempenho para a carga de trabalho:

1. O número de requisições executadas (vazão) no último segundo;
2. O tempo de resposta médio das requisições executadas no último segundo; e
3. A quantidade de violações de SLA por segundo.

4.3.1.2 Filtragem e agregação

Os dados brutos contidos no *log* não podem ser utilizados diretamente para realizar as análises preditivas pelos métodos de regressão devido ao grande volume de informação, visto que foram 600 experimentos e a cada segundo uma entrada é gravada no *log*. Por isso, os *logs* gerados são agregados como de forma a produzir um conjunto menor de dados. Estes dados foram formatados de tal sorte a serem utilizados como entrada nas técnicas de aprendizado de máquina. Estas técnicas requerem um conjunto de dados que é composto por duas partes: uma matriz de características contendo as amostras do fenômeno a ser modelado e um vetor de valores de saída referente a cada amostra da matriz de características.

Como o objetivo final é modelar o desempenho baseada nas três métricas de tempo de resposta médio por segundo, vazão e violações por segundo, cada métrica foi tratada individualmente, i.e. foi gerado um conjunto de dados e um modelo preditivo para cada métrica.

Para cada métrica de desempenho foi gerado um conjunto de dados (*i.e.* matriz de características e vetor de valores de saída). Cada experimento executado na Seção 4.3.1.1 é representado como uma linha da matriz de características, também chamada de vetor de características. Esse vetor é composto pelos parâmetros da carga de trabalho e tamanho do *cluster* utilizados em um dado experimento. Assim, esse vetor contém 6 posições <*DB*, t ,t_{read}, t_{scan}, t_{insert}, t_{update}> onde *DB* é o tamanho do *cluster* no experimento, t é a quantidade de requisições alvo que são emitidas ao sistema (parâmetro *target*) e t_{tipo} é a quantidade alvo de requisições de um tipo específico que são emitidas ao sistema, e.g, t_{read} é a quantidade alvo de requisições *read* por segundo, o que equivale ao parâmetro *target* multiplicado por p_{read}.

O valor de saída para esse vetor corresponde ao valor resumido de uma dada métrica de desempenho. Esse valor é obtido por meio da agregação dos dados filtrados do *log* desse experimento. Primeiramente, o primeiro minuto de experimento que corresponde ao aquecimento do *cache* é descartado. Em seguida, são removidos as medições que são maiores que o 80° percentil e menores que o 20° percentil. Isso serve para retirar o ruído causado por efeitos inesperados de escalonamento e interferência de outros clientes que rodam na mesma nuvem. Finalmente, o valor resumido corresponde à média das medidas do *log* filtrado.

4.3.2 Transformação das características

É importante que os modelos gerados pelas técnicas de regressão a partir do conjunto de treinamento tenham poder de generalização e extrapolação. Essa é uma tarefa difícil pois o modelo deve capturar diversos aspectos como os citados na Seção 4.3 que afetam as métricas de desempenho e geram modelos de desempenho não-lineares.

De modo mais específico, (GRAY et al., 1996) sugere de forma teórica que para modelos de replicação mestre/escravo o número de nós no *cluster* e o número de requisições impacta o desempenho de forma quadrática devido à contenção de *locks* gerada pela execução concorrente de requisições no mesmo SGBD relacional distribuído. Esse trabalho estende esse conceito para SBGDs NoSQL.

Contudo, SBGDs NoSQL podem ter diversos modelos de distribuição de dados como replicação, fragmentação ou híbrido replicação/fragmentação. Vários desses modelos podem ser mais eficientes e escaláveis que a replicação mestre/escravo. Assim, em geral, esses modelos de distribuição de dados são impactados quadraticamente ou menos em relação ao número de nós e de requisições emitidas. Logo, são necessárias técnicas de regressão que sejam capazes de capturar esses aspectos.

Existem métodos que são capazes de capazes de modelar qualquer tipo de relações não lineares entre as características e o vetor de valor alvo como os métodos da categoria *boosting* como o *Gradient Boosting Machine* mas esse métodos carecem de poder de extrapolação (LOH et al., 2007). Por isso, existem outros métodos mais indicados para extrapolação como os métodos de regressão linear. No entanto, esses métodos são capaz de capturar apenas relações lineares entre as características e os vetor de saída. Nesse caso, é necessário uma manipulação das características para permitir que a regressão linear modele as relações não-lineares.

Por tal, este trabalho propõe duas manipulações de características que serão posteriormente testadas com uma variedade de métodos de regressão:

1) **Características lineares:** O conjunto de dados base gerado como descrito na Seção 4.3.1.2 é fornecido diretamente aos métodos de regressão. Essa transformação mais indicada para métodos que conseguem capturar superfícies arbitrárias como os métodos baseados em *boosting*. Para métodos de regressão linear, as características t , t_{read} , t_{scan} , t_{update} , t_{insert} contam para a contribuição de cada classe da carga de trabalho para o desempenho (aspecto 1, seção 4.3). E *DB* representa o ganho ou perda de desempenho ao se aumentar ou diminuir o tamanho de um *cluster*.

2) **Características quadráticas:** Para modelar o efeitos quadráticos das características no valor de saída, o conjunto de dados base gerado na fase 4.3.1.2 foi modificado. Em complemento às características originais $< BD, t, t_{read}, t_{scan}, t_{update}, t_{insert} >$, foram adicionadas para cada amostra as seguintes características:

- 15 características resultantes do produtos de todos os pares distintos das características de composição (t, t_{read}, t_{scan}, t_{update}, t_{insert}) incluindo cada termo ao quadrado: $<t^2, t \times t_{read}, t \times t_{scan}, ...>$. Essas características expressam a degradação do desempenho causado pelos aspectos de concorrência (aspecto 2, Seção 4.3).
- 5 características resultantes dos produtos da característica *DB* com as características de composição t (t, t_{read}, t_{scan}, t_{update}, t_{insert}) : $<t, t_{read}, t_{scan}, t_{insert}, t_{update}>$. Essas características representam o decaimento do desempenho por esperas e *deadlocks* distribuídos para cada classe da carga de trabalho. (aspecto 3, Seção 4.3)

4.3.3 Seleção e Avaliação de Modelos

Em geral, algoritmos de aprendizado de máquina tem vários hiperparâmetros de entrada. Por exemplo, o método de regressão linear com regularização tem o hiperparâmetro coeficiente de regularização para controlar a magnitude dos coeficientes. Desse modo, uma escolha inapropriada desses hiperparâmetros pode acarretar na geração de modelos preditivos de baixa qualidade. O problema de selecionar a melhor configuração de hiperparâmetros de entrada é conhecido como otimização de hiperparâmetros. Em um algoritmo de seleção de hiperparâmetros é necessário escolher quais modelos preditivos são mais acurados. A seguir é descrito a técnica usada para avaliar a acurácia do modelo e para a otimização de hiperparâmetros.

4.3.3.1 Avaliação do modelo

Para avaliar a acurácia de um modelo, foi utilizada a validação cruzada *K-fold* tradicional. Nessa técnica, o conjunto de dados é aleatoriamente dividido em k subconjuntos de tamanhos iguais. Assim, treinou-se os métodos de aprendizado de máquina com $k-1$ subconjuntos e o erro é calculado a partir do subconjunto restante. Esse processo é feito k vezes, uma vez para cada conjunto de treino composto por $k-1$ subconjuntos.

Como métrica de erro foi usado o coeficiente de determinação R^2 que é dado por:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(p_i - y_i)^2}{\sum_{i=1}^{N}(y_{media} - y_i)^2}$$

Onde y_i é o valor de saída real da i-ésima observação do conjunto de teste e p_i denota o valor predito pelo modelo preditivo para essa observação. Note que valores mais próximos de 1 da métrica R^2 indicam que as predições são quase perfeitas. Por outro lado, R^2 pode assumir valores arbitrariamente negativos assinalando predições de baixa acurácia.

4.3.3.2 Otimização de hiperparâmetros

A escolha correta dos hiperparâmetros de entrada dos algoritmos de aprendizado de máquina é feita por meio do método *grid-search* associado com a validação cruzada *K-fold*. O *grid-search* consiste em uma busca exaustiva sobre um conjunto de configurações de hiperparâmetros manualmente definidos. Desse modo, é construído um modelo preditivo para cada configuração de hiperparâmetros que é avaliado posteriormente usando a validação cruzada *K-fold*.

Em nosso caso, nós analisamos dois métodos de regressão: regressão linear e *Gradient boosting machine*. A Tabela 3 traz os parâmetros definidos para cada métodos a serem usados pela técnica *grid-search*.

Método	**Parâmetro**	**Valores**
Regressão linear regularizada	Coeficiente de regularização	0.0,1.0,5.0,10.0,50.0,100.0, 200.0,500.0,1000.0,5000.0
Gradient Boosting Machine	Taxa de aprendizado	0.01,0.02, 0.05, 0.1
	Altura máxima da árvore	4,6
	Quantidade de modelos base	10,30,50,70,100,1000

Tabela 3 – Tabela de valores fornecido à técnica *grid-search*

4.3.4 Gerenciamento da Incerteza

Até este ponto foram construídos modelos que são capazes de predizer o comportamento médio de uma métrica de desempenho para uma determinada configuração de características de carga de trabalho e tamanho de *cluster*. Contudo, a métrica de desempenho apresenta ruído para características de carga de trabalho e tamanho de *cluster* fixas o que gera uma incerteza na predição da métrica. Para ilustrar, a Figura 10 mostra a quantidade de violações por segundo (VPS) em um experimento simples de 600 segundos em que executamos o *benchmark* YCSB com uma carga com parâmetro *target* fixo em 1000 e parâmetro $mix = [p_{read}, p_{scan}, p_{insert}, p_{update}]$ fixo em $[40\%, 60\%, 0\%, 0\%]$. A média de violações nesse experimentos é 192.35 violações por segundo com um desvio padrão de 35.94.

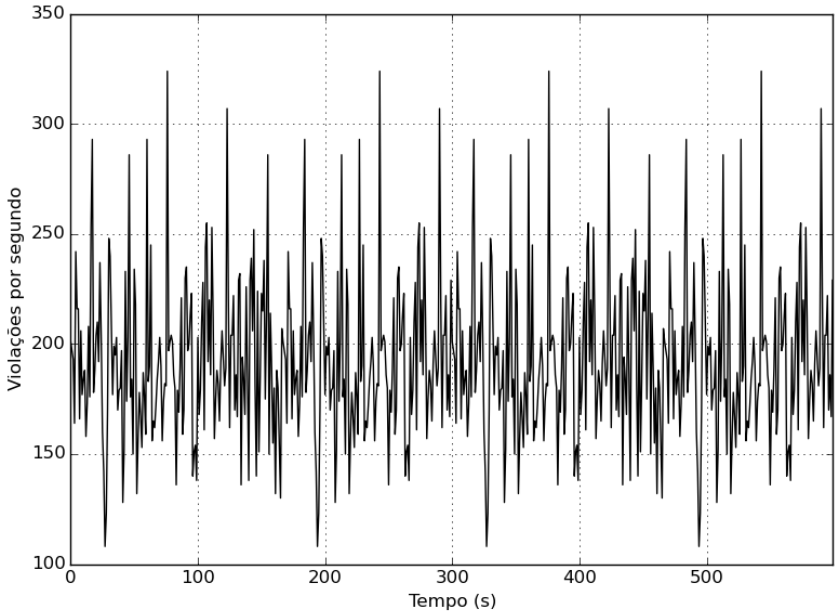


Figura 10 – Ruído sobre a métrica VPS.

Alguns mecanismos de QoS podem ser induzidos ao erro por não considerar o ruído das métricas de desempenho ao longo do tempo. Por isso, esses mecanismos devem considerar este ruído para entregar ações de manutenção de QoS mais confiáveis. Desse modo, é necessária uma modelagem do desempenho que contemple o ruído, quantificada pelo desvio-padrão, de uma dada métrica nos experimentos.

Por meio de experimentos, também foi possível identificar que o desvio-padrão muda de acordo com o tamanho do *cluster* e com as características da carga de trabalho. Para tratar esta questão, propomos a construção de modelos de incerteza que estimam o desvio-padrão também por meio de técnicas de aprendizado de máquina da categoria de métodos de regressão, pois a variável a ser predita é contínua e tem como entrada também as características da carga de trabalho e o tamanho do *cluster*. O desvio-padrão foi utilizado como medida de ruído.

Assim como feito anteriormente, um conjunto de treinamento precisa ser fornecido para que algoritmos de aprendizado de máquina gerem modelos preditivos. Os experimentos descritos na Seção 4.3.1.1 foram reutilizados e geramos um conjunto de dados (matriz de características e vetor de valores de saída) relativo às medidas de desvio-padrão obtidas nos experimentos. A matriz de características é construída como descrito na Seção 4.3.1.2. No entanto, o valor de saída relativo a cada experimento é construído de modo diferente. Nesse caso, esse valor representa o desvio-padrão da métrica de desempenho do experimento correspondente. Assim, os primeiros 60 segundos desse experimento foram descartados e calculou-se o desvio padrão amostral da métrica nos 300 segundos finais.

Com isso, é possível aplicar uma gama de algoritmos de aprendizado de máquina diretamente sobre o conjunto de dados para gerar modelos de incerteza e posteriormente escolher quais modelos apresentam melhor capacidade preditiva. Esse processo é idêntico ao descrito na Seção 4.3.3.

4.4 PROVISIONAMENTO ELÁSTICO

Provedores de nuvem precisam se adequar ao exigências do SLA enquanto maximizam seus lucros. Logo, é crucial gerenciar os recursos de modo a evitar o *overprovisioning* e o *underprovisioning* por meio da adição e remoção de recursos de forma dinâmica. Nessa fase foi proposto um problema de otimização matemática que faz uso do modelo de desempenho *M* construído na fase anterior para verificar a necessidade de alguma ação de provisionamento e executá-la caso seja necessário. Esse modelo recebe as características da carga de trabalho corrente e computa a alocação ótima para essa carga com objetivo de atender ao SLA e minimizar os custos de infraestrutura. Também é apresentada uma forma para aproveitar a modelagem da incerteza feita na fase de modelagem de desempenho para oferecer um provisionamento mais confiável. A estratégia de provisionamento elástico é composta por dois componentes: monitoramento e provisionamento.

4.4.1 Monitoramento

O sistema de monitoramento obtém informações sobre a carga de trabalho de entrada no sistema periodicamente. Considerou-se um intervalo de monitoramento de 1 segundo. A cada segundo, é coletada a quantidade total de requisições que chegam ao sistema (*target*) e a quantidade de requisições por classe ($target_{read}$, $target_{scan}$, $target_{update}$ e $target_{insert}$). Vale notar que essas informações podem ser coletadas de forma não-intrusiva, pois é necessário apenas capturar a consulta e seu respectivo tipo.

4.4.2 Provisionamento

Com os dados provenientes do sistema de monitoramento, a nossa estratégia de provisionamento verifica periodicamente se alguma ação de provisionamento é necessária e, em caso positivo, calcula qual deve ser a nova alocação de recursos. O período em que esse processo acontece é definido como ciclo de provisionamento.

Dado isso, buscou-se meios de computar a alocação de recursos corretas para se adequar ao requerimentos de SLA (evitar *underprovisioning*) e minimizar a quantidade de recursos alocados para maximizar o lucro do provedor de serviço (i.e. evitar *overprovisioning*). Neste caso, o SLA é definido como a proporção máxima (em porcentagem) de requisições por segundo que ultrapassam o limite pré-definido de tempo no SLA, denominado SLA_{DB}.

Desse modo, propomos um problema de otimização (4.1) que recebe as dados de monitoramento e é capaz de computar a nova alocação de recursos baseado nos critérios citados anteriormente. Em cada ciclo de provisionamento esse problema é resolvido para com a finalidade de gerar uma nova alocação de recursos.

Esse problema é baseado em uma função de custo *C* a ser otimizada que se refere aos custos incorridos, seja de infraestrutura ou de penalidades do SLA, em um certo estado do sistema (carga de trabalho de entrada e tamanho do *cluster*). Além de permitir a adição de restrições sobre a adequação do SLA para esse estado do sistema.

Nesse trabalho, a função de custo considerada é baseada nos custos de infraestrutura. Como consideramos recursos homogêneiso e os custos de infraestrutura são proporcionais à quantidade de nós alocados, definimos C como $C(DB) = DB$ onde DB é a quantidade de nós na nova alocação, assim cada máquina virtual é representada como uma unidade de custo. Em relação ao atendimento do SLA, foi adicionado uma restrição ao problema de otimização que faz uso de um modelo M_{VPS} de desempenho construído na fase de modelagem de desempenho que é capaz de predizer a quantidade de violações por segundo para expressar que a nova alocação deve atender o SLA.

$$\begin{aligned} \underset{DB}{arg\,\min} \quad & DB \\ \text{s.a.} \quad & M_{VPS}(DB, T) < SLA_{DB} \times t \end{aligned} \tag{4.1}$$

Onde M_{VPS} recebe o tamanho de *cluster DB* da nova alocação de recursos e um vetor T de características da carga de trabalho (t, t_{read}, t_{scan}, t_{update}, t_{insert}). O termo $SLA_{DB} \times t$ representa a quantidade absoluta máxima de violações permitidas nesse cenário. Assim, para evitar a alocação errônea de recursos por flutuações na carga de trabalho, assumiu-se que cada característica da carga de trabalho é constante e igual à média das medidas para cada segundo durante o último ciclo de provisionamento.

Para resolver o problema de otimização 4.1 onde DB é a variável de otimização utilizou-se regressão linear para produzir M_{VPS}. Esse formulação é caracterizada como um caso de programação inteira pois DB é um número inteiro e a restrição é linear. Esse problema é conhecidamente NP-Completo no caso geral. Contudo, formulação proposta tem-se apenas uma variável de otimização, o que simplifica a solução.

Dessa forma, foi utilizado uma meta-heurística busca exaustiva para computar o DB ótimo. A busca exaustiva é baseada em testar cada valor no espaço de busca e escolher o valor que minimize a função objetivo e satisfaça as restrições. Note que DB só pode assumir valores positivos e, além disso, considerou-se *clusters* de tamanho pequenos e médios. Dessa forma, testar todo o espaço de busca quando limita-se DB entre 1 e 50 é um processo computacionalmente rápido. Além disso, pode-se garantir que o ótimo global é encontrado pois a função objetivo é monotonicamente crescente com respeito a DB.

Ainda existe a questão de que o modelo M_{VPS} possa não ser acurado para valores grandes dos parâmetros DB e T. Todavia, note que é possível lidar com esse problema retroalimentando ou retreinando o modelo considerando os dados de monitoramento do sistema em produção.

Após a resolução do problema de otimização, tem-se o novo valor de DB que corresponde a nova alocação de recursos. Com isso, calcula-se a ação de provisionamento necessária para atingir a nova alocação. O número de nós a serem adicionados ou removidos é dado pela diferença entre a quantidade de nós atualmente alocados e a quantidade de nós na nova alocação. Caso essa diferença seja positiva, a ação de provisionamento é a adição dessa quantidade de nós e, caso seja negativa, removemos essa quantidade de nós. Em geral, os SGBDs NoSQL são projetados para executar na nuvem e já tem implementada a funcionalidade de

adicionar ou remover recursos computacionais. Dessa forma, essa funcionalidade foi utilizada, delegando ao SGBD a ação de adicionar ou remover nós.

4.4.2.1 Provisionamento com Gerenciamento da Incerteza

A formulação 4.1 baseia suas ações de provisionamento no desempenho médio esperado do sistema na nova alocação. No entanto, este desempenho apresenta uma incerteza, i.e., um ruído, como mostrado na Figura 10, que pode gerar violações inesperadas. Por isso, foi elaborado um modelo de incerteza construído na fase de modelagem de desempenho para enriquecer abordagem proposta.

Desse modo, também almejamos adicionar mais controle sobre a nossa abordagem de elasticidade definindo níveis de confiança para adequação ao SLA. Introduzimos o parâmetro α ($0 < \alpha < 1$) definido pelo usuário que representa a confiança (i.e. probabilidade) da alocação computada pela nossa estratégia atender às demandas de SLA. Na prática, o objetivo é garantir que pelo menos $100 \times \alpha\%$ das medições da métrica de desempenho VPS sejam menores do que $SLA_{DB} \times t$.

Para isso, projetamos um modelo composto $\hat{M}$ baseado no modelo de desempenho para a métrica VPS M_{VPS} e no modelo de incerteza para a métrica VPS M^{σ}_{VPS}. $\hat{M}(T, DB, \alpha)$ tem 3 parâmetros de entrada: 1) vetor de características da carga de trabalho T, 2) tamanho do *cluster DB* e a confiança α e retorna um número de violações por segundo que é maior do que $100 \times \alpha\%$ das medições para esse cenário.

Considera-se que o ruído nas métricas de desempenho seguem uma distribuição Gaussiana. A distribuição Gaussiana é uma escolha bastante comum pois possui a capacidade de modelar o efeito combinado de diversas fatores externos conforme verificado por meio do teorema central do limite (KNILL, 1994). Neste trabalho, existe uma combinação de fatores como interferência de rede ou interferência de outro inquilinos na mesma máquina físicas em que as máquinas do sistema testado está implantada. Adicionalmente, a distribuição Gaussiana é interessante pois é completamente definida por dois parâmetros: média e desvio padrão. Especificamente, o ruído é definido como um distribuição normal com média dada por M_{VPS} e desvio padrão dado por M^{σ}_{VPS}. Desse modo, definimos $\hat{M}(T, DB, \alpha)$ mais precisamente como:

$$\hat{M}(T, DB, \alpha) = P^{-1}(\alpha, M_{VPS}(T, DB), M^{\sigma}_{VPS}(T, DB)) \tag{4.2}$$

Onde P^{-1} é função inversa da função distribuição acumulada da distribuição normal com média $M_{VPS}(T, DB)$, desvio padrão $M^{\sigma}_{VPS}(T, DB)$ e probabilidade α. A função P^{-1} tem a capacidade de retornar z tal que a métrica VPS está dentro do intervalo $(-\infty, z)$ com probabilidade α. Com isso, propomos um segundo problema de otimização matemática usando a nova função $\hat{M}$:

$$\begin{aligned} \underset{DB}{arg\,\min} \quad & DB \\ \text{s.a.} \quad & \hat{M}(T, DB, \alpha) < SLA_{DB} \times t \end{aligned} \tag{4.3}$$

O parâmetro α fornece um meio do usuário controlar o quão conservador ele é na em suas decisões de provisionamento. Valores maiores e α indicam decisões mais seguras enquanto valores baixos de α indicam decisões mais arriscadas.

4.5 CONCLUSÃO

Nesse capítulo foi apresentada uma abordagem de modelagem de desempenho e elasticidade para bancos de dados NoSQL. A modelagem de desempenho é baseada em técnicas de aprendizado de máquinas onde, inicialmente, foram executados experimentos para gerar o conjunto de observações de desempenho. Em seguida foram propostos dois modos de manipulação das características para melhorar a acurácia dos modelos gerados: características lineares e características quadráticas. Foi discutido sobre a acurácia dos modelos gerados e qual cenário é mais vantajoso usar características lineares e quadráticas. Também foi proposto uma abordagem para o problema de elasticidade que apresenta uma formulação de otimização e faz uso dos modelos preditivos gerados pela fase de modelagem de desempenho para calcular a nova alocação de recursos.

5 AVALIAÇÃO EXPERIMENTAL

Foram realizados experimentos para avaliar a efetividade das abordagens de modelagem de desempenho e de elasticidade. Para tanto, cada umas das abordagens foi analisada separadamente. Os objetivos específicos da avaliação experimental de cada abordagem são discutidos nas seções seguintes.

5.1 INFRAESTRUTURA

Foi implementado um protótipo das abordagens de modelagem de desempenho e de elasticidade usando as linguagens Python e Bash Script juntamente com a automatização dos experimentos da seção 4.3.1.1. Para os métodos de aprendizado de máquina foi utilizado as implementações disponíveis na biblioteca Scikit-learn (PEDREGOSA et al., 2011).

Como citado anteriormente, o *bechmark* YCSB versão 0.5.0 foi utilizado nos experimentos com os seguintes parâmetros: 40 threads de clientes, 1000000 tuplas na base dados, padrão de acesso uniforme e 1 byte como tamanho de cada campo da tupla. Os parâmetros $target$ (quantidade de requisições submetidas ao sistema por segundo) e $<p_{read},p_{scan},p_{insert},p_{update}>$ (proporção de cada tipo de requisição submetida ao sistema) são variados conforme descritos nos experimentos a seguir. Os parâmetros restantes foram mantidos com os valores padrões. Entre cada experimento a base de dados foi excluída e carregada novamente. O SGBD MongoDB (INC, 2013) versão 3.0.8 foi utilizado em seu modo replicação.

Esse protótipo foi implantado e testado na nuvem pública Amazon EC2. O YCSB foi executado em um máquina *m3.xlarge* por possuir alta vazão de rede. Máquinas *t2.small* foram utilizadas para os nós de armazenamento.

5.2 AVALIAÇÃO DA MODELAGEM DE DESEMPENHO

A abordagem da modelagem de desempenho visa predizer as métricas utilizando como base a carga de trabalho e no tamanho do *cluster* por meio da construção de modelos de aprendizado de máquina. Assim, os experimentos específicos dessa abordagem foram projetados para atingir os seguintes objetivos:

- Avaliar o poder preditivo dos modelos de desempenho e incerteza gerados: este objetivo feito realizado por meio da validação cruzada *k-fold* adotando a métrica R^2.
- Investigar a complexidade subjacente das diferentes métricas de desempenho.
- Determinar em quais métodos é mais interessante adotar as características lineares (CL) ou quadráticas (CQ) para os modelos de desempenho.

5.2.1 Resultados experimentais

Os resultados dos métodos são expressos em termos da métrica R^2 em conjunto com a validação cruzada 10-fold com o *grid-search* no parâmetros de entrada do método. Os

resultados de dois métodos foram comparados: Regressão Linear Regularizada (RL) e *Gradient Boosting Machine* (GBM). A métrica de acurácia R^2 pode ser contra intuitivo do ponto de vista de um DBA. Mas, vale ressaltar que valores de R^2 próximos de 1 indicam predições perfeitas e valores menores que 1 (ou, até mesmo, negativos) representam predições de baixa acurácia.

Além da métrica R^2 com *grid-search*, também foi apresentado um modo gráfico de verificação de acurácia dos modelos gerados que exploram a relação do valor real com o valor predito. Para cada caso, treinamos um modelo com o conjunto de dados completo e geramos predições para os valores de saída reais desse mesmo conjunto de dados. Desse modo, em um gráfico de dispersão, adicionamos um ponto (x,y) para cada amostra do conjunto de dados, sendo x o valor predito pelo modelo e y o valor real da amostra. Assim, em situações onde os pontos estão próximos da reta $y = x$, as predições são mais acurada, e pontos longe da reta $y = x$ indicam predição menos acuradas.

Com isso, pode-se observar os casos onde é melhor utilizar características lineares ou características quadráticas para melhorar a acurácia dos modelos de desempenho gerados. Para tal, foram comparados ambos os casos com cada uma das seguintes métricas de desempenho:

- *TRS*: Tempo de resposta médio por segundo.
- *RPS*: Quantidade de requisições executadas pelo sistema por segundo (vazão).
- *VPS*: Número de violações por segundo.

A discussão sobre cada métrica citada acima é apresentada nas subseções seguintes.

5.2.1.1 Tempo de resposta médio por segundo (TRS)

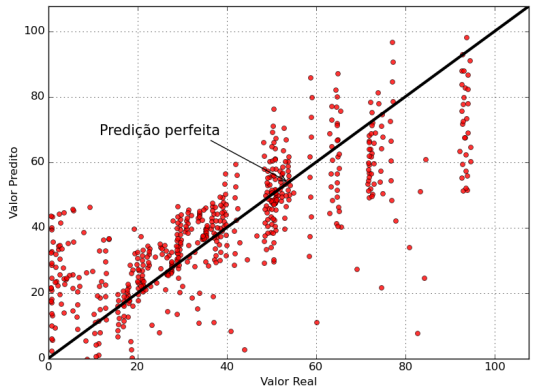
A métrica TRS corresponde a média dos tempos de resposta (latência) de todas as requisições executadas com sucesso pelo sistema. A Tabela 4 exibe pontuação R^2 obtida com a técnica *grid-search* e 10-*fold* usando os métodos RL e GBM. Também é apresentado os resultados quando as transformação de características lineares (CL) e de características quadráticas (CQ) são realizadas.

	TRS	
	RL	**GBM**
CL	0.49	0.85
CQ	0.74	0.83

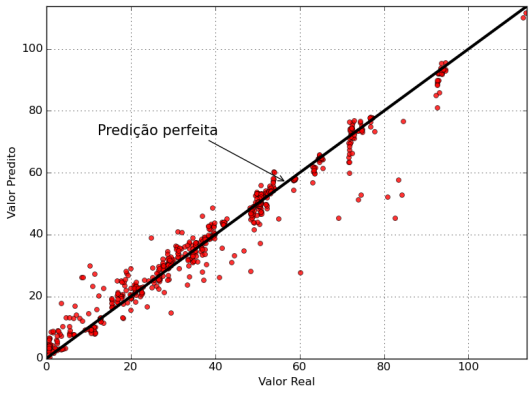
Tabela 4 – Métrica TRS: Pontuação R^2 obtido com *grid-search* e 10-*fold* para características lineares e quadráticas com métodos RL e GBM

O método RL apresentou resultados medianos quando aplicado com características lineares ($R^2 = 0.49$). Porém, ao utilizar características quadráticas, o RL conseguiu ajustar-se melhor aos dados ($R^2 = 0.83$), o que mostra que os dados apresentam um comportamento próximo do quadrático. Por sua vez, o método GBM foi capaz de se ajustar à curva dos dados de modo mais eficaz, por ser baseado em árvores. Como esperado, não houveram mudanças

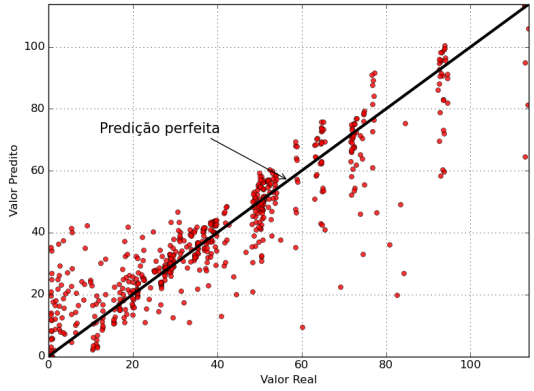
expressivas ao se utilizar características lineares ou quadráticas para o método GBM. A Figura 11 apresenta a relação entre os valores preditos e os valores reais para os modelos gerados.



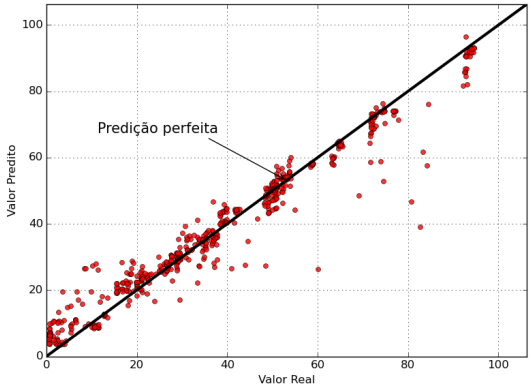
(a) Características lineares - Regressão Linear



(b) Características lineares - Gradient Boosting Machine



(c) Características quadráticas - Regressão Linear



(d) Características lineares - Gradient boosting machine

Figura 11 – Métrica TRS: relação entre valor predito e valor real para os modelos gerados

Como indica a pontuação R^2, os modelos gerados pelo método RL tem os pontos mais distantes da reta $y = x$ e, consequentemente, são modelos de acurácia mais baixa, o que pode ser um erro impraticável para algumas aplicações. No entanto, percebe-se que as amostras situadas no quadrante inferior-direito obtiveram predições ruins para todos os casos.

5.2.1.2 Vazão (RPS)

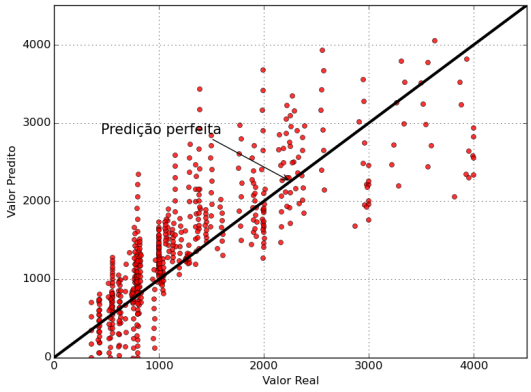
A métrica RPS considera a quantidade de requisições executadas a cada segundo. A abordagem para modelar o desempenho foi aplicada em termos da métrica RPS. A Tabela 5 exibe a pontuação R^2 obtida com a técnica *grid-search* e 10-*fold* usando os métodos RL e GBM assim como os resultados quando é feita a transformação de características lineares (CL) e características quadráticas (CQ) .

Similarmente à métrica TRS, o método RL teve pontuação R^2 mais baixa ao se

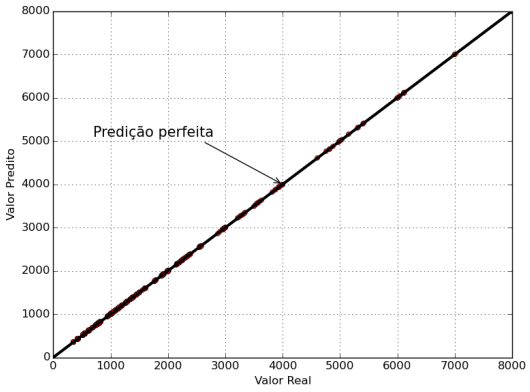
	RPS	
	RL	**GBM**
CL	0.61	0.98
CQ	0.84	0.98

Tabela 5 – Métrica RPS: Pontuação R^2 obtido com *grid-search* e 10-*fold* para características lineares e quadráticas com métodos RL e GBM

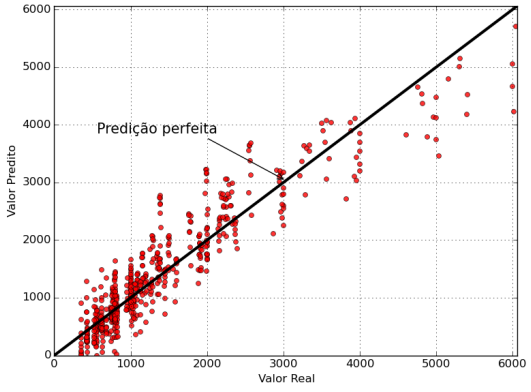
utilizar características lineares ($R^2 = 0.61$). Contudo, devido ao comportamento quadrático dos dados, as características quadráticas ajudaram o RL a se ajustar melhor aos dados ($R^2 = 0.84$). Já o método GBM, conseguiu predições próximas da perfeição ($R^2 = 0.98$) com características lineares ou quadráticas. A Figura 12 apresenta a relação entre os valores preditos e reais para dos modelos gerados.



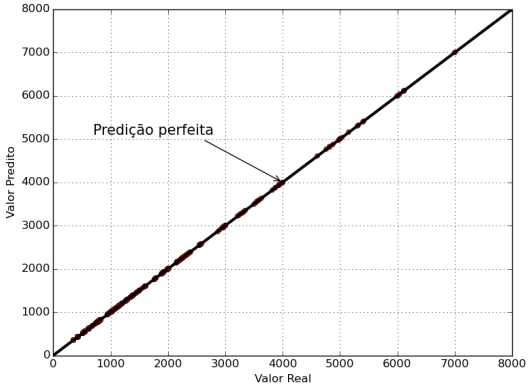
(a) Características lineares - Regressão Linear



(b) Características lineares - Gradient Boosting Machine



(c) Características quadráticas - Regressão Linear



(d) Características lineares - Gradient boosting machine

Figura 12 – Métrica RPS: relação entre valor predito e valor real para os modelos gerados

O gráfico 12a demonstra uma falta de acurácia pelo modelo gerado pelo método RL com características lineares pois os valor predito pode, em alguns casos, diferir do valor

predito em mais de 2000 unidades o que pode ser um erro demasiadamente alto para algumas aplicações. Mas ao se utilizar características quadráticas (Figura 12c), o métodos RL conseguiu uma acurácia melhor visto que em a grande maioria dos casos o valor real difere do predito em menos de 1000 unidades. Por sua vez, os gráficos 12b e 12d ilustram as predições da perfeição com todas as duas transformações de características.

5.2.1.3 Violações por segundo (VPS)

Similarmente às métricas anteriores, abordagem de modelagem de desempenho foi utilizada com o objetivo de modelar a métrica VPS. Definiu-se o limite de tempo para requisições em 100ms. A Tabela 6 apresenta um comparativo de pontuações R^2 usando RL e GBM com características lineares e quadráticas.

	VPS	
	RL	**GBM**
CL	0.93	0.99
CQ	0.97	0.99

Tabela 6 – Métrica VPS: Pontuação R^2 obtido com *grid-search* e 10-*fold* para características lineares e quadráticas com métodos RL e GBM

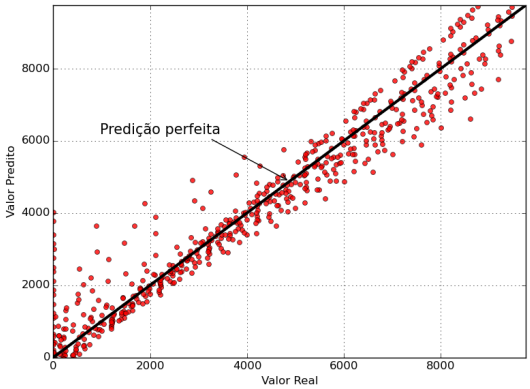
Como é possível constatar por meio da Tabela 6, a métrica VPS é mais previsível que as anteriores visto que, em todos modelos gerados, a pontuação R^2 foi alta. De qualquer maneira, percebe-se que houve uma melhoria na pontuação R^2 ao usar as característica quadráticas em relação a linear ($R^2 = 0.93$ para $R^2 = 0.97$). Dessa forma, o método GBM obteve acurácia próximo da perfeição em ambos casos ($R^2 = 0.99$)

A Figura 13 demonstra relação entre os valores preditos e os valores reais dos modelos gerados. Todos os gráficos demonstram predições acuradas pelos modelos. Mas vale destacar que as predições feitas pelo método GBM foram muito próximos do valor real.

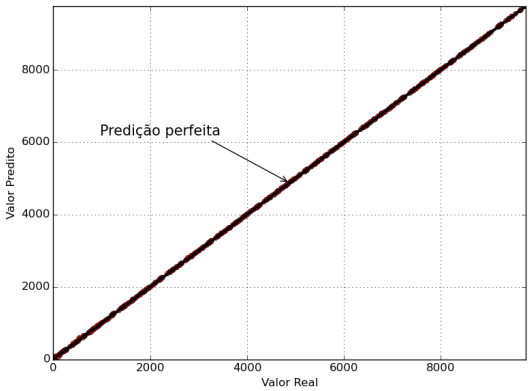
5.2.1.4 Verificação dos modelos de incerteza

O objetivo dos modelos de incerteza é predizer o desvio-padrão das métricas de desempenho considerando as características da carga de trabalho e do *cluster*. Para tanto, foi aplicado a abordagem de incerteza descrita na Seção 4.3.4 sobre as métricas TRS, RPS e VPS. A Tabela 7 apresenta a pontuação R^2 obtida por meio do *grid-search* e 10-*fold* para o método GBM.

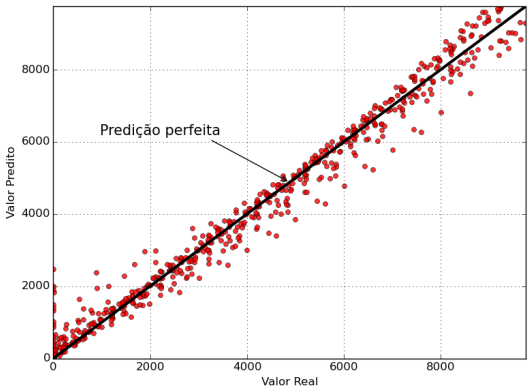
O método GBM conseguiu gerar modelo preditivos com ótima acurácia para as métricas TRS e RPS. Contudo, o método RL não conseguiu resultados aceitáveis em nenhumas das métricas o que mostra a não-linearidade das mesmas. A Figura 14 ilustra graficamente a relação entre os valores reais e os valores preditos pelos modelos gerados por GBM.



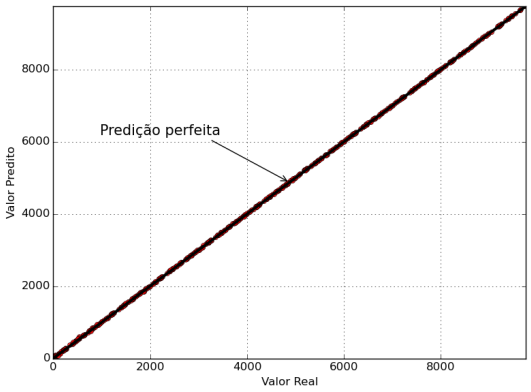
(a) Características lineares - Regressão Linear



(b) Características lineares - Gradient Boosting Machine



(c) Características quadráticas - Regressão Linear



(d) Características lineares - Gradient boosting machine

Figura 13 – Métrica VPS: relação entre valor predito e valor real para os modelos gerados

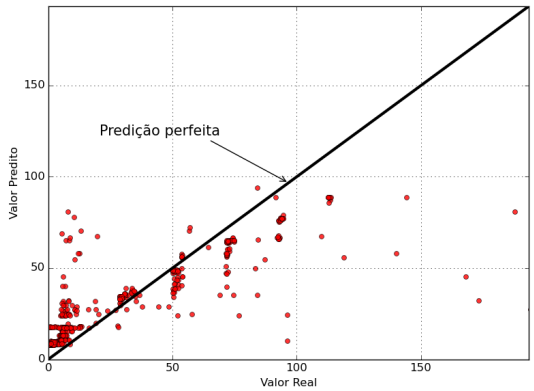
Método	**TRS**	**RPS**	**VPS**
RL	-0.15	-0.45	-14.86
GBM	0.31	0.98	0.98

Tabela 7 – Modelos de incerteza: Pontuação R^2 obtida através do *grid-search* para o método GBM usando características lineares.

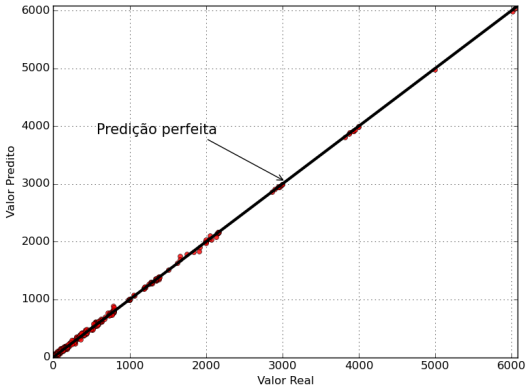
5.3 VERIFICAÇÃO DA ESTRATÉGIA ELASTICIDADE

Estratégias de elasticidade devem evitar violações de SLA e reduzir custos de infraestrutura utilizando mecanismos para a adição e remoção dinâmica de recursos. O objetivo experimental dessa seção é avaliar se a nossa abordagem de elasticidade consegue alocar o quantidade correta de recursos sob dois cenários:

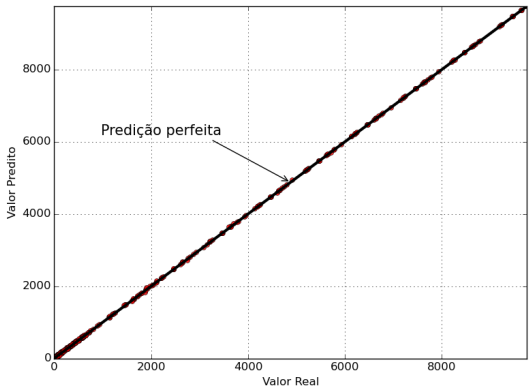
1. Quando a volume de requisições (requisições/segundo) da carga de trabalho aumenta
2. Quando a composição da carga de trabalho é alterada



(a) Métrica TRS.



(b) Métrica RPS.



(c) Métrica VPS.

Figura 14 – Modelos de incerteza: relação entre valores reais e valores preditos pelo modelo gerado pelo método GBM usando características lineares.

Adicionalmente, foi verificado a efetividade da nossa estratégia de provisionamento com gerenciamento da incerteza considerando alterações no volume de requisições. A efetividade da nossa abordagem de elasticidade é dependente da acurácia do modelo produzido pela fase de modelagem de desempenho. Assim, devemos também mostrar que o erro preditivo dos modelos é suficientemente pequeno para não induzir a nossa abordagem de elasticidade ao erro.

Os experimentos foram executados sobre o SGBD MongoDB em seu modo de replicação. Os procedimentos de adição e remoção de nós foram delegados ao próprio SGBD que já implementa essa funcionalidade por padrão. Para acelerar a adição de um nó, utilizou-se uma imagem de um máquina virtual com o SGBD instalado e com o banco já previamente carregado. Isso elimina o tempo de se carregar toda a base de dados em um novo nó. A variável SLA_{DB} que limita a proporção máxima de violações de SLA foi definida em 15%. Desse modo, o objetivo é manter a métrica VPS menor do que $SLA_{DB} \times target$.

5.3.1 Mudança no volume da carga de trabalho

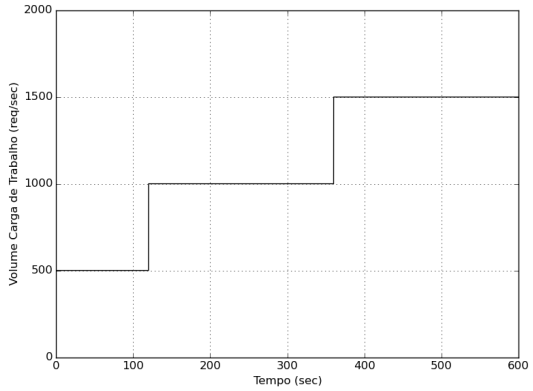
Nesse cenário, foi simulado a variação no volume de requisições da carga de trabalho. Para isso, foram executados dois experimentos para mostrar a adaptabilidade da abordagem

proposta em diferentes situações. O primeiro experimento visa testar a abordagem com cargas de trabalhos que aumentam o volume suavemente. Por sua vez, a segunda carga de trabalho submete o sistema a uma mudança mais brusca no volume de requisições. O volume é alterado através do parâmetro *target* do *benchmark* YCSB. Os parâmetros de mix $< p_{read}, p_{scan}, p_{update}, p_{insert} >$ foram fixados em $[0.5, 0.5, 0, 0]$ nos experimentos 1 e 2.

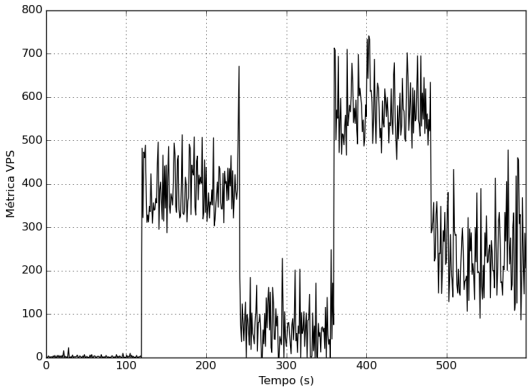
Note que foram avaliados apenas cenários com o aumento no volume da carga de trabalho, mas a estratégia se comporta do mesmo modo para aumento ou diminuição no volume da carga de trabalho. Isso ocorre porque a solução proposta se baseia apenas nas características da carga de trabalho atual e não do histórico.

5.3.1.1 Experimento 1: Aumento suave da carga de trabalho

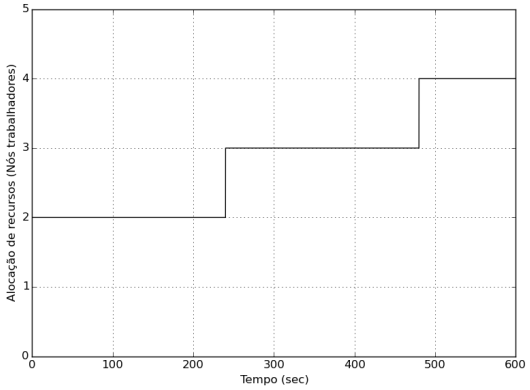
Nesse primeiro experimento, o SGBD é submetido a uma carga de trabalho onde o seu volume aumenta suavemente. A carga de trabalho é dada por uma sequência de valores para *target* e esta sequência é graficamente exposta em 15a. Essa carga dura um total de 600 segundos. Nos primeiros 120 segundos, o *target* é mantido em 500. Após isso, o *target* é aumentado para 1000 e é mantido por 240 segundos. E, finalmente, o *target* é definido em 1500 e é mantido por mais 240 segundos.



(a) Carga de trabalho submetida ao sistema



(b) Métrica VPS para experimento 1.



(c) Alocação de recursos.

Figura 15 – Carga de trabalho, métrica VPS e alocação para o experimento 1.

A métrica VPS da execução dessa carga de trabalho sobre o MongoDB é mostrada na Figura 15b e a Figura 15c mostra a alocação de recursos ao longo do experimento. O experimento começa com 2 máquinas sendo suficiente para manter a métrica VPS abaixo de $SLA_{DB} \times target = 15\% \times 500 = 75$. Como o aumento no volume no segundo 120 provocou um aumento em VPS que ficou em um estado de *undeprovisioning*, visto que o VPS flutua em torno de 400 e o ideal é $SLA_{DB} \times target = 15\% \times 1000 = 150$. Assim, no próximo ciclo de provisionamento (1 minuto), a estratégia de elasticidade decidiu que é necessário tomar uma ação de provisionamento para adicionar uma nova máquina.

A adição dessa máquina demora um pouco mais que 1 minuto, pois é necessário instanciar esse novo nó. Assim, por volta do segundo 250, a métrica VPS volta a um estado aceitável flutuando em torno de 80 o que é menor que 150. E, finalmente, no segundo 360, o volume de requisições por segundo sobe para 1500 e o VPS aumenta para cerca de 580 o que é maior do que o aceitável $SLA_{DB} \times target = 15\% \times 1500 = 225$. Igualmente, no próximo ciclo de provisionamento no segundo 420, a estratégia de elasticidade decide adicionar mais um nó. Assim, 1 minuto após, uma máquina é instanciada e VPS diminui para cerca de 240 o que é próximo do desejado (225). Isso ocorreu devido ao modelo preditivo ter subestimado o VPS para esse cenário. Mas, mesmo assim, esse erro foi suficientemente pequeno para que a QoS fosse muito próximo da desejada.

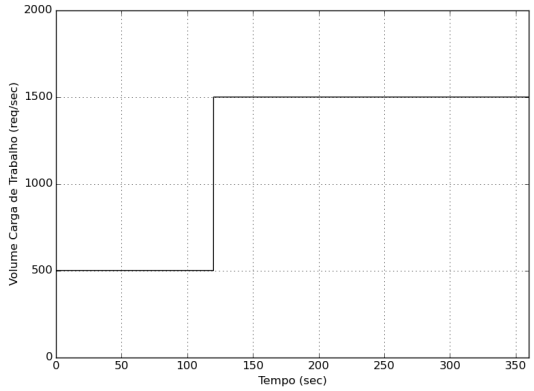
5.3.1.2 Experimento 2: Aumento brusco da carga de trabalho

Nesse segundo experimento, uma carga de trabalho, que muda de forma mais brusca, foi submetida ao sistema. Com isso, pode-se avaliar se estratégia consegue adicionar um número arbitrário de nós com fim de satisfazer o SLA e reduzir custos de infraestrutura. A carga de trabalho utilizada neste experimento é apresentada na Figura 16a dura 360 segundos e tem uma mudança mais brusca no volume da carga de trabalho de 500 requisições por segundo para 1500 no segundo 120.

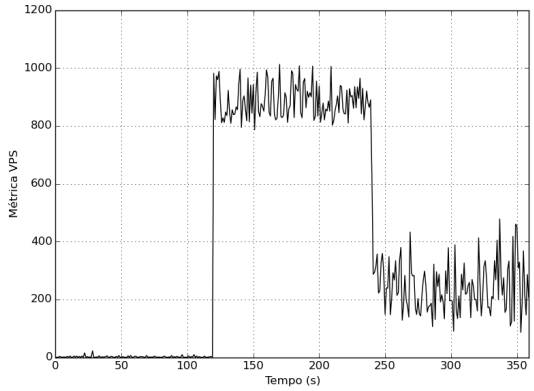
A métrica VPS e a alocação de recursos são exibidas nas Figuras 16b e 16c. O experimento foi iniciado com 2 nós. Esse alocação é suficiente para satisfazer a quantidade máxima de violações de SLA para essa caso: $SLA_{DB} \times target = 15\% \times 500 = 75$. No segundo 120 a volume de requisições salta para 1500 o que coloca o sistema em um estado de *undeprovisioning* pois o VPS flutua em torno de 900 que é bem maior que o objetivo do SLA: $SLA_{DB} \times target = 15\% \times 1500 = 225$. No próximo ciclo de provisionamento que ocorre no minuto 180, a estratégia calcula que é necessário se adicionar mais dois nós para retornar a um estado aceitável. Após a instanciação, as duas novas máquinas são adicionadas ao *cluster* por volta do segundo 240. Com isso, a métrica VPS retorna a um estado aceitável.

5.3.2 Experimento 3: Mudança na composição da carga de trabalho

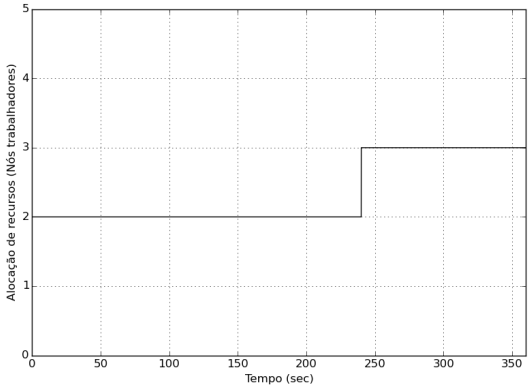
Este experimento tem como objetivo demostrar que a estratégia proposta é capaz de tratar carga de trabalho que mudam de composição ao longo do tempo. Para isso, o volume de requisições da carga de trabalho ($target$) foi fixado ao longo do tempo e a composição da carga



(a) Carga de trabalho submetida ao sistema.



(b) Métrica VPS.



(c) Alocação de recursos.

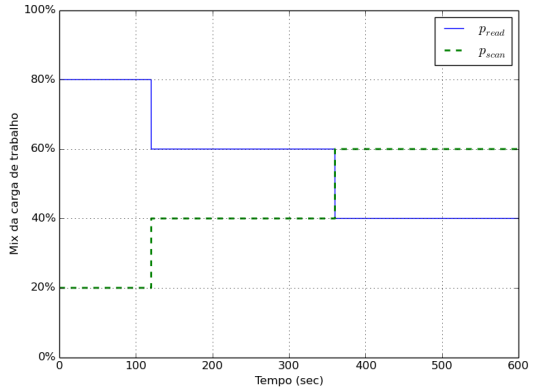
Figura 16 – Carga de trabalho, métrica VPS e alocação para o experimento 2.

(*mix*) foi alterada.

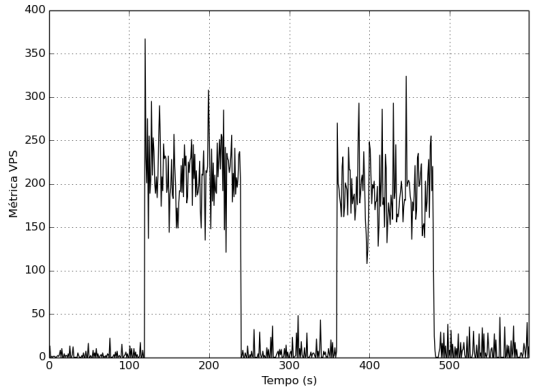
Para efeitos de simplicidade, este experimento variou dois parâmetros do *mix* da carga de trabalho: p_{read} e p_{scan} como mostra o gráfico da Figura 17a, onde o p_{read} começou com um valor alto 80% e vai diminuindo a media que o p_{scan} aumenta, tornando a carga de trabalho mais complexa de ser executada. O parâmetro $target$ foi mantido em 1000 durante todo o experimento. Além disso, o máximo de violação por segundo permitida é $SLA_{DB} \times target = 15\% \times 1000 = 150$.

A métrica VPS e a alocação de recursos durante o experimento 3 estão expostas nas Figuras 17b e 17c, respectivamente. Similarmente aos experimentos anteriores, o sistema inicia o experimento com 2 nós o que é suficiente para manter VPS abaixo de 150 até o segundo 120. No segundo 120 a carga se torna mais pesada pois a proporção de requisições *scan* é elevada para 40%. Requisições *scan* são mais complexas pois são consultas por intervalo e retornam um número aleatório entre 1 e 100 tuplas.

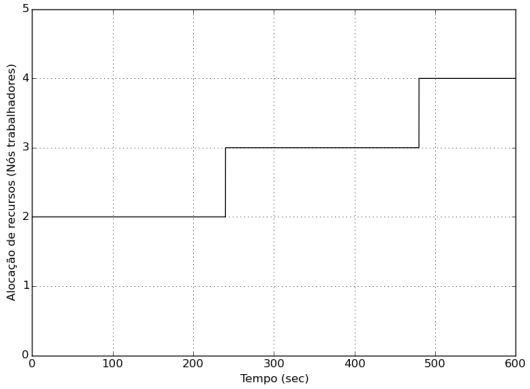
A métrica VPS flutua em torno de 220. Após um ciclo de provisionamento, a estratégia decide adicionar mais uma máquina para reduzir a quantidade de violações. Por volta do segundo 230, a máquina é adicionada ao *cluster* e o VPS cai para menos de 150. Novamente, a proporção de requisições *scan* sobe para 60% e o sistema entra em uma situação de *underprovisioning* onde o VPS atinge cerca de 200. Assim, no próximo ciclo de provisionamento,



(a) Variação dos parâmetros p_{read} e p_{scan}.



(b) Métrica VPS ao longo do experimento.



(c) Alocação de recursos.

Figura 17 – Carga de trabalho, métrica VPS e alocação para o experimento 3.

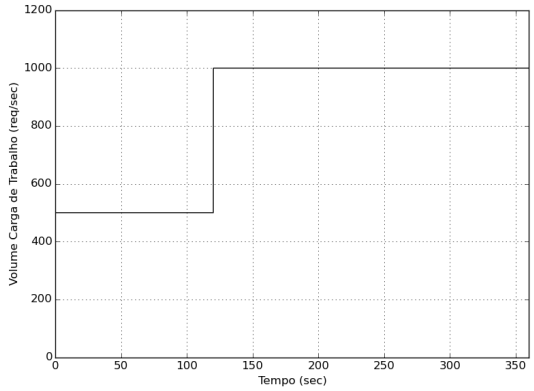
a estratégia adicionou uma máquina virtual que é incorporada ao *cluster* por volta do segundo 480. Desse modo, o VPS é controlado e diminui para um valor menor do que 150, satisfazendo as exigências de SLA.

5.3.3 Experimento 4: Verificação da estratégia elasticidade com gerenciamento da incerteza

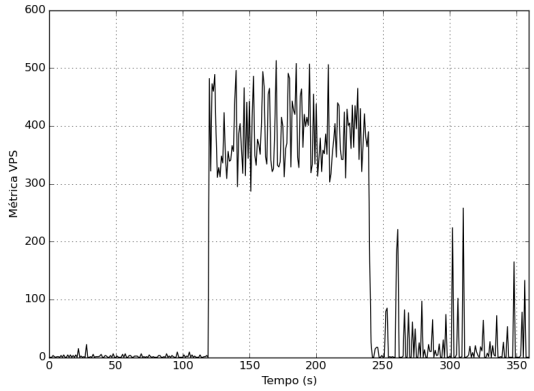
Nos experimentos anteriores pode-se perceber que a métrica de desempenho VPS apresenta ruído e pode gerar violações de SLA inesperadas. Para analisar esta situação, este experimento verifica a efetividade da estratégia de elasticidade com o gerenciamento da incerteza. Assim sendo, este experimento explora o caso em que é um alto nível de confiança é utilizado. Neste caso, o nível de confiança $\alpha = 90\%$ foi utilizado.

Este experimento utiliza as mesma configuração do experimento 1, onde variou-se o volume de requisições por segundo (*target*) e foi fixou-se o *mix* $[p_{read}, p_{scan}, p_{insert}, p_{update}]$ da carga de trabalho em $[50\%, 50\%, 0\%, 0\%]$. O valores *target* variaram segundo o gráfico da Figura 18a.

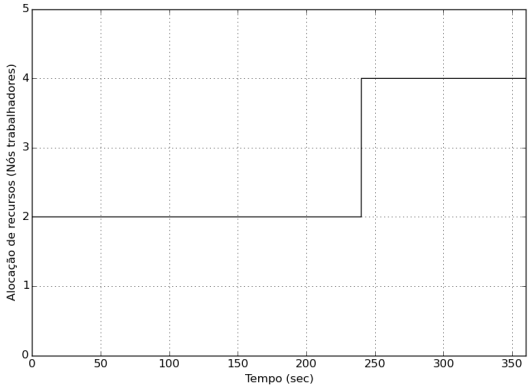
As Figuras 18b e 18c mostram a métrica VPS e a alocação de recursos da execução do experimento 4. Similarmente aos experimentos anteriores, o sistema foi inicializado com



(a) Variação dos parâmetros p_{read} e p_{scan}.



(b) Métrica VPS ao longo do experimento.



(c) Alocação de recursos.

Figura 18 – Carga de trabalho, métrica VPS e alocação para o experimento 4.

2 máquinas virtuais o que ocasiona em níveis aceitáveis da métrica VPS pois o nível máximo aceitável de violações nesse momento é $SLA_{DB} \times target = 15\% \times 500 = 75$.

O parâmetro $target$ foi elevado para 1000 no segundo 120 e o VPS sobe cerca de 400 o que caracteriza um estado de *underprovisioning* pois nesse momento o nível aceitável VPS é no máximo $SLA_{DB} \times target = 15\% \times 1000 = 150$. Assim, no ciclo de provisionamento seguinte, a estratégia disparou a adição de 2 novos nós e eles são adicionados ao *cluster* por volta do segundo 240. Assim, pode-se observar que essa ação faz com o VPS seja mantida abaixo de 150 em grande parte das medições, apenas 4 medições entre o segundo 240 e o segundo 360 ultrapassam 150 violações.

A estratégia com gerenciamento da incerteza apresenta resultados interessantes no segundo 240. Uma abordagem sem gerenciamento da incerteza adicionaria apenas 1 nó no lugar de 2 nós. Isso acarreta em um maior número de medições de VPS acima do aceitável (150). Para ilustrar essa situação, foi executado um pequeno experimento com as mesmas condições dos experimento 4 entre os segundo 360: $target = 1000$ e $mix = [50\%, 50\%, 0\%, 0\%]$. Pode-se observar pela Figura 19 que a adição de apenas 1 nó ocasionaria em um VPS em níveis maiores que 150 em várias medições.

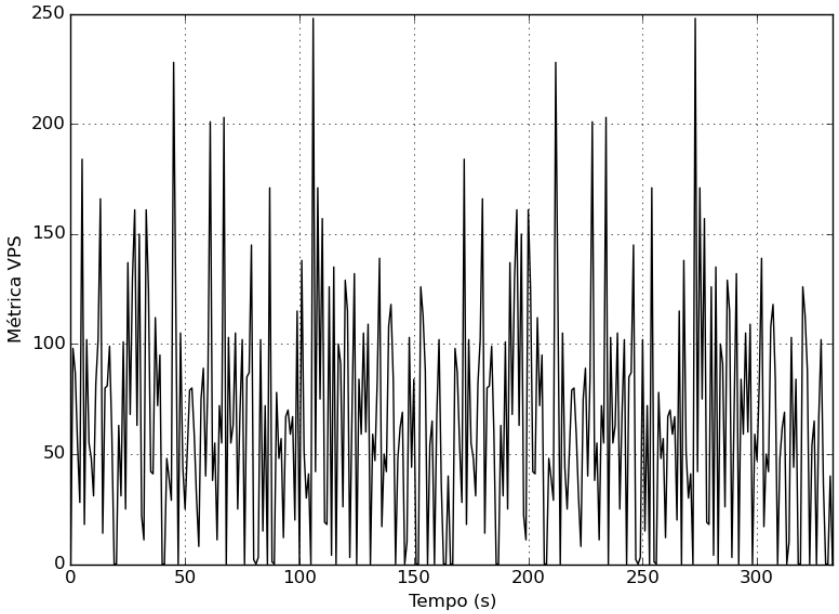


Figura 19 – Exemplo com a adição de apenas 1 nó no segundo 240.

5.4 CONCLUSÃO

Neste capítulo apresentamos a verificação da nossa abordagem de modelagem do desempenho onde medimos a capacidade de generalização dos modelos de desempenho e incerteza. Para os modelos de desempenho, analisamos em quais situações é mais vantajoso usar características lineares ou quadráticas. Para métodos de regressão linear, é mais vantajoso usar características quadráticas. Por sua vez, o método *Gradiente Boosting Machine* se manteve indiferente ao uso de características lineares ou quadráticas, por isso recomendamos usar características lineares por serem a abordagem mais simples. Para os modelos de incerteza, o modelo de regressão linear teve resultados inviáveis enquanto o *Gradient Boosting Machine* conseguiu construir modelos acurados para duas das três métricas estudadas.

Além da abordagem de modelagem de desempenho, também verificamos a nossa abordagem de elasticidade. Foram executados vários cenários de carga de trabalho para mostrar a adaptabilidade da nossa abordagem em relação a mudanças no volume e na composição da carga de trabalho para provisionar quantidade correta de recursos evitando *overprovisioning* e *underprovisioning*.

6 CONSIDERAÇÕES FINAIS

6.1 CONCLUSÃO

Esse trabalho apresentou uma nova abordagem para modelagem do desempenho para bancos de dados NoSQL. A abordagem proposta gera modelos que são capazes de predizer métricas de desempenho baseadas no SLA, capturando efeitos não-lineares causados por aspectos de distribuição e do modelo de concorrência. Essa estratégia gera modelos preditivos para estimar métricas de desempenho por meio de técnicas de aprendizado de máquina. O conjunto de treinamento para treinar os modelos de aprendizado foi gerado a partir de uma extensa experimentação do SGBD alvo, sobre o qual foram testados várias configurações de carga de trabalho e diferentes tamanhos de *cluster*. Dos experimentos foram extraídos apenas características que são genéricas a qualquer SGBD NoSQL: tamanho do *cluster* e parâmetros sobre a carga de trabalho (t,t_{read},t_{scan},t_{insert},t_{update}).

Foram investigados aspectos de como as configurações do *cluster* e da carga de trabalho influenciam de modo não-linear no desempenho. Técnicas de aprendizado baseadas em árvore, como o *Gradien Boosting Machine*, conseguem capturar naturalmente dependências não-lineares entre as características e as métricas. Mas, em geral, essa abordagem tem baixo poder de extrapolação por ser baseada em árvores. Assim, foi proposto uma transformação de características (lineares e quadráticas) para permitir que a classe de métodos de regressão linear capture o comportamento das métricas de desempenho. As métricas de desempenho apresentaram ruídos ao longo dos experimentos por fatores como interferência de rede ou interferência advinda do compartilhamento de recursos, e.g. outros inquilinos. Por esse motivo, além de modelar o comportamento médio do SGBD, também foi modelado o ruído que ocorre nas métricas de desempenho. Resultados experimentais confirmam que as abordagens *boosting* e modelos lineares conseguem gerar modelos acurados, sendo que os modelos lineares apresentam menor acurácia e maior poder extrapolação do que as abordagens *boosting*. Para os modelos de incerteza que delineam o ruído, os métodos *boosting* apresentaram superioridade sobre os outros métodos.

Também foi apresentada uma estratégia de provisionamento elástico para bancos de dados distribuídos baseada em modelos de desempenho para computar a alocação ótima de recursos para a carga de trabalho corrente com o objetivo de, simultaneamente, reduzir custos de infraestrutura e atender ao SLA. Para isso, foi proposto um problema de otimização matemática cuja solução expressa a melhor alocação de recursos para a carga de trabalho corrente. Este problema de otimização usa o modelo de desempenho para estimar qual é o QoS oferecido em outros cenários de alocação. Ademais, esse problema de otimização foi estendido, utilizando modelos de incerteza, para melhorar a confiabilidade no provisionamento. Com isso, é possível definir um parâmetro $\alpha \in]0,1[$ que controla o nível de confiança desejado no provisionamento. Os resultados experimentais apontam que estratégia de elasticidade proposta utiliza os recursos eficientemente enquanto reduz custos de infraestrutura.

6.2 TRABALHOS FUTUROS

Existem algumas oportunidades de trabalhos futuros que derivam do presente trabalho, das quais se destacam: (i) modelagem da desempenho *online* de bancos de dados distribuídos; (ii) considerar recursos heterogêneos ou nuvens híbridas; (iii) investigar a influência do período de provisionamento para perfis específicos de aplicação; (iv) testar cargas de trabalho reais; e (v) incorporar modelos de *forecast* e suas implicações para provisionamentos proativos.

Este trabalho também provê meios diretos de responder várias questões sobre desempenho de bancos de dados NoSQL tais como "Quais classes de requisições degradam mais o desempenho?Quais pares de classes de requisições são mais conflituosas por contenção de dados?Como identificar se o modelo de dados, particionamento de dados e configuração do SGBD são adequados para uma carga de trabalho em particular?". Investigações sobre algumas dessas questões já estão em curso.

REFERÊNCIAS

ALPAYDIN, E. **Introduction to machine learning**. [S.l.]: MIT press, 2014.

AZURE. ***Microsoft Azure***. 2012. <http://www.microsoft.com/azure/>.

BINNIG, C.; KOSSMANN, D.; KRASKA, T.; LOESING, S. How is the weather tomorrow?: towards a benchmark for the cloud. In: **DBTest '09: Proceedings of the Second International Workshop on Testing Database Systems**. New York, NY, USA: ACM, 2009. p. 1–6. ISBN 978-1-60558-706-6.

BISHOP, C. M. **Pattern recognition and machine learning**. [S.l.]: springer, 2006.

BRANTNER, M.; FLORESCU, D.; GRAF, D.; KOSSMANN, D.; KRASKA, T. Building a database on s3. In: **Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08**. New York: ACM Press, 2008. p. 251. ISBN 9781605581026.

BROWNE, J. Brewer's cap theorem. **J. Browne blog**, 2009.

BUYYA, R.; YEO, C. S.; VENUGOPAL, S.; BROBERG, J.; BRANDIC, I. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. **Future Gener. Comput. Syst.**, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 25, n. 6, p. 599–616, 2009. ISSN 0167-739X.

CHANG, F.; DEAN, J.; GHEMAWAT, S.; HSIEH, W. C.; WALLACH, D. A.; BURROWS, M.; CHANDRA, T.; FIKES, A.; GRUBER, R. E. Bigtable: A distributed storage system for structured data. **ACM Transactions on Computer Systems (TOCS)**, ACM, v. 26, n. 2, p. 4, 2008.

CHI, Y.; MOON, H. J.; HACIGüMüs, H.; TATEMURA, J. Sla-tree: A framework for efficiently supporting sla-based decisions in cloud computing. In: **Proceedings of the 14th International Conference on Extending Database Technology**. New York, NY, USA: ACM, 2011. (EDBT/ICDT '11), p. 129–140. ISBN 978-1-4503-0528-0. Disponível em: <http://doi.acm.org/10.1145/1951365.1951383>.

CIURANA, E. **Developing with Google App Engine**. Berkely, CA, USA: Apress, 2009. ISBN 1430218312, 9781430218319.

COOPER, B. F.; SILBERSTEIN, A.; TAM, E.; RAMAKRISHNAN, R.; SEARS, R. Benchmarking cloud serving systems with ycsb. In: ACM. **Proceedings of the 1st ACM symposium on Cloud computing**. [S.l.], 2010. p. 143–154.

COUTINHO, E. F.; SOUSA, F. R. C.; REGO, P. A. L.; GOMES, D. G.; SOUZA, J. N. de. Elasticity in cloud computing: a survey. **annals of telecommunications - annales des télécommunications**, Springer Paris, p. 1–21, 2014. ISSN 0003-4347.

CRUZ, F.; MAIA, F.; MATOS, M.; OLIVEIRA, R.; PAULO, J. a.; PEREIRA, J.; VILAcA, R. Met: Workload aware elasticity for nosql. In: **Proceedings of the 8th ACM European Conference on Computer Systems**. [S.l.]: ACM, 2013. (EuroSys '13), p. 183–196.

DECANDIA, G.; HASTORUN, D.; JAMPANI, M.; KAKULAPATI, G.; LAKSHMAN, A.; PILCHIN, A.; SIVASUBRAMANIAN, S.; VOSSHALL, P.; VOGELS, W. Dynamo: amazon's highly available key-value store. In: ACM. **ACM SIGOPS Operating Systems Review**. [S.l.], 2007. v. 41, n. 6, p. 205–220.

DIDONA, D.; ROMANO, P. Performance modelling of partially replicated in-memory transactional stores. In: IEEE. **Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2014 IEEE 22nd International Symposium on**. [S.l.], 2014. p. 265–274.

DUGGAN, J.; CETINTEMEL, U.; PAPAEMMANOUIL, O.; UPFAL, E. Performance prediction for concurrent database workloads. In: **Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data**. [S.l.]: ACM, 2011. (SIGMOD '11), p. 337–348. ISBN 978-1-4503-0661-4.

ELMORE, A. J.; DAS, S.; AGRAWAL, D.; ABBADI, A. E. Zephyr: live migration in shared nothing databases for elastic cloud platforms. In: **SIGMOD '11**. [S.l.: s.n.], 2011. p. 301–312. ISBN 978-1-4503-0661-4.

FITO, J. O.; PRESA, I. G.; GUITART, J. Sla-driven elastic cloud hosting provider. **PDP, Euromicro'10**, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 111–118, 2010. ISSN 1066-6192.

FRIEDMAN, J. H.; MEULMAN, J. J. Multiple additive regression trees with application in epidemiology. **Statistics in medicine**, Wiley Online Library, v. 22, n. 9, p. 1365–1381, 2003.

GANAPATHI, A.; KUNO, H.; DAYAL, U.; WIENER, J. L.; FOX, A.; JORDAN, M.; PATTERSON, D. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In: IEEE. **Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on**. [S.l.], 2009. p. 592–603.

GRAY, J.; HELLAND, P.; O'NEIL, P.; SHASHA, D. The dangers of replication and a solution. In: ACM. **ACM SIGMOD Record**. [S.l.], 1996. v. 25, n. 2, p. 173–182.

HBASE, A. ***HBase***. 2013. <http://hbase.apache.org/>.

INC, M. ***MongoDB***. 2013. <http://www.mongodb.com>.

KASSELA, E.; BOUMPOUKA, C.; KONSTANTINOU, I.; KOZIRIS, N. Automated workload-aware elasticity of nosql clusters in the cloud. In: IEEE. **Big Data (Big Data), 2014 IEEE International Conference on**. [S.l.], 2014. p. 195–200.

KNILL, O. Probability and stochastic processes with applications. **Havard Web-Based**, 1994.

KONSTANTINOU, I.; ANGELOU, E.; BOUMPOUKA, C.; TSOUMAKOS, D.; KOZIRIS, N. On the elasticity of nosql databases over cloud management platforms. In: ACM. **Proceedings of the 20th ACM international conference on Information and knowledge management**. [S.l.], 2011. p. 2385–2388.

KONSTANTINOU, I.; ANGELOU, E.; TSOUMAKOS, D.; BOUMPOUKA, C.; KOZIRIS, N.; SIOUTAS, S. Tiramola: elastic nosql provisioning through a cloud management platform. In: ACM. **Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data**. [S.l.], 2012. p. 725–728.

LABS, R. ***Redis***. 2015. <http://redis.io>.

LIU, S.; LIANG, Y.; BROOKS, M. Eucalyptus: a web service-enabled e-infrastructure. In: **CASCON '07: Proceedings of the 2007 conference of the center for advanced studies on Collaborative research**. New York, NY, USA: ACM, 2007. p. 1–11.

LOH, W.-Y.; CHEN, C.-W.; ZHENG, W. Extrapolation errors in linear model trees. **ACM Transactions on Knowledge Discovery from Data (TKDD)**, ACM, v. 1, n. 2, p. 6, 2007.

MALKOWSKI, S.; HEDWIG, M.; JAYASINGHE, D.; PU, C.; NEUMANN, D. Cloudxplor: a tool for configuration planning in clouds based on empirical data. In: **SAC '10**. New York, NY, USA: ACM, 2010. p. 391–398. ISBN 978-1-60558-639-7. Disponível em: <http://doi.acm.org/10.1145/1774088.1774172>.

MASSIE, M. L.; CHUN, B. N.; CULLER, D. E. The ganglia distributed monitoring system: design, implementation, and experience. **Parallel Computing**, Elsevier, v. 30, n. 7, p. 817–840, 2004.

MELL, P.; GRANCE, T. ***NIST Working Definition of Cloud Computing (Draft)***. 2011. National Institute of Standards and Technology. <http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_cloud-definition.pdf>.

MOZAFARI, B.; CURINO, C.; JINDAL, A.; MADDEN, S. Performance and resource modeling in highly-concurrent oltp workloads. In: ACM. **Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data**. [S.l.], 2013. p. 301–312.

OLSHEN, L.; STONE, C. J. et al. Classification and regression trees. **Wadsworth International Group**, v. 93, n. 99, p. 101, 1984.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

PRITCHETT, D. Base: An acid alternative. **Queue**, ACM, v. 6, n. 3, p. 48–55, 2008.

PROJECT, T. apache cassandra. ***Cassandra***. 2013. <http://cassandra.apache.org/>.

ROBINSON, D. **Amazon Web Services Made Simple: Learn how Amazon EC2, S3, SimpleDB and SQS Web Services enables you to reach business goals faster**. London, UK, UK: Emereo Pty Ltd, 2008. ISBN 1921573066, 9781921573064.

SANTOS, G. A. C.; MAIA, J. G. R.; MOREIRA, L. O.; SOUSA, F. R. C.; MACHADO, J. C. Scale-space filtering for workload analysis and forecast. In: **CLOUD '13**. [S.l.: s.n.], 2013. p. 677–684.

SCHAD, J.; DITTRICH, J.; QUIANÉ-RUIZ, J.-A. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. **PVLDB**, v. 3, n. 1, p. 460–471, 2010.

SCHNJAKIN, M.; ALNEMR, R.; MEINEL, C. Contract-based cloud architecture. In: **CloudDB '10**. New York, NY, USA: ACM, 2010. p. 33–40. ISBN 978-1-4503-0380-4. Disponível em: <http://doi.acm.org/10.1145/1871929.1871936>.

SOUSA, F. R. C.; MACHADO, J. C. Towards elastic multi-tenant database replication with quality of service. In: **UCC '12**. [S.l.: s.n.], 2012. p. 168–175.

SOUSA, F. R. C.; MACHADO, J. C. Gerenciamento de dados em nuvem. In: **XXXIII Jornadas de Atualização em Informática (JAI 2014)**. [S.l.: s.n.], 2014. p. 1–40.

SOUSA, F. R. C.; MOREIRA, L. O.; MACHADO, J. C. Computação em nuvem: Conceitos, tecnologias, aplicações e desafios. In: ____. Piauí: EDUFPI, 2009. (In: MOURA, R. S. (Org.) ; SOUZA, F. V. (Org.) ; OLIVEIRA, A. C. (Org.). Escola Regional de Computação (Ceará, Maranhão e Piauí, ERCEMAPI 2009, 1. ed.).

SOUSA, F. R. C.; MOREIRA, L. O.; SANTOS, G. A. C.; MACHADO, J. C. Quality of service for database in the cloud. In: **Proceedings of the 2nd International Conference on Cloud Computing and Services Science - CLOSER '12**. [S.l.: s.n.], 2012. p. 595–601.

TSOUMAKOS, D.; KONSTANTINOU, I.; BOUMPOUKA, C.; SIOUTAS, S.; KOZIRIS, N. Automated, elastic resource provisioning for nosql clusters using tiramola. In: **13th IEEE/ACM International Symposium on Cloud and Grid Computing Cluster (CCGrid)**. [S.l.: s.n.], 2013. p. 34–41.

VAQUERO, L. M.; RODERO-MERINO, L.; CACERES, J.; LINDNER, M. A break in the clouds: towards a cloud definition. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 39, n. 1, p. 50–55, 2009. ISSN 0146-4833.

XIONG, P.; CHI, Y.; ZHU, S.; MOON, H. J.; PU, C.; HACIGÜMÜŞ, H. Intelligent management of virtualized resources for database systems in cloud environment. In: IEEE. **Data Engineering (ICDE), 2011 IEEE 27th International Conference on**. [S.l.], 2011. p. 87–98.

XIONG, P.; CHI, Y.; ZHU, S.; MOON, H. J.; PU, C.; HACIGÜMÜS, H. Intelligent management of virtualized resources for database systems in cloud environment. In: **ICDE'11**. [S.l.: s.n.], 2011. p. 87–98.

YANG, F.; SHANMUGASUNDARAM, J.; YERNENI, R. A scalable data platform for a large number of small applications. In: **CIDR 2009**. [s.n.], 2009. p. 1–10. Disponível em: <http://www-db.cs.wisc.edu/cidr/cidr2009/Paper_17.pdf>.

ZHANG, G.; PATUWO, B. E.; HU, M. Y. Forecasting with artificial neural networks:: The state of the art. **International journal of forecasting**, Elsevier, v. 14, n. 1, p. 35–62, 1998.