Proposta de um Sistema Inteligente de Monitoramento Baseado em Seleção de Características

Gustavo Silveira Marques

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2020

Gustavo Silveira Marques

Proposta de um Sistema Inteligente de Monitoramento Baseado em Seleção de Características

Dissertação de mestrado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Rafael Pasquini
Coorientador: Raquel Fialho Queiroz de Lafetá

Uberlândia
2020

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

M357p 2020

Marques, Gustavo Silveira, 1995-
Proposta de um sistema inteligente de monitoramento baseado em seleção de características [recurso eletrônico] / Gustavo Silveira Marques. - 2020.

Orientador: Rafael Pasquini.
Coorientadora: Raquel Fialho Queiroz de Lafetá.
Dissertação (mestrado) - Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação.
Modo de acesso: Internet.
Disponível em: http://doi.org/10.14393/ufu.di.2021.5595
Inclui bibliografia.
Inclui ilustrações.

1. Computação. I. Pasquini, Rafael, 1981-, (Orient.). II. Lafetá , Raquel Fialho Queiroz de, 1983-, (Orient.). III. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. IV. Título.

CDU: 681.3

Glória Aparecida
Bibliotecária - CRB-6/2047

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Coordenação do Programa de Pós-Graduação em Ciência da Computação
Av. João Naves de Ávila, nº 2121, Bloco 1A, Sala 243 - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4470 - www.ppgco.facom.ufu.br - cpgfacom@ufu.br

ATA DE DEFESA - PÓS-GRADUAÇÃO

Programa de Pós-Graduação em:	Ciência da Computação				
Defesa de:	Mestrado Acadêmico, 37/2020, PPGCO				
Data:	16 de dezembro de 2020	Hora de início:	09:00	Hora de encerramento:	12:00
Matrícula do Discente:	11812CCP017				
Nome do Discente	Gustavo Silveira Marques				
Título do Trabalho:	Proposta de um Sistema Inteligente de Monitoramento Baseado em Seleção de Características				
Área de concentração:	Ciência da Computação				
Linha de pesquisa:	Sistemas de Computação				
Projeto de Pesquisa de vinculação:	-				

Reuniu-se, por videoconferência, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Ciência da Computação, assim composta: Professores Doutores: Paulo Rodolfo da Silva Leite Coelho - FACOM/UFU; Fábio Luciano Verdi - UFSCar; Raquel Fialho de Queiroz Lafetá - UNIESSA (coorientadora) e Rafael Pasquini - FACOM/UFU, orientador do candidato.

Os examinadores participaram desde as seguintes localidades: Fábio Luciano Verdi - Sorocaba/SP; Paulo Rodolfo da Silva Leite Coelho, Raquel Fialho de Queiroz Lafetá e Rafael Pasquini - Uberlândia/MG. O discente participou da cidade de Uberlândia/MG.

Iniciando os trabalhos o presidente da mesa, Prof. Dr. Rafael Pasquini, apresentou a Comissão Examinadora e o candidato, agradeceu a presença do público, e concedeu ao Discente a palavra para a exposição do seu trabalho. A duração da apresentação do Discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir o candidato. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o candidato:

Aprovado.

Esta defesa faz parte dos requisitos necessários à obtenção do título de Mestre.

O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Rafael Pasquini**, **Professor(a) do Magistério Superior**, em 16/12/2020, às 15:31, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Paulo Rodolfo da Silva Leite Coelho**, **Professor(a) do Magistério Superior**, em 16/12/2020, às 17:16, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Fábio Luciano Verdi**, **Usuário Externo**, em 16/12/2020, às 17:54,

conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Raquel Fialho de Queiroz Lafetá**, **Usuário Externo**, em 04/03/2021, às 22:10, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **2458897** e o código CRC **950F360C**.

A Deus, à família e aos amigos, sem os quais nenhum trabalho pode ser realizado.

Agradecimentos

Agradeço, primeiramente, a Deus e a minha família, cujo apoio foi fundamental não só para que eu finalizasse este trabalho, como também para que eu lhe desse início.

Agradeço ao meu orientador, Rafael Pasquini, e a minha coorientadora, Raquel Fialho, pela paciência e pela dedicação que empenharam para que o presente trabalho pudesse ser concluído.

Esta trabalho é financiado com recursos da 4º chamada colaborativa BR-EU no contexto do H2020, registrados no acordo 777067 (*NECOS - Novel Enablers for Cloud Slicing*), que é fomentado pelo Ministério da Ciência e Tecnologia no lado Brasileiro e pela Comissão Europeia de Tecnologia no lado Europeu.

"Educa e transformarás a irracionalidade em inteligência, a inteligência em humanidade e a humanidade em angelitude. Educa e edificarás o paraíso na Terra."
(Emmanuel)

Resumo

Esse trabalho se baseia no paradigma *Slice-as-a-Service* proposto no projeto *Novel Enablers for Cloud Slicing* (NECOS). Assumindo fatias (*slices*) fim-a-fim, compostas por recursos de múltiplos provedores de infraestrutura, esse trabalho propõe um sistema inteligente de monitoramento capaz de dinamicamente selecionar métricas que melhor atendam às necessidades de gerenciamento das *slices*, mantendo a sua precisão. Basicamente, deseja-se evitar o tráfego de dados desnecessários extraídos dos diferentes provedores de infraestrutura, entregando um conjunto essencial de dados para as funções de gerenciamento. Apresentados os experimentos realizados que comparam algoritmos de correlação e de aprendizado de máquina, é possível perceber que a redução dos dados, através da seleção de características, aumenta de maneira satisfatória a precisão na estimativa de métricas de qualidade de serviço.

Palavras-chave: NECOS. Monitoria. Seleção de características. Inteligência artificial. *Slice. Slice-as-a-Service.*

Abstract

This work builds upon the *Slice-as-a-Service* paradigm proposed in the *Novel Enablers for Cloud Slicing* (NECOS) project. Assuming the provision of end-to-end slices which are composed by resources coming from multiple infrastructure providers, this work proposes an intelligent monitoring system, aiming to dynamically select a set of features that best fits the real life time management needs of the slices, keeping the accuracy of that management. We want to avoid the movement of unnecessary information from the infrastructure providers, delivering essential data to management functions. Presenting the experiments made in our testbed, which compared both algorithms of correlation and machine learning, it's possible to notice that the data reduction, through the feature selection, increases satisfactorily the quality of service estimative precision.

Keywords: NECOS. Monitoring. Feature selection. Artificial intelligence. Slice. Slice-as-a-Service..

Lista de ilustrações

Lista de tabelas

Lista de siglas

API *Application Programming Interface*

CPU *Central Processing Unit*

CSV *Comma-Separated-Values*

DHT *Distributed hash table*

IMA *Infrastructure & Monitoring Abstraction*

KPI *Key Performance Indicator*

NECOS *Novel Enablers for Cloud Slicing*

QoS *Quality of Service*

RAM *Random Access Memory*

RVMnt *Resource & VM Monitoring*

RVMgt *Resource & VM Management*

SLA *Service Level Agreement*

SRO *Service and Resource Orchestrator*

SVM *Support Vector Machine*

TCP *Transmission Control Protocol*

WIM *Wide-area network Infrastructure Manager*

vCPU *virtual Central Processing Unit*

VIM *Virtual Infrastructure Manager*

VM *Virtual Machine*

Lista de algoritmos

Sumário

Capítulo 1

Introdução

A evolução da tecnologia da computação sempre envolveu diferentes áreas do conhecimento, buscando alavancar novas formas de suportar/facilitar as mais variadas funções do nosso cotidiano. Nos últimos anos, observamos a consolidação de tecnologias que tornaram a computação, de fato, ubíqua e, desta forma, certamente mais inclusiva. Isto se nota, por exemplo, pois ao falarmos, hoje, de computação em nuvem, não mais é necessário explicar do que se trata, nem mesmo a sua importância. Permitir o uso de recursos talhados à sua real demanda, suportar processamentos pesados, maior capacidade de armazenamento de informação, segurança dos dados e acesso a partir de qualquer localidade são vantagens claras e conhecidas desta tecnologia.

Empresas como Google, Microsoft, Amazon, IBM e outras, disponibilizam seus servidores de computação em nuvem oferecendo aos seus usuários uma diversidade de serviços, criando conceitos como IaaS (*Infrastructure as a Service*), PaaS (*Platform as a Service*) e SaaS (*Software as a Service*) (KäCHELE et al., 2013), o que demonstra as inúmeras possibilidades dentro da arquitetura de computação em nuvem, que serão tanto mais aproveitadas quanto maior for a necessidade dos usuários. Pode-se citar alguns exemplos desses serviços como sendo: virtualização de infraestruturas, plataformas de desenvolvimento colaborativo, oferta dos mais variados serviços, tais como, hospedagem de sites, serviços de e-mail e serviços de armazenamento de dados.

Naturalmente, acompanhando toda esta evolução, novos requisitos são apresentados e desafiam a tecnologia. No contexto deste trabalho, um questionamento central para a fundamentação da pesquisa é: um determinado provedor de nuvem é sempre capaz de suprir todas as demandas de seus clientes?

Como desdobramentos deste questionamento, uma vez que certamente um único provedor não pode atender integralmente as diferentes demandas que podem vir de seus clientes, um novo questionamento seria: é possível estabelecer comunicação entre diversos provedores, de maneira que um provedor, ao não ser capaz de atender totalmente o pedido do seu cliente, possa contratar serviços de terceiros, buscando atender integralmente e de maneira transparente, ao seu cliente?

Para responder essas perguntas pode-se pensar em uma plataforma, que ofereça aos provedores de computação em nuvem a possibilidade de interagir uns com os outros, de forma que a demanda dos usuários que os procurassem pudesse ser totalmente atendida de forma transparente. Uma plataforma que ofertasse aos seus clientes níveis de gerenciamento dos serviços contratados, alertando-os caso se mostrassem inadequados e que, enfim, disponibilizasse uma interface clara e simples entre os diversos provedores de computação em nuvem. É nesse contexto, buscando suprir estas necessidades, que a plataforma *Novel Enablers for Cloud Slicing* (NECOS)[1] ganha destaque. Buscando ofertar aos seus usuários as facilidades mencionadas, construindo uma interface confiável de comunicação entre os provedores de computação na nuvem, otimizando o processo de atendimento aos clientes e possibilitando variados níveis de gerenciamento da infra-estrutura contratada, uma proposição de arquitetura para este sistema foi elaborada através da colaboração de onze instituições brasileiras e europeias.

O NECOS pode ser entendido por meio de quatro características principais:

1. O NECOS traz um novo tipo de serviço denominado *Slice-as-a-Service*. Uma *slice* pode ser entendida como um grupo de recursos físicos ou virtuais (rede, processamento e armazenamento) que pode agir como um *sub-cloud*, isto é, uma parte isolada de um servidor de computação na nuvem, capaz de suportar serviços de maneira independente de outras *slices*. Gerenciar uma *slice* significa controlar os componentes dos serviços que rodam nela, bem como as funções virtualizadas de rede e funções de sistema atribuídas a ela;

2. Permitir a configuração das *slices*, a fim de acomodar melhor as diversas demandas de serviço. Adaptações e reconfigurações, incluindo questões de interoperabilidade e portabilidade de dados, são feitas em um nível de granularidade por *slice*, ao invés de uma nuvem inteira. Essa configuração é obtida por meio do uso de software especialmente projetado que pode descrever e gerenciar os vários aspectos que compõem as *slices* no ambiente de nuvem;

3. Permitir que cada aspecto que compreende o ambiente de nuvem - desde a infraestrutura de rede entre máquinas virtuais, ao *Service Level Agreement* (SLA) dos aplicativos hospedados nas *slices* - seja gerenciado por meio de software. Isso reduz a complexidade relacionada à configuração e operação da infraestrutura, o que, por sua vez, facilita o gerenciamento da *slice*. Essa infraestrutura tende a ser de grande escala, geralmente composta de milhares de servidores e elementos de rede, suportando dezenas de milhares de máquinas virtuais, redes virtuais e aplicativos;

4. Utilizar sistemas de gerenciamento e virtualização leves e uniformes, com componentes pequenos, implantáveis em um grande número de pequenos servidores e sistemas

[1] http://www.h2020-necos.eu/project-overview/

em nuvem, tanto no núcleo quanto nas bordas da rede. Esses elementos leves permitem a integração de *data centers* no núcleo e, também, na borda móvel, garantindo a portabilidade de dados e a compatibilidade entre nuvens.

Uma das funcionalidades ofertadas pelo NECOS é o gerenciamento das *slices*, isto é, oferece um serviço que consegue orquestrar cada uma das *slices* para atender aos SLA's preestabelecidos. Por conta disso, naturalmente há a demanda por um serviço de monitoramento de cada *slice*. O monitoramento de *slices* considera todos os seus componentes (físicos e virtuais), bem como os serviços nela instanciados. A partir dos dados de monitoramento, uma infinidade de mecanismos podem ser implementados, não apenas para a orquestração das *slices*, mas também na prevenção de ataques e aplicação de políticas de segurança.

Entretanto, na medida em que as *slices* se tornam mais complexas, ou seja, possuem uma complexa estrutura, formada por muitos e diversos componentes, a tarefa de monitoramento pode se tornar ineficiente. Por exemplo, a quantidade de informações a serem coletadas pode atingir um volume que inviabiliza até mesmo a operação dos serviços instanciados nas *slices*.

Especificamente nesse contexto do monitoramento das *slices*, este trabalho investiga de que forma os mecanismos de aprendizado de máquina podem auxiliar na eficiência do mecanismo de monitoramento do projeto NECOS. Uma vez que as *slices* possuem composição infraestrutural variada e podem hospedar diferentes tipos de serviços, o monitoramento tende a tornar-se complexo, produzindo um expressivo volume de dados de monitoramento. Sendo assim, como ponto de partida, este trabalho considera aplicar técnicas de aprendizado de máquina que auxiliem na filtragem do que precisa, de fato, ser monitorado.

Denominadas de técnicas de seleção de características, elas automatizam a construção de conjuntos essenciais de dados a serem monitorados em cada *slice*. Os dados considerados essenciais podem, por exemplo, estar relacionados ao conjunto de dados que permitem uma melhor orquestração dos recursos das *slices*, ou que permitem a detecção de ataques de segurança.

Neste trabalho, buscamos construir um conjunto de dados de monitoramento que permite representar o nível de serviço sendo ofertado aos usuários finais das *slices*, ou seja, o sistema de monitoramento inteligente proposto neste trabalho busca identificar na infraestrutura das *slices* o conjunto de métricas que pode ser mapeado, com precisão, na situação atual dos serviços.

1.1 Motivação

Em primeiro lugar, há de se ressaltar a importância do módulo de monitoramento no contexto do NECOS. Uma vez que o sistema ofertará alguns níveis de gerenciamento

da *slice*, é de suma importância que a plataforma seja capaz de prevenir problemas nos serviços hospedados. No contexto do NECOS, há medidas proativas, de forma que é necessário o monitoramento constante da infraestrutura para que o sistema seja capaz de predizer se o serviço sofrerá ou não perda de qualidade em um futuro próximo, para que então possa se tomar medidas preventivas a tempo.

Predizer métricas de *Quality of Service* (QoS) é um desafio, porque para que isso seja possível é necessário coletar um volume de dados expressivo da infra-estrutura. Por exemplo, monitorar uso de *Central Processing Unit* (CPU), memória da máquina, uso de memória pelo processo, disco e largura de banda. Todo este volume de dados dificulta sua manipulação, e além disso é preciso processá-los, selecionando os que mais importam, isto é, os dados que mais influenciam na qualidade do serviço. Para este trabalho, uma métrica da infra-estrutura é dita relevante se alterações nos seus valores provocam alterações no serviço hospedado.

Alguns trabalhos, como (PASQUINI; STADLER, 2017), sugerem a utilização de algoritmos de inteligência artificial para que se possa realizar a seleção de características. Essa seleção realiza uma filtragem nos dados, permitindo que se trabalhe com conjuntos bem menores sem perder de vista a margem de erro da predição, isto é, se com o conjunto completo de dados era possível prever com uma margem de erro inferior a, por exemplo, 20%, com o conjunto reduzido não se deseja ficar muito longe disso.

Entretanto, há aqui um problema a mais: a infraestrutura sofre alterações constantemente. Suponha um serviço de leitura e escrita em uma *Distributed hash table* (DHT) hospedado em uma *slice* gerenciada pelo NECOS. Neste caso, a medida que cresce o número de usuários, pode ser necessário subir outra instancia do serviço; melhorar a capacidade de processamento da máquina; aumentar quantidade de memória *Random Access Memory* (RAM) ou de disco; aumentar a largura de banda. Mas, se as características são selecionadas com base em algoritmos de aprendizagem de máquina que foram alimentados com os dados da infraestrutura em um determinado estado, será verdade que, uma vez alterada essa infraestrutura, o conjunto de dados relevantes se mantenha o mesmo? Será que ao alterar o número de instâncias do serviço, ou os recursos da máquina hospedeira, não seja necessário fazer todo o aprendizado novamente?

Buscar responder esses questionamentos, aliado ao desafio que a seleção de características representa em conjunto com a necessidade da plataforma NECOS, foi o que motivou este trabalho.

1.2 Objetivos e Desafios da Pesquisa

Essa seção apresenta o objetivo geral e os objetivos específicos da pesquisa realizada.

- Objetivo geral:

Construir um módulo de monitoramento que identifica e coleta um conjunto reduzido e essencial de características de cada *slice* utilizando aprendizado de máquina. Este conjunto essencial de características deve suportar as diferentes demandas de gerenciamento das *slices*, buscando manter a qualidade dos serviços nelas instanciados.

❑ Objetivos específicos:

1. Avaliar ferramentas de coletas dos dados no sentido de granularidade dos dados coletados e possibilidade de filtragem dos dados, isto é, capacidade de descartar informações não relevantes, sendo considerado tão melhor quanto mais próximo da fonte dos dados estas informações possam ser descartadas;

2. Avaliar se os algoritmos retornam um bom conjunto reduzido de características, ou seja, se as características selecionadas permitem uma boa estimativa de qualidade dos serviços analisados;

3. Avaliar o quanto o volume de dados trafegados na rede é reduzido após a seleção;

4. Avaliar se o conjunto de características selecionadas continua sendo adequado, isto é, continua permitindo boas predições, ainda que a infraestrutura sofra alterações.

1.3 Hipótese

Para este trabalho, há três hipóteses principais que, se comprovadas, levam ao cumprimento do objetivo geral. Estas são denominadas H1, H2 e H3 e são apresentadas abaixo, bem como as perguntas que serão respondidas para comprová-las ou não:

H1. Existe um conjunto de características assertivo, isto é, um conjunto reduzido em relação ao conjunto original (completo) de características que permite uma boa predição de métricas de qualidade de serviço.

P1.1 É possível selecionar os dados do conjunto completo que mais interferem na qualidade dos serviços analisados?

P1.2 Comparando o resultado obtido através do conjunto completo de características com o conjunto reduzido, a predição das métricas sofre alteração significativa, ou seja, a margem de acerto varia de forma a tornar inviável a redução do conjunto?

P1.3 Não havendo alteração significativa e comparando o resultado obtido através do conjunto completo de características com o conjunto reduzido, o tempo de execução para predição das métricas caiu de maneira satisfatória a ponto de ser necessária a redução?

H2. Existe um conjunto de características que permite boas predições de métricas, ainda que a infraestrutura sofra alterações nos recursos ofertados.

P2.1 Será que, alterando os recursos da máquina hospedeira do serviço, não será necessário fazer a seleção novamente?

H3. As ferramentas são capazes de prover dados em granularidade suficiente e permitir a filtragem de características.

P3.1 Existe pelo menos uma ferramenta que cumpre a hipótese H3?

1.4 Contribuições

As principais contribuições deste trabalho são:

- A apresentação de um protótipo de monitoramento que busca atender às necessidades do sistema NECOS;
- Construção de um mecanismo de monitoramento que realiza seleção de características utilizando aprendizado de máquina;
- *Traces* coletados e mantidos em repositório para permitir a reprodução dos experimentos realizados neste trabalho.

1.5 Organização da Dissertação

O restante desta dissertação está organizado da seguinte forma: o Capítulo 2 apresenta o projeto NECOS, os trabalhos relacionados e conceitos básicos para o entendimento do sistema desenvolvido; o Capítulo 3 apresenta a proposta, detalhando as características do projeto; o Capítulo 4 apresenta o ambiente de testes, a implementação do sistema de monitoria; o Capítulo 5 os experimentos e a avaliação dos resultados obtidos; por fim, o Capítulo 6 apresenta as conclusões, contribuições e trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta o projeto NECOS, dando enfoque no módulo de monitoramento proposto inicialmente no sistema, conceitos básicos de monitoria e inteligência artifical. Também são apresentados os trabalhos relacionados, dentre os quais há o que buscou aplicar as técnicas de seleção de características em um cenário similar ao desta dissertação.

2.1 O projeto NECOS

O NECOS propõe o conceito de *Lightweight Slice Defined Cloud* (LSDC), o que significa que o sistema funciona por meio de conjuntos de recursos físicos e virtuais, possibilitando manipular cada serviço de maneira independente, ainda que utilizem a mesma infra-estrutura. Isso permite que o serviço seja implantado atendendo aos requisitos específicos ao longo do tempo (NECOS, 2017).

Uma *slice*, conforme apresentado no Capítulo 1, é uma fatia da infra-estrutura de nuvem que funciona de forma independente das demais. Cada provedor de infra-estrutura pode fornecer uma *slice part*, podendo ser um recurso físico ou virtual, que será utilizado na composição da *slice*. Portanto, uma *slice part* pode ser entendida como uma fatia de rede, disco, CPU.

Na arquitetura do projeto NECOS[1], pode-se encontrar quatro subsistemas: *Tenant's Domain*; *Slice Provider*; *Marketplace*; e *Infrastructure Providers*. Todos suportam o paradigma *slice-as-a-service* proposto, oferecendo *slices* sob demanda aos *Tenants* (NECOS, 2019a) (NECOS, 2019b). O fluxo de criação de uma *slice* no NECOS inicia-se no *Slice Provider* ao receber uma requisição para uma nova *slice* vinda de um *Tenant*. Isto é, informações como quantidade de CPU, tamanho do disco, quantidade de memória RAM, velocidade de *uplink* e *downlink* e descrição do serviço a ser hospedado na *slice*, são fornecidos pelo *Tenant* ao *Slice Provider*, como também o SLA desejado. Em interação com os

[1] http://www.h2020-necos.eu/documents/deliverables/

demais subsistemas, o *Slice Provider* cria uma *slice* fim-a-fim que preenche por completo os requisitos do *Tenant*.

Finalizada a criação da *slice*, o *Slice Provider* passa a monitorá-la para fornecer ao *Tenant*, bem como aos outros módulos do sistema, as informações sobre o estado da infraestrutura na qual a *slice* está hospedada. Esse monitoramento é essencial para o gerenciamento da *slice*, porque além de ela poder sofrer alterações na sua infraestrutura, é necessário saber se a composição atual continua atendendo às necessidades dos serviços sendo ofertados nela.

A Figura 1 apresenta a arquitetura do NECOS, em que é possível observar os diversos componentes do sistema. Entre eles, há o módulo *Infrastructure & Monitoring Abstraction* (IMA) que é responsável pela funcionalidade de monitoramento da infraestrutura compondo as diversas *slices*, melhor detalhado no Capítulo 3. Dentro do IMA, há o componente *Resource & VM Monitoring*, que é o foco deste trabalho de mestrado.

O IMA recebe do módulo de criação da *slice* uma descrição completa desta. Tal descrição contém não somente a topologia da *slice*, com toda a sua composição tecnológica, mas também com seus pontos de acesso, isto é, IPs e portas, para os *Virtual Infrastructure Manager* (VIM)/*Wide-area network Infrastructure Manager* (WIM) que fazem parte da *slice*. Usando tais pontos, o IMA é capaz de monitorar todas as métricas disponíveis.

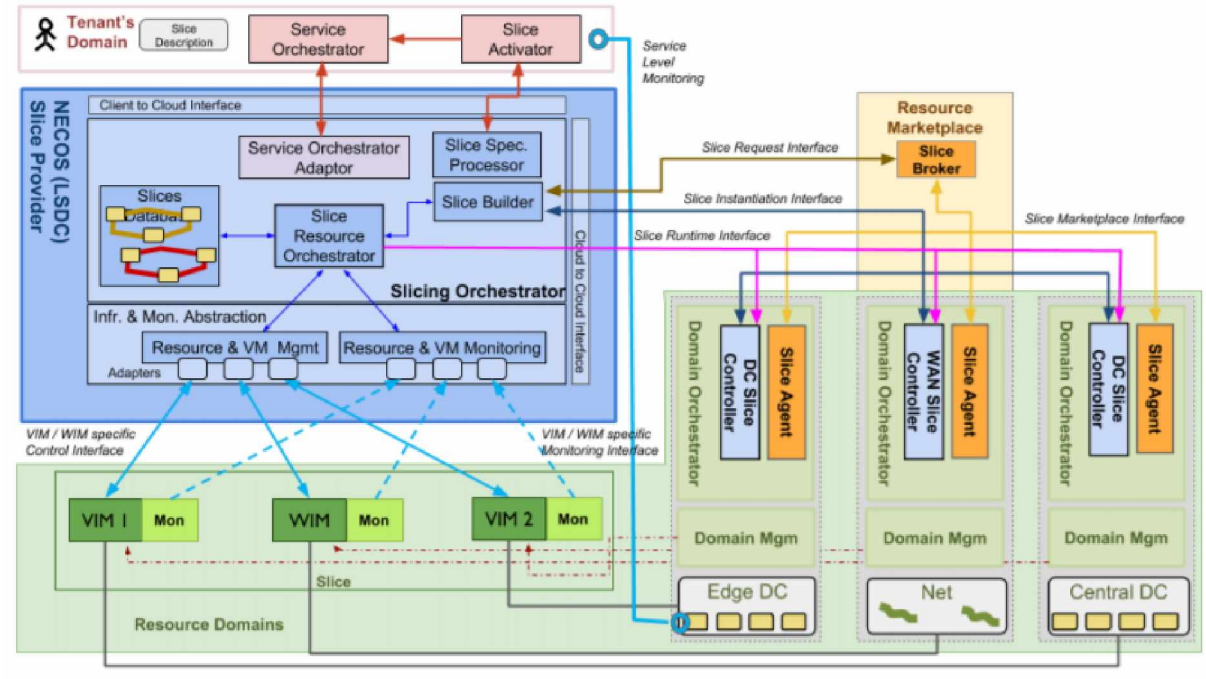

Figura 1 – Arquitetura do sistema NECOS. Retirada de Necos (2019a).

2.2 Conceitos básicos

Essa seção apresenta alguns conceitos que devem ser especificados para o bom entendimento desta dissertação.

2.2.1 Algoritmos de Inteligência Artificial

A literatura diverge quanto a classificação dos algoritmos de inteligência artificial, no que diz respeito à aprendizagem de máquina. Há autores que dão enfoque maior em aprendizagem supervisionada e por reforço, como acontece em (RUSSELL; NORVIG, 1995); há autores que defendem a divisão entre algoritmos de aprendizagem supervisionada e não-supervisionada (LITJENS et al., 2017); e ainda há os que defendem um terceiro conceito, ao lado desses dois últimos, denominado aprendizagem semi-supervisionada, que é uma proposta híbrida, buscando as vantagens dos dois tipos (ZHU, 2005).

No caso de algoritmos de aprendizagem supervisionada, todos os dados são rotulados. Isso significa que para amostragem do conjunto de dados coletados, encontra-se disponível um ou mais rótulos que indicam o que a amostra representa (FULMARI1; CHANDAK, 2013). Os rótulos podem ser discretos, utilizados em algoritmos de classificação, como também podem ser valores contínuos, utilizados em algoritmos de regressão.

Portanto, em aprendizado supervisionado há dois conjuntos de dados essenciais, comumente denominados X e Y (aqui também referenciado como *Key Performance Indicator* (KPI) do serviço) (LITJENS et al., 2017). O conjunto X contém as amostras coletadas para as análises e o conjunto Y contém os rótulos relativos a estas amostras coletadas. Por exemplo, num cenário onde se deseja prever se o dia será ensolarado, chuvoso ou nublado, podem ser analisadas amostras relativas a temperatura, umidade do ar, latitude e longitude. Nesse caso, se construiria um conjunto X formado pelas informações de temperatura, umidade do ar, latitude e longitude, e um conjunto Y com as respectivas condições do dia.

Desta forma, o objetivo de um algoritmo de aprendizado supervisionado é construir uma função F que receba os dois conjuntos de dados, X e Y, e busca estabelecer a relação entre ambos os conjuntos. Assim, quando F for aplicada aos valores encontrados em X, deseja-se obter os respectivos rótulos em Y. Uma vez estabelecida tal função F, esta passa a ser aplicada em novas amostras de X para que se possa estimar o rótulo que existiria em Y.

Há ainda uma divisão quanto ao objetivo de se utilizar um algoritmo de aprendizado de máquina. Nesse contexto, pode-se desejar fazer uma classificação ou uma regressão. Classificação significa que o objetivo é dividir os dados coletados em classes discretas ou categorias. O exemplo mencionado sobre prever a condição do dia é uma forma de classificação. Na regressão deseja-se construir uma função F com base nos valores encontrados

em X e em Y, de tal forma que se possa usar F para estimar valores contínuos de Y, dados novos valores de X.

Para um melhor entendimento, suponha uma situação onde se deseje estimar o salário de um profissional. Nesse caso o conjunto X poderia ser formado de métricas como a idade, o tempo de experiência, a profissão, cidade de outras pessoas e o conjunto Y seria o salário real delas. A partir daí, cria-se a função F que recebe os dados de X e calcula Y. Assim, fornecidos os dados de um novo profissional, que se deseja estimar o salário, o algoritmo chegaria a um valor baseado nos dados que já possui. Nesse caso teria sido realizado uma regressão.

Entretanto, se poderia estar interessado em classificar o salário como ótimo, bom, regular, ruim ou péssimo. Nesse caso, acrescentaríamos ao conjunto X a métrica do valor do salário, enquanto que no conjunto Y haveria a informação de como considerar o salário dos profissionais fornecidos. Nesse caso a resposta do algoritmo é uma classe (ótimo, bom, regular, ruim ou péssimo), razão pela qual o algoritmo seria conhecido na literatura como um algoritmo de classificação.

A dificuldade com os algoritmos de aprendizagem supervisionada é ter de antemão o conjunto Y para que se possa criar a função F. A dificuldade se deve ao fato de que nem sempre os dados são facilmente coletados e, frequentemente, se conhece pouco ou nada sobre os rótulos. Isso se apresenta como um desafio para o uso desse tipo de algoritmo, embora não constitua em uma inviabilidade.

2.2.2 Conceitos de seleção de características

Em Chandrashekar e Sahin (2014), os autores apresentam uma revisão da literatura abordando os algoritmos reconhecidos como estado da arte no que diz respeito à seleção de características. Além disso, é apresentada uma classificação dos algoritmos, dividindo-os segundo a disponibilidade de rotular as informações e também segundo a estratégia de busca implementada para selecionar as características.

Neste sentido, os algoritmos podem ser subdivididos em: supervisionados, aplicados quando todas as características podem ser rotuladas; semi-supervisionados, aplicados quando poucas características podem ser rotuladas; e não-supervisionados, aplicados quando nenhuma característica pode ser rotulada no que tange à disponibilidade de rotular informações;

No que diz respeito à estratégia de busca, a seguinte divisão foi estabelecida:

- *wrapper*: métodos que utilizam o mesmo algoritmo de aprendizagem de máquina tanto no processamento/mineração dos dados quanto para seleção de características;

- *filter*: métodos que primeiro constroem um *ranking* das características baseado em algum critério, para, em seguida, utilizá-lo selecionando as características de acordo

com a posição no *ranking* e posteriormente fazer a classificação ou *clusterização* dos dados;

- *embedded*: métodos que selecionam as melhores características a medida em que se cria o modelo de aprendizagem.

Desta forma, para este trabalho, algoritmos supervisionados utilizando as abordagens *wrapper* e *filter* foram mais explorados, por se adequarem melhor ao cenário visado.

2.2.3 *Stepwise Feature Selection*

Um dos algoritmos de seleção de características utilizado em Pasquini e Stadler (2017) e apresentado em John, Kohavi e Pfleger (1994) é conhecido como *Forward Stepwise Feature Selection*. Seu funcionamento pode ser, resumidamente, descrito assim:

Algoritmo 1 *Forward Stepwise Feature Selection*

Seja N o número de métricas existentes em X.
Seja C_0 um conjunto inicial vazio.
Gera-se N subconjuntos C_i de X, em que cada C_i possui i métricas, com $1 \leq i \leq N$.
O conjunto C_i é construído com base no conjunto C_{i-1}.
Para cada métrica M que ainda não está em C_{i-1}, gera-se a estimativa com o conjunto $C_{i-1} + M$, e o conjunto que obtiver o menor erro, isto é, que mais se aproximar de Y é armazenado como C_i.
Construídos todos os N conjuntos, retorna-se o que tiver alcançado o menor erro.

De forma análoga há o algoritmo *Backward Stepwise Feature Selection*, cujo funcionamento pode ser, resumidamente, descrito assim:

Algoritmo 2 *Backward Stepwise Feature Selection*

Seja N o número de métricas existentes em X.
Seja C_N um conjunto inicial com todas as métricas de X.
Gera-se N subconjuntos C_i de X, em que cada C_i possui i métricas, com $1 \leq i < N$.
O conjunto C_i é construído com base no conjunto C_{i+1}.
Para cada métrica M que está em C_{i+1}, gera-se a estimativa com o conjunto $C_{i+1} - M$, e o conjunto que obtiver o menor erro, isto é, que mais se aproximar de Y é armazenado como C_i.
Construídos todos os N conjuntos, retorna-se o que tiver alcançado o menor erro.

É importante ressaltar que esses algoritmos se utilizam de um outro que seja capaz de gerar as estimativas $\hat{Y}$.

Essas abordagens possuem um custo computacional bastante expressivo, visto que a complexidade de tempo pode ser expressa por $O(N^2 * R)$, em que N é o número de métricas existentes em X e R é a complexidade do algoritmo utilizado para gerar as estimativas. No entanto, o algoritmo *Forward Stepwise Feature Selection* foi testado neste trabalho, em comparação com um algoritmo de correlação.

2.3 Trabalhos correlatos

Esta seção apresenta alguns trabalhos relacionados, apresentando algoritmos de seleção de características, além de um trabalho que buscou aplicar algumas técnicas de seleção de características em um cenário bastante similar ao trabalho apresentado nesta dissertação.

No trabalho Yang, Yoon e Shahabi (2005) há um cenário em que as características são ondas cerebrais emitidas pelo usuário e, com base nelas, procura-se identificar quem são os usuários do sistema. Entretanto, várias são as ondas captadas e a aplicação de um algoritmo de seleção de característica se fez necessária. Para isso, foi proposto um algoritmo novo de seleção de características, nomeado *Corona*, baseado em *Support Vector Machine* (SVM).

Uma vez que em Yang, Yoon e Shahabi (2005) as características são séries temporais multivariadas, isto é, para um mesmo instante de tempo numa mesma série temporal há mais de uma informação a ser analisada, há uma relação direta com o cenário deste trabalho de mestrado, dado que um mesmo serviço pode ter várias instâncias, o que significa que para um mesmo instante de uma mesma métrica poderá existir mais de um valor a ser analisado. Em um serviço hospedado em uma *slice*, em que há mais de um servidor sendo utilizado para ofertá-lo, haverá uma série temporal multivariada, o que permite que se faça uma comparação dos resultados obtidos.

No trabalho Kira e Rendell (1992) foi desenvolvido um novo algoritmo para seleção de características, denominado *Relief*, executado com base em métodos estatísticos que, segundo os autores, não depende de heurísticas. Além disso, é dito que o algoritmo tem tempo linear com relação ao número de características dadas e na quantidade de instâncias de treinamento. Para testar o algoritmo, os autores forneceram um conjunto de características em que apenas duas delas eram consideradas relevantes. Em um dos piores resultados apresentados, o algoritmo *Relief* conseguiu encontrá-las em 70% dos casos, com um tempo relativamente baixo de aprendizado. Por conta da similaridade do cenário, o *Relief* poderia ser testado como algoritmo de seleção de características no sistema desenvolvido nesta dissertação.

Em John, Kohavi e Pfleger (1994) é apresentado um algoritmo de aprendizado supervisionado que determina o grau de relevância de uma característica. No método proposto, os autores compararam as abordagens *backward stepwise elimination* e *forward stepwise elimination*. Essas abordagens são baseadas, respectivamente, nos algoritmos de *backward elimination* e *forward elimination*. O primeiro consiste em inicialmente se tomar o conjunto completo de características e, de forma gulosa, retirar aquela em que o conjunto resultante é melhor que o conjunto atual, ou que o conjunto restante seja ligeiramente pior, mas que compense a diminuição de uma característica. O segundo funciona de forma análoga, porém iniciando com um conjunto vazio.

As abordagens comparadas adicionam uma melhoria nos algoritmos *backward elimination* e *forward elimination*. Esta melhoria consiste em se considerar a ação de adicionar

ou remover características a cada passo. No primeiro algoritmo há apenas remoção, enquanto no segundo há apenas adição. Mas com a melhoria, tomando o *backward stepwise elimination* como exemplo, é possível, em cada passo, adicionar características que foram removidas, caso estas melhorem a performance do conjunto resultante. Desta forma, tal método é aplicável para executar os algoritmos de seleção de características no sistema proposto neste trabalho.

Em Pasquini e Stadler (2017) o cenário é bem relacionado com o deste trabalho. O objetivo era prever as métricas de qualidade de serviço de vídeo sob demanda e de leitura e escrita de dados em uma DHT. Para isso, os autores compararam dois algoritmos de aprendizado de máquina (*random forest* e *regression tree*). Ambos os algoritmos também foram utilizados na construção de conjuntos reduzidos de características correlacionadas as da qualidade do serviço. Os autores chegaram a conclusões relevantes, atestando que, embora as predições passassem a ser feitas com base no conjunto reduzido de métricas, o erro não subiu de maneira substancial, e em alguns casos o erro foi bastante próximo do obtido com o conjunto completo. Dos dois serviços apresentados em Pasquini e Stadler (2017), o de leitura e escrita em uma DHT também foi avaliado no contexto deste trabalho, porém com uma importante modificação: será avaliado o serviço do Cassandra Apache (2016) e não o Voldemort (VOLDEMORT, 2016).

Capítulo 3

Proposta

O NECOS auxilia os *Tenants* na construção de *slices* sob demanda. Este processo inclui desde uma simples combinação de recursos para construção da *slice* até um gerenciamento complexo e automático, que consiste em alterações nos recursos que a compõem, ou mesmo do local físico em que ela está hospedada. Ao requisitar uma *slice*, o *Tenant* opta, entre outros pontos, pelo nível de gerenciamento que necessita. Esses níveis de gerenciamento podem ir desde o nível 0, quando o *Tenant* gerencia a *slice* por conta própria, até o nível 3, em que o *Tenant* deseja que o NECOS faça todo o controle. Neste caso, a solução proposta pelo NECOS também é responsável por realizar, por exemplo, as alterações nas *slices* visando manter o SLA (NECOS, 2019a).

Desta forma, para suportar tais funcionalidades em todos os níveis de gerenciamento, o NECOS possui um sistema de monitoramento contido no módulo IMA. Esse sistema permite a coleta de métricas a partir dos diferentes provedores de infraestrutura presentes na federação, trabalhando com diferentes tecnologias e atuando em diferentes granularidades de coleta.

3.1 Monitoria no sistema NECOS

Conforme a Figura 2, o IMA provê uma camada abstrata acima do VIM / WIM, que são os componentes responsáveis por expor as métricas das *slices* e permitir o seu gerenciamento (NECOS, 2019a). Para isso, este módulo do NECOS é constituído de dois subcomponentes principais: *Resource & VM Management* (RVMgt) e *Resource & VM Monitoring* (RVMnt).

O primeiro é responsável por permitir a interação de outros módulos do NECOS com os VIM / WIM das diferentes *slice parts*. Essa interação tem por intuito gerenciar a *slice*, segundo os níveis ofertados pelo NECOS, e ela se dá via *Application Programming Interface* (API) exposta pelo RVMgt (NECOS, 2019a).

Já o RVMnt é responsável por coletar os dados de monitoria, disponibilizados pelos VIM / WIM, permitindo que se possa acompanhar, de forma uniformizada, o estado das

diversas *slice parts*. Esses dados são métricas coletadas da infraestrutura e que estão relacionadas à topologia dos recursos de cada *slice part*; aos recursos propriamente ditos; ao estado de cada recurso virtual, para, por exemplo, detectar falhas; bem como métricas relacionadas diretamente com o KPI, isto é, métricas que influenciam diretamente na qualidade do serviço hospedado, tais como uso de CPU, memória e rede, por exemplo. Além disso, o IMA também deve expor as métricas coletadas para os demais módulos do sistema, em especial o módulo *Service and Resource Orchestrator* (SRO).

O SRO, como o próprio nome diz, é o módulo do sistema responsável por orquestrar recursos e serviços ofertados nas *slices*. Isso significa que este módulo realiza as mudanças necessárias da infraestrutura, que permitirão um melhor funcionamento do serviço nela hospedado.

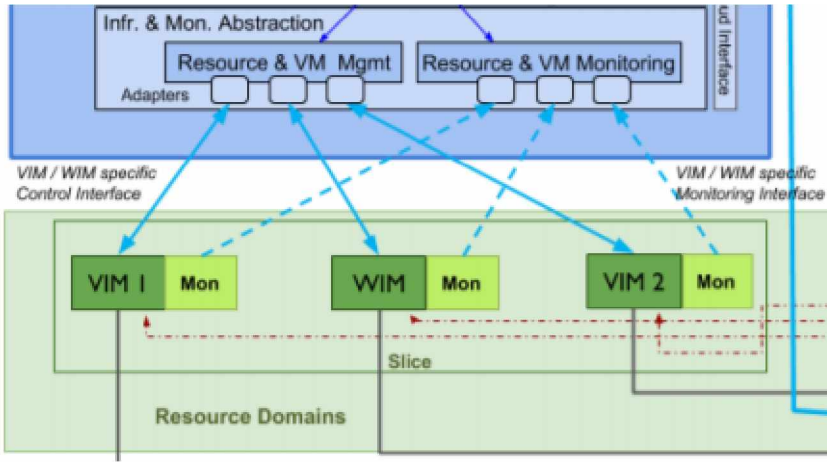

Figura 2 – Ilustração da arquitetura original do IMA no projeto NECOS.

No entanto, para que essa uniformização entre os diferentes provedores seja possível, é preciso passar por algumas questões importantes. Dentro da proposta de gerenciamento estabelecida pelo NECOS, percebe-se a necessidade de o sistema executar comandos e instruções nas diversas *slice parts*, para orquestrá-las segundo a necessidade. Mas, nessa perspectiva, será verdade que todos os provedores possuam o mesmo conjunto de instruções e comandos? É natural pensar que não, e por isso, questiona-se: como o IMA poderá gerenciar uma *slice* se ele não conhece os comandos que pode ou não pode executar?

E no contexto do componente de monitoria, indaga-se: quem se responsabilizará por gerar os dados da infraestrutura? Como esses dados serão disponibilizados? Será via API, via arquivos? Haverá uma sintaxe pré-determinada ou o IMA será flexível nesse sentido? Qual tecnologia de monitoração será escolhida e porquê? O *Tenant* poderá opinar nessa escolha?

Os cenários que se desdobram da análise desse problema são diversos o que demonstra a sua complexidade e evidenciam a necessidade da discussão.

Concentrando a análise no ponto de vista da monitoria, que de forma análoga se estende ao componente de gerenciamento, em um primeiro momento, poderia-se sugerir

que o NECOS definisse um padrão de exportação de métricas, e forçasse cada provedor a se adequar a ele. Nesse caso, o *Tenant* não teria o que opinar e o IMA ficaria apenas com a responsabilidade de ler essas métricas e ofertá-las a outros módulos do sistema que necessitem dos dados.

No entanto, essa abordagem pede que provedores que já têm um modo de trabalho, já possuem um fluxo de implantação, se adéquem a um novo sistema que se propõe a facilitar a interação do *Tenant* com o provedor. Isso gera uma complexidade para os provedores, que não parece viável, já que apenas o NECOS se aproveitaria dessa funcionalidade.

Portanto, a hipótese que aliviaria o IMA deixando-o menos complexo termina por ser impraticável dadas as exigências a serem feitas. Parece mais lógico o NECOS se adequar aos provedores e se responsabilizar pelas métricas, do que tentar o oposto.

Neste caso, um caminho possível é sempre optar pela mesma tecnologia de monitoração para toda e qualquer *slice part*. Entretanto, isso eventualmente criaria complicações para o *Tenant*, que pode ter maior familiaridade com outra tecnologia que não a definida pelo NECOS, como também, no caso de uma atualização da tecnologia, poderia criar embaraços para a *slice* já instanciada. Além disso, é justo considerar se a forma de monitoria não precisa ser analisada conforme o tipo de infraestrutura e conforme o provedor.

Por isso, o cenário mais interessante, do ponto de vista do *Tenant* e do ponto de vista dos provedores, é que o NECOS permita que cada *Tenant* determine a tecnologia de monitoramento a ser utilizada em cada *slice part*. Essa abordagem desobriga os provedores de se adequarem ao NECOS, permite que o *Tenant* opte pela tecnologia que mais lhe convenha e também não impede que o NECOS possua uma sugestão, no caso de o cliente não ter alguma preferência. Todavia, tal caminho deixa a implantação das *slices* um pouco mais complexa e acarreta uma dificuldade para o IMA como um todo.

Portanto, dado que cada uma das partes que compõem a infraestrutura da *slice* pode ter uma tecnologia de monitoramento diferente, é natural pensar que elas poderão ser distintas entre si no que diz respeito à formatação, à geração e à exposição das métricas. Isso quer dizer, por exemplo, que as métricas podem ser disponibilizadas ora em formato de arquivo *txt*, ora *Comma-Separated-Values* (CSV); podem também ser expostas via API em formato de *JSON*, ou *XML*; enfim, as possibilidades são inúmeras. A geração também importa, porque pode ocorrer de determinada tecnologia coletar uma métrica, que outra não coleta, e isso pode representar um outro problema, que será discutido mais a frente.

Isto posto, fica evidente que deve-se pensar em como estabelecer a comunicação entre o componente de monitoria e as diversas *slice parts* gerenciadas pelo NECOS. Então a questão agora é: como estabelecer essa comunicação?

Uma abordagem possível seria criar uma espécie de tradutor que fosse capaz de interpretar tanto a linguagem do NECOS, quanto a linguagem de cada tecnologia, fazendo o papel de converter o padrão das métricas exportadas em um padrão que facilite o gerenciamento do NECOS. Evidentemente que essa hipótese possui um problema que é demandar

que o NECOS construa, para cada tecnologia de monitoramento, esse tradutor que deve ser anexado ao RVMnt permitindo a comunicação. Surgindo uma nova tecnologia, ou havendo atualizações nas já existentes, haveria demanda de desenvolvimento, o que cria uma necessidade de atualizações constantes e pessoas dedicadas para isso.

Apesar de não ser perfeita, essa abordagem foi escolhida, o que demonstra o porquê de o RVMnt e o RVMgt possuirem, na Figura 2, caixinhas denominadas *Adaptors*. Nesse caso, para cada *slice part*, o IMA deverá contar com um adaptador capaz de traduzir a métrica exportada para a sintaxe esperada pelo RVMnt, como também traduzir os comandos de gerenciamento feitos pelo RVMgt em comandos que aquela *slice part* consegue executar. Desta forma, o NECOS pode, de maneira mais simples, se comunicar com a infraestrutura.

3.1.1 Considerações sobre o Funcionamento do Sistema de Monitoria

Dentro da solução inicial do NECOS, o componente de monitoria se utiliza dos adaptadores para coletar as métricas e armazená-las. Tão logo isso aconteça, os módulos de gerenciamento, tais como o de Segurança, o de Controle de Políticas, o Orquestrador de Serviços e Recursos (SRO), podem consumir essas informações, viabilizando o gerenciamento da *slice*. Isso significa, por exemplo, que o SRO ao examinar uma métrica, qual seja a de uso de CPU, perceba que o recurso está sendo quase que totalmente utilizado, então poderá aumentar o número de CPUs da *slice*. Tal cenário também se aplica à capacidade de rede, à quantidade de memória RAM, e até mesmo à quantidade de instâncias daquele determinado serviço, bem como a sua localização. E da mesma forma que é possível colocar recursos, deve ser possível retirá-los, no caso de estarem ociosos, o que reduz custos e pode ser interessante para o *Tenant*.

Entretanto, ainda aqui há questões a serem resolvidas. Haverá uma única instância do módulo de monitora para todas as *slices* ou existirá uma instância particular para cada *slice* ou para cada grupo delas? Os *Adaptors* são parte dessa(s) instância(s), ou são microsserviços à parte, usados por ela(s)?

Ter apenas uma instância do módulo de monitoria pode simplificar o desenvolvimento, mas por outro lado parece ir na contramão dos grandes sistemas que se tem hoje em produção. De fato, há uma gama de razões evidentes para que se queira distribuir a carga em diversas instâncias: permitir atender mais clientes ao mesmo tempo, pode reduzir a latência, tendo em vista que o cliente pode ser atendido pela instância fisicamente mais próxima, como também diminuir consideravelmente a chance de indisponibilidade.

Um dos problemas para os sistemas onde se opta por trabalhar de forma distribuída é o banco de dados. Sincronizar os bancos, ou ter apenas um banco? É uma questão importante, porque os problemas de concorrência e de inconsistência precisam de atenção.

No entanto, o cenário aqui tem uma característica que favorece bastante a implantação com várias instâncias: pode-se ter vários bancos de dados e eles não precisam conhecer uns aos outros. Isso porque as informações de uma *slice*, ou as métricas coletadas de sua infraestrutura não têm relação com as informações de uma outra, o que permite dizer que os bancos podem trabalhar de forma isolada.

Portanto, uma hipótese viável seria criar uma instância do módulo de monitoria e também uma instância do banco de dados, sem ter necessidade de sincronizá-los. Evidentemente que se poderia ter apenas um banco de dados, porque não deve haver concorrência de leitura ou escrita em virtude das métricas da *slice*, no entanto distribuir também as instâncias de banco de dados pode favorecer o seu funcionamento, dado que trabalhará com um menor volume de informação e com um menor número de acessos.

Com relação aos *Adaptors* a melhor decisão parece precisar de testes para que seja tomada com segurança. Aparentemente não há problemas em se ter apenas uma instância do *Adaptor* por tecnologia de monitoração. Entretanto, a ideia de que cada instância do módulo de monitoria possua a sua instância do *Adaptor* parece igualmente adequada. Se por um lado se teria apenas um canal de comunicação - primeira hipótese -, a resposta poderia ser assíncrona o que permite um trabalho paralelizado, por outro lado rodar um *Adaptor* por instância de monitoria poderia facilitar o trabalho - segunda hipótese.

Uma vez que o cenário vai ficando cada vez mais complexo e amplo, é necessário tomar decisões, baseadas em argumentos que as sustentem e que viabilizem o trabalho aqui desenvolvido. Nesse sentido, buscando apresentar uma forma de lidar com as questões levantadas e os problemas destacados anteriormente, a Figura 3 ilustra uma proposta de fluxo de monitoria do NECOS, o qual envolve principalmente o IMA. O processo de monitoramento se inicia com a alocação de recursos nos provedores de infraestrutura para uma determinada *slice*. Também são apresentados três *slices* formadas por diferentes *slice parts*.

A comunicação dos módulos do sistema com as *slices* se dá via API, conforme o conceito de *Slice-as-a-Service* criado pelo NECOS. As diferentes cores utilizadas no desenho da *slice* exemplificam as *slice parts*, sendo provisionadas pelo provedor com a respectiva cor. Nesta proposta, optou-se pela abordagem de distribuir a tarefa de monitoria em diversas instâncias do componente, razão pela qual há M *slices* ilustradas e M instâncias do sistema de monitoria.

Com relação aos *Adaptors*, optou-se por serem módulos integrantes do componente de monitoria, em que cada instância do sistema possui o(s) seu(s) adaptador(es) para a(s) *slice part(s)* que possa(m) ter que monitorar. Por essa razão, nota-se que cada instância do sistema de monitoria possui o seu conjunto de adaptadores, um para cada tipo de infraestrutura, demonstrando a necessidade de comunicar-se com diferentes tecnologias de monitoração implantados nas diferentes *slice parts*. A escolha dessa abordagem está bem relacionada com o que já está, de certa forma, proposto no NECOS, uma vez que

ela também pode ser vista na Figura 2, onde se nota que os *Adaptors* são membros tanto do subcomponente RVMgnt quanto do RVMnt.

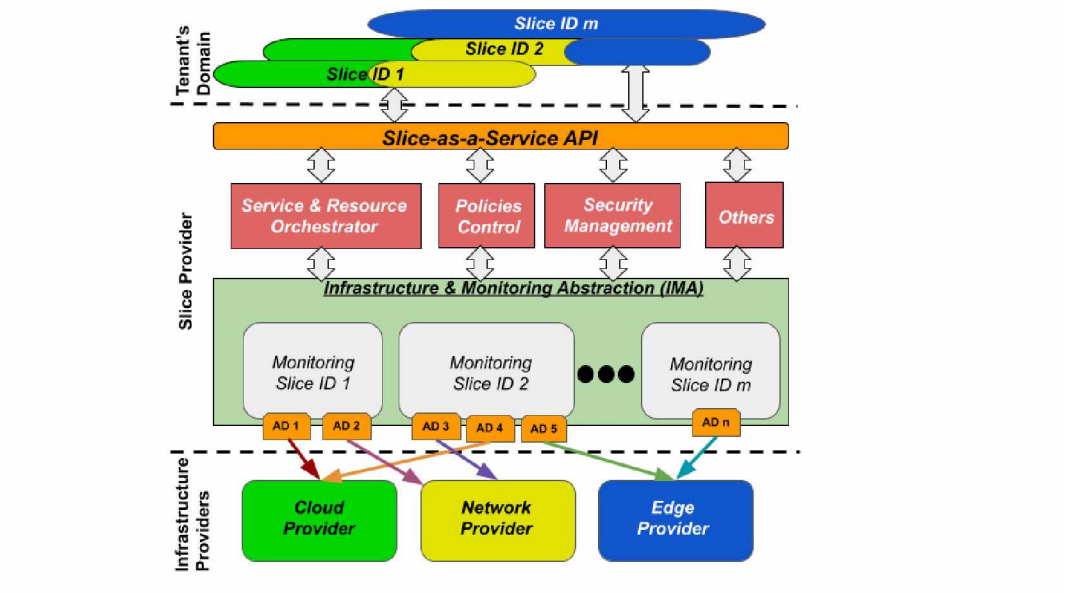

Figura 3 – Visão geral da função do IMA no NECOS.

3.1.2 Considerações sobre as Métricas a Serem Monitoradas

A solução original do módulo IMA, proposta no NECOS, não lida com algumas dificuldades que são encontradas em sistemas que se dedicam à tarefa de monitoria de infraestrutura. Embora essas ainda não possuam uma solução definitiva na literatura, no contexto do NECOS faltam algumas tratativas que poderiam, pelo menos, amenizá-las. Por isso mesmo, o trabalho desenvolvido nessa dissertação buscou olhar para esses problemas visando propor um complemento ao IMA, tornando-o mais robusto e capaz de funcionar em um cenário real.

Uma vez que o NECOS se propõe a ser um sistema escalável, abre-se a necessidade de lidar com um volume expressivo de dados coletados da infraestrutura. E uma grande quantidade de dados a serem analisados gera, pelo menos, dois problemas que precisam ser considerados: o primeiro é a sobrecarga da rede durante o tráfego dessas informações, o que pode prejudicar o monitoramento e a disponibilidade do sistema; e o segundo é a dificuldade de tomada de decisão de módulos adjacentes ao IMA, como o SRO, uma vez que o excesso de dados pode acabar gerando ruídos que prejudicam a análise, tornando-a menos precisa e/ou até mesmo mais lenta e, a depender do caso, inviável.

Estabelecidos os dois problemas principais, que naturalmente podem levar, por exemplo, à indisponibilidade do sistema e à lentidão e mau funcionamento do gerenciamento, uma solução que poderia ajudar é assumir sempre um mesmo conjunto reduzido de métricas a serem selecionadas e analisadas.

Tal abordagem parece uma boa opção, porque é de conhecimento geral que existem recursos que influenciam diretamente no funcionamento de uma máquina onde se deseja hospedar um serviço. Por exemplo: quantidade de memória RAM, número de CPU's, tipo de processador, largura de banda.

Entretanto, ainda que definir um conjunto reduzido de métricas seja uma boa abordagem, ela possui grandes pontos de dúvidas e falhas. Suponha um servidor em que o uso de CPU e RAM não estejam altos, o processador esteja trabalhando dentro do limite e a largura de banda esteja suficiente para atender às demandas de *download* e *upload* do serviço hospedado. Deve-se daí concluir que o serviço está funcionando bem? E se ele realizar operações, cujo tempo de resposta é longo e, embora haja recursos disponíveis, ele não esteja sendo capaz de utilizá-los? Se a forma de trabalho daquele serviço em especial não pode paralelizar seu processamento, segue que os recursos não podem ser usados, embora estejam livres. Nesse caso, a solução seria uma nova instância do serviço e não adicionar memória RAM ou CPU's à máquina.

Além disso, na Seção 3.1 foi discutido que os dados da infraestrutura serão gerados por diferentes tecnologias, que podem variar no conjunto de métricas ofertadas. Nesse caso, como pode ser possível que se fixe um determinado conjunto de métricas a serem analisadas, se não se sabe se elas de fato serão geradas? Ainda que o *Adaptor* possa converter determinada métrica na informação esperada, pode ocorrer de simplesmente não existir uma métrica relacionada com o que se deseja. Para tanto, seria preciso uma análise das tecnologias disponíveis, podendo concluir-se que tal ou qual tecnologia não atenda os padrões do NECOS. Porém, esse cenário não é diferente daquele outro, em que o NECOS impõe ao *Tenant* a tecnologia a ser utilizada, podendo, assim, criar inconvenientes para o seu cliente.

Portanto, com uma análise rápida é possível perceber duas necessidades do componente de monitoria. A primeira é que as métricas do próprio serviço precisam ser levadas em conta, para que se faça uma análise correta do seu estado, pois é avaliando a qualidade do serviço que se gerará a necessidade de orquestração de recursos ou, possivelmente, de instâncias do serviço, como citado no primeiro exemplo. Isso inviabiliza o uso de um conjunto fixo de métricas, pois que os serviços a serem hospedados não são previamente conhecidos e, portanto, não se tem conhecimento das suas métricas. Ou, mesmo que se suponha que haja tal conhecimento, seria isto viável do ponto de vista de um sistema escalável e de grandes proporções, como o NECOS se propõe a ser? Além disso, qual o critério para selecionar esse conjunto fixo? Quem faria esse trabalho? Seria possível atestar que as mesmas métricas podem ser usadas para tomar decisões tendo o serviço uma instância, ou duas, ou três, ou dez?

A segunda necessidade percebida é que o componente de monitoria precisa se adequar às diversas tecnologias de monitoração, porque do contrário o NECOS pode não atender uma demanda dos *Tenants*. Portanto, é preciso que haja uma certa flexibilização no

tratamento dos dados da infraestrutura e, mais uma vez, percebe-se a inviabilidade de se pré-determinar um conjunto fixo de métricas a serem coletadas e analisadas.

Assim, tendo em vista que o problema é maior do que se poderia supor, o trabalho desenvolvido nessa dissertação buscou abordagens que pudessem auxiliar nesse cenário delicado e complexo, o que será discutido na Seção 3.2 a seguir.

3.2 Abordagem proposta

Diante do cenário do NECOS e dos desafios levantados com relação à monitoria, as técnicas de seleção de características, presentes na literatura, em conjunto com o uso de inteligência artificial e aprendizado de máquina ganham espaço e podem ser uma boa opção, fazendo com que o sistema de monitoria seja mais dinâmico e viável.

Tal abordagem pode ser aplicada visando, principalmente:

- reduzir os dados trafegados entre o IMA e as *slices*, bem como entre os demais módulos do projeto;
- deixar a monitoria mais flexível, no que diz respeito à manipulação e seleção das métricas, abandonando a ideia de um conjunto fixo e buscando gerar um conjunto reduzido de métricas de forma dinâmica e adaptável ao serviço e à *slice part*; e
- promover a diminuição de ruídos, possibilitando uma melhor análise dos dados. Essa análise é importante, porque através dela, módulos, como o SRO, podem fazer previsões de métricas de QoS e com isso evitar degradações do serviço hospedado. A previsão será tão melhor quanto menos ruídos estiverem presentes nos dados analisados, mas, por outro lado, a seleção de métricas também deve se atentar para não deixar faltar informações essenciais para a predição.

Assim, a proposta desta dissertação é uma melhoria no sistema de monitoramento atual do NECOS discutido anteriormente, inserindo uma inteligência capaz de se utilizar das técnicas de seleção de características para reduzir, tanto quanto possível, o conjunto de métricas coletadas, de forma que a análise dos dados não sofra pioras significativas, ou que, possivelmente, ela possa ser melhorada.

Essa inteligência deverá, também, cumprir o requisito de seleção dinâmica das métricas (facilitando a adequação aos mais variados contextos dos serviços), bem como atender à necessidade de se levar em consideração as características do próprio serviço, possibilitando que um bom conjunto reduzido de métricas de infraestrutura seja construído. Esse conjunto será tão melhor quanto mais precisas forem as previsões que possam ser feitas a partir dele.

O que se deseja nesse caso é olhar para um conjunto reduzido e dinamicamente gerado de dados da infraestrutura que compõe uma *slice*, para que se possa indicar, com alguma

antecedência, a qualidade do serviço em um tempo futuro, tendo o estado atual dessas métricas selecionadas. Isso permitirá uma melhor precisão no gerenciamento ofertado pelo NECOS, otimizando o uso dos recursos disponíveis.

Para cumprir esse objetivo, faz-se necessário o uso de algoritmos de aprendizado de máquina. Assim, o sistema poderá de forma independente estudar as métricas e decidir sobre quais precisam ser coletadas, o que reduz o conjunto a ser analisado e, consequentemente, diminui o tráfego na rede de controle.

Por conta disso, a inteligência do sistema, doravante denominado *SIMON*, se baseia em um algoritmo de aprendizagem de máquina supervisionada, em que se possa aplicar tanto a técnica *wrapper* quanto a *filter*, ambas apresentadas no Capítulo 2. Este algoritmo poderia ser, por exemplo, o *Random Forest* Breiman (2001), ou ainda o *Regression Tree* conforme utilizado em Pasquini e Stadler (2017). Pode-se também buscar implementar o algoritmo *Corona* proposto em Yang, Yoon e Shahabi (2005), ou ainda um algoritmo que seja aplicável ao cenário que estiver em implementação.

Para a coleta de métricas da infraestrutura o *SIMON* pode ser incrementado com diversas ferramentas já disponíveis tais como *Prometheus* Prometheus (2014c), *Ceilometer* Ceilometer (2018) ou mesmo o *SAR* SAR (2018), simulando a escolha do *Tenant*, sendo necessária a implementação dos adaptadores mencionados.

Além disso, deve-se expor uma API que receba o pedido de seleção de métricas e, de forma assíncrona, retorne o conjunto X das métricas selecionadas.

3.2.1 Desafios da Abordagem Proposta

Dado que o *SIMON* se utiliza de um algoritmo de aprendizagem supervisionada, faz-se necessário resolver o problema da obtenção dos rótulos, ou, em outras palavras, o problema da obtenção do conjunto Y, explicado no Capítulo 2.

Dentro do contexto do NECOS, assume-se que o *Tenant* fornecerá os dados de qualidade do serviço hospedado na *slice* para permitir seu gerenciamento. Ao lado disso, considerando que o contexto do NECOS envolve, principalmente, a disponibilização de serviços em nuvem, uma outra proposta para a geração do conjunto Y é o próprio NECOS instanciar um cliente real deste serviço, de forma que as métricas possam ser conhecidas. Assim, assume-se que o problema da obtenção dos rótulos dos dados está resolvido e o sistema pode ser implantado.

O conjunto Y pode ser então enviado aos módulos de gerenciamento, que por sua vez repassam para o IMA, possibilitando, assim, a seleção de características. Este conjunto Y pode ser um conjunto de informações do serviço ou, até mesmo, uma métrica coletada na própria infraestrutura, cujo módulo de gerenciamento tem relação e/ou interesse.

No entanto, um outro desafio se apresenta quando se fala de conjunto reduzido de métricas a ser analisado. Sendo o objetivo principal a redução do conjunto total de métricas sem perdas significativas de precisão na análise da qualidade de serviço, é preciso

pensar sobre como decidir o tamanho desse conjunto a ser construído. Deve-se fixar em um número K de métricas a serem selecionadas ou deixar que o próprio sistema decida a melhor quantidade?

No primeiro caso é fácil notar que métricas importantes poderiam ser excluídas, ou até mesmo ruídos poderiam ser adicionados ao conjunto. Suponha que o número ideal de métricas seja $K = 15$. Se um $K = 10$ for definido, faltariam métricas importantes no conjunto gerado. Mas, ao mesmo tempo, se se determinar um $K = 20$, o resultado traria métricas desnecessárias (ruídos). No entanto, essa abordagem tem o benefício de que a geração do resultado se dá, em média, de maneira mais rápida (tendo em vista que o número a ser selecionado deve ser bem menor que a quantidade total, o que faria com que a seleção fosse interrompida tão logo se atingisse o conjunto com K métricas).

Já na segunda hipótese, há o benefício de se trabalhar com um conjunto teoricamente melhor, com menos ruídos e sem eliminar informações necessárias, mas traz o inconveniente do tempo de geração do conjunto, já que, normalmente, mais subconjuntos precisariam ser analisados até que se decida qual o melhor de todos.

Uma vez que as duas abordagens possuem bons pontos positivos e consideráveis pontos negativos, optou-se por atender aos dois casos, deixando a cargo do usuário qual caminho seguir. De modo geral, espera-se que o K seja ofertado, mas se não o for, o *SIMON* se encarrega de gerar o conjunto mesmo assim.

Independente do caso, uma abordagem possível é adotar um dos algoritmos de *Stepwise Feature Selection* propostos no Capítulo 2. Tendo o K, a análise pára ao se definir o melhor conjunto com K métricas, mas na ausência deste, o algoritmo gera todos os conjuntos possíveis e retorna o melhor de todos. Porém, esta última é uma abordagem que, para ser utilizada na prática, precisa ser analisada criteriosamente, já que o volume de métricas no contexto do NECOS é expressivo, e um algoritmo com uma complexidade tão alta pode se tornar inviável.

Portanto, espera-se que os módulos de gerenciamento que se utilizam das métricas ofertadas pelo IMA determinem o número de métricas a ser selecionado. Desta forma, os módulos devem repassar ao IMA tanto o conjunto Y quanto o número K de métricas a serem selecionadas. A independência do sistema com relação ao número K é tão somente uma funcionalidade que visa atender a todos os casos possíveis, mas não é recomendada na prática.

O tempo de execução da seleção de características representa um fator importante, mas não determinante, a ser considerado no contexto desta dissertação. É que se considera que o tempo não é um fator que pode inviabilizar a proposta, ainda que leve algumas horas para realizar a seleção, pois esse processo não é executado com grande frequência e funciona de forma assíncrona. Uma vez decidido o conjunto de métricas a ser coletado, a necessidade de renová-lo pode vir apenas de uma mudança na infraestrutura, ou no serviço hospedado, e não a todo instante.

3.2.2 *Workflow* do sistema

Para exemplificar um possível fluxo de trabalho envolvendo o sistema de monitoria, a Figura 4 ilustra um cenário em que o Orquestrador de Recursos solicita a seleção de características ao módulo de monitoramento (IMA).

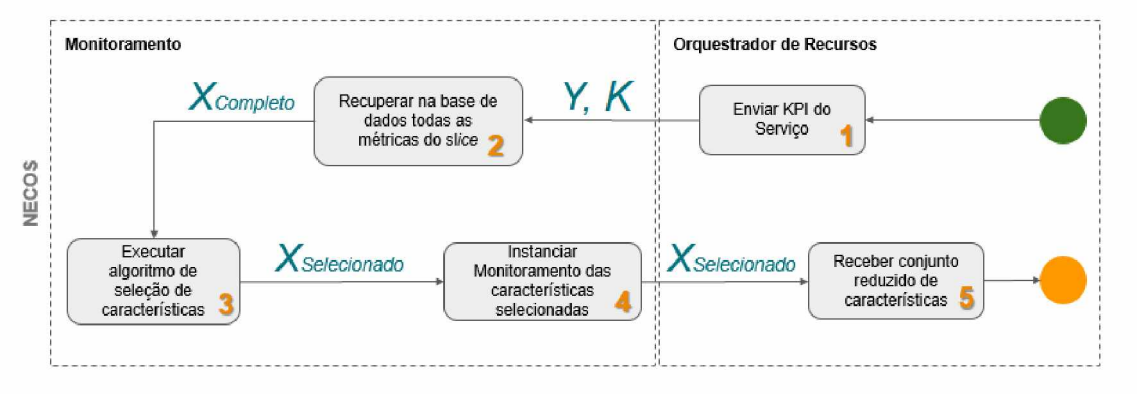

Figura 4 – Fluxo proposto do sistema inteligente de monitoramento considerando solicitações vindas do módulo Orquestrador de Recursos.

Neste exemplo, o Orquestrador de Recursos envia ao módulo de monitoramento da *slice* o conjunto Y e a quantidade K de métricas a serem selecionadas (passo 1). Por sua vez, o módulo de monitoramento recupera todas as métricas de infraestrutura relativas a *slice*, formando o conjunto $X_{Completo}$ (passo 2). Na sequência, executa-se o algoritmo de seleção de características para construção de um conjunto reduzido de métricas denominado $X_{Selecionado}$ (passo 3), instanciando o monitoramento destas características (passo 4) a serem fornecidas ao Orquestrador de Recursos (passo 5).

O fluxo ilustrado ocorre sempre que o módulo de monitoramento ainda não realizou a seleção de características, ou quando a renovação desse conjunto for requisitada. Nos demais casos, o sistema de monitoramento simplesmente retorna os dados coletados, sem necessidade de refazer a seleção. Essa renovação pode se dar por conta de troca do serviço hospedado, ou quando o módulo solicitante entenda que há a necessidade de reconstrução do conjunto reduzido. Há ainda a possibilidade de essa renovação ser necessária devido a mudanças de infraestrutura, no entanto isso é uma hipótese a ser testada.

Feita a seleção, o *SIMON* passa a monitorar as métricas selecionadas mais frequentemente que as demais, o que significa que todas as métricas continuam a ser geradas. A frequência na coleta das métricas selecionadas $X_{Selecionado}$ respeita a solicitação do *Tenant*. No entanto, a coleta do $X_{Completo}$ se dá mediante a necessidade de seleção das métricas, o que evita, no contexto do NECOS, a sobrecarga da rede de comunicação entre o subcomponente de monitoria e os VIM/WIM.

A necessidade de se manter a geração de todas as métricas da infraestrutura existe, porque é preciso permitir a renovação do conjunto reduzido. Entretanto, é importante

ressaltar que evitar a coleta constante dessas informações já permite amenizar os efeitos levantados na Seção 3.1.2.

Capítulo 4

O ambiente de testes e a implementação do *SIMON*

Este capítulo apresenta como a proposta foi desenvolvida, detalhando todos os seus componentes e a estruturação do ambiente de testes.

4.1 Ambiente de teste

Esta seção detalha o ambiente utilizado para realização dos testes do sistema construído. O serviço testado, em analogia com o serviço que é hospedado em uma *slice*, foi o de leitura e escrita em uma DHT. Para tanto, criou-se um *cluster* com o *Cassandra* na versão $v3.11.3$ Apache (2016), hospedado em 5 máquinas virtuais ofertadas pelo *OpenStack* (VEXXHOST, 2020a). Cada uma dessas máquinas recebeu um *exporter* de dados (KOCHIE, 2018) para que as métricas pudessem ser geradas e coletadas pelo *Prometheus*.

Além disso, outras duas máquinas virtuais complementam o ambiente de teste: uma na qual um cliente do serviço *Cassandra* é executado, sendo esta utilizada para a coleta de métricas Y; e outra na qual um gerador de carga foi desenvolvido para simular uma comunidade dinâmica de clientes do serviço em questão.

O conjunto de métricas Y contém estatísticas relativas a ambas as operações de leitura e escrita no *Cassandra* que foram desempenhadas pelo cliente. Dentre as métricas temos o número de operações realizadas e tempos de resposta, incluindo, por exemplo, média e percentis destes. O objetivo da utilização do gerador de carga é conduzir o serviço do *Cassandra* através de diferentes níveis de carga. Desta forma, pode-se simular diversos cenários de uso do serviço, observando desde um cenário onde apenas um cliente o utiliza, até o cenário em que haja estresse da aplicação.

Toda a infraestrutura é virtualizada no *OpenStack*. Cada uma das 5 máquinas virtuais, que compõem o *cluster* do *Cassandra*, possui a seguinte configuração: 1 *virtual Central Processing Unit* (vCPU) atribuído pelo *OpenStack*, $4GB$ de RAM e $50GB$ de disco. A máquina cliente para coleta do Y possui: 1 vCPU atribuído pelo *OpenStack*, $2GB$ de

RAM e $20GB$ de disco. A máquina do gerador de carga possui: 4 vCPUs atribuídos pelo *OpenStack*, $8GB$ de RAM e $20GB$ de disco. Todas estas máquinas virtuais rodam com sistema operacional Linux Ubuntu Server 16.04. Os mecanismos de aprendizado para seleção de características foram executadas em uma máquina com processador Intel $i7$ de sétima geração e $16GB$ de memória RAM, em um ambiente com sistema operacional Linux Ubuntu 18.04 LTS. O *Prometheus* foi utilizado como um *container* do *Kubernetes*. Isso facilita o seu manuseio, uma vez que facilita a sua configuração e instanciação.

4.1.1 Correspondência do ambiente de teste com o NECOS

Uma vez que o contexto desta dissertação está inserido no contexto do projeto NECOS, foi preciso montar um ambiente de teste que reproduzisse da maneira mais próxima possível o ambiente real do projeto.

Neste sentido, o *OpenStack* permitiu a virtualização de todo o ambiente, fazendo com que a ideia das *slices* pudesse ser representada de maneira mais fiel. Com máquinas virtuais o trabalho de gerenciamento de infraestrutura é simplificado, dado que não se precisa atuar diretamente no *hardware* para modificar os recursos da máquina, conforme as necessidades da aplicação.

O *Prometheus* fez o papel de tecnologia de monitoramento, que pode ser escolhida pelo *Tenant* no momento da criação da *slice*. Instanciá-lo como um *container* do *Kubernetes* e integrar esse *container* com o *SIMON* foi uma forma de representar a interação do subcomponente de monitoria do IMA com os VIM/WIM da arquitetura do NECOS. Além disso, o fato de o *Prometheus* ter sido utilizado em um *cluster* do *Kubernetes* facilitou a implantação da proposta, conforme será detalhado no Capítulo 5.

O *Adaptor* foi implementado como um módulo do *SIMON* e foi responsável por extrair os dados do *Prometheus*, gravando cada uma separadamente em um arquivo de extensão CSV, depois unificá-los em apenas um, garantindo que para cada segundo considerado da monitoria há medição de todas as métricas. A necessidade de se ter apenas um arquivo CSV vem da maneira pela qual o *SIMON* foi implementado: as bibliotecas do *Python* utilizadas trabalham melhor com arquivos CSV (a implementação dessa parte do sistema está detlhada nas Subseções 4.3.4 e 4.3.5).

O *SIMON* expôs uma API que permite solicitar a seleção de métricas, tendo como parâmetro obrigatório o arquivo de métricas Y, a métrica a ser considerada como métrica de QoS, e como parâmetro opcional tem-se o número K de métricas a serem selecionadas. Na ausência do K, o *SIMON* se encarrega de encontrá-lo. Assim fica representada a forma de integração dos demais módulos do NECOS com o subcomponente de monitoria contido no IMA. Como resposta dessa API, o *SIMON* oferece os dados da infraestrutura no mesmo intervalo de tempo passado no arquivo Y para que o módulo solicitante possa executar as devidas operações que necessitar.

Além disso, o *SIMON* publicou os valores das métricas selecionadas, de forma *online*, em um tópico do *Kafka* que pode ser consumido também pelos diversos módulos do sistema interessados nessas informações.

As configurações das máquinas foram ajustadas na medida em que os testes iniciais foram sendo realizados, visando permitir que se analisasse, de maneira mais simples, muitos dos estados pelos quais a infraestrutura do NECOS poderia passar.

4.2 Implantação do ambiente de teste

Esta seção detalha o funcionamento das ferramentas utilizadas na construção do ambiente de teste, também com o intuito de facilitar a replicação deste.

4.2.1 Kubernetes

Segundo Kubernetes (2018a), esta ferramenta pode ser entendida como "um sistema de código aberto para a implantação, dimensionamento e gerenciamento automático de aplicações em *containers*". Um *container* tem um funcionamento similar ao de uma VM, mas que compartilha o sistema operacional com outros *containers*. Por isso mesmo, ele é considerado mais "leve" podendo ser transportado entre servidores de nuvem de maneira simplificada. Assim como uma VM, o *container* possui seu próprio sistema de arquivos, CPU, memória (KUBERNETES, 2018b).

É possível criar um *cluster Kubernetes*, no qual vários nós são adicionados a rede, que fica isolada da rede física. O nó que inicia o *cluster* é denominado *master* e os demais são tidos como *workers*. A inserção de um nó *worker* no *cluster* se dá mediante a um comando de *join* executado no *worker*, em que um *token* e o endereço do *master* são passados para autenticação. Todo o controle do *cluster* se realiza pelo *master*.

Quando se deseja instanciar uma aplicação em um *cluster Kubernetes*, descreve-se o *container* em um arquivo de extensão YAML. Executando esse arquivo, um *pod* é criado para rodar o *container*. Um *pod* é um processo executado no *cluster* e oferta aos *containers* recursos de infraestrutura, tais como memória, rede, CPU e disco. Um *pod* pode encapsular vários *containers*, ainda que estes estejam rodando em *workers* diferentes do *cluster*. O nó *master* é o responsável por distribuir os *pods* entre os diversos *workers* da rede, dividindo a carga no *cluster*, embora seja possível deixar fixo, isto é, é possível especificar em qual *worker* o *pod* será instanciado.

No contexto desta dissertação, conforme dito anteriormente, o *Prometheus* foi configurado como um *container* do *Kubernetes*. A vantagem principal é que iniciar o processo e finalizá-lo se dá de forma bastante simples, necessitando executar apenas um comando no *master* do *cluster*.

4.2.2 OpenStack

Em (VEXXHOST, 2020b), a ferramenta *OpenStack* é definida como:

> Um sistema operacional em nuvem que controla grandes *pools* de recursos de computação, armazenamento e rede em um *datacenter*, todos gerenciados e provisionados por meio de APIs com mecanismos de autenticação comuns.
>
> Um painel também está disponível, fornecendo aos administradores controle e capacitando seus usuários a fornecer recursos por meio de uma interface da web.
>
> Além da funcionalidade padrão de infraestrutura como serviço, componentes adicionais fornecem orquestração, gerenciamento de falhas e gerenciamento de serviços, entre outros serviços, para garantir alta disponibilidade dos aplicativos do usuário.

Portanto, no contexto desta dissertação todas máquinas foram virtualizadas no *OpenStack*, exceção feita da que foi utilizada para executar a avaliação dos resultados. Isso significa que as 5 máquinas do *cluster* do *Cassandra*, as máquinas do cliente e do gerador de carga, bem como as máquinas que suportaram o *cluster* do *Kubernetes*, o *Prometheus*, *Kafka* e o prórpio *SIMON* estavam virtualizadas no *OpenStack*.

4.2.3 Cassandra

Segundo (APACHE, 2016), o *Cassandra* é um banco de dados não relacional com boa escalabilidade e alta disponibilidade, que não compromete seu desempenho. Este foi desenhado para funcionar em *cluster* e garante que cada nó é idêntico aos demais.

No contexto deste trabalho, o *Cassandra* foi utilizado para disponibilizar um serviço de leitura e escrita em uma tabela de banco de dados. Uma vez que o cenário do NECOS envolve diretamente serviços descentralizados, essa ferramenta permitiu que um contexto distribuído fosse simulado e também possibilitou, de maneira simples, a coleta de diversas métricas desse mesmo serviço: tempo de leitura e escrita, número de operações por segundo, 95, 99 e 99,99 percentis dos tempos de leitura e escrita, número de linhas escritas ou lidas por segundo, média de tempo de leitura e escrita, entre outras.

4.3 Implementação do SIMON

Esta seção apresenta as ferramentas que foram utilizadas na implementação do sistema *SIMON*, proposto no Capítulo 3.

4.3.1 *Prometheus*

Para a coleta de métricas da infraestrutura, utilizou-se um *software* denominado *Prometheus* (PROMETHEUS, 2014c). Atualmente, o *Prometheus* é um *software* de código

aberto e independente, disponível desde 2012. Segundo Prometheus (2014c), o *software* é adequado "para gravar qualquer série temporal puramente numérica. Ele se encaixa no monitoramento centrado na máquina, bem como no monitoramento de arquiteturas orientadas a serviços altamente dinâmicas". Isso, a princípio, faz com que ele seja uma boa opção, dado o cenário desta dissertação. Além disso, um outro fator foi muito importante para que se optasse pelo *Prometheus*: é possível descartar uma métrica, caso necessário, evitando o tráfego desnecessário de dados. Esta funcionalidade é essencial para o contexto deste trabalho, uma vez que podemos evitar que métricas não essenciais sejam movimentadas, desde a infraestrutura onde são geradas, até o módulo de monitoramento (IMA), para serem descartadas.

Para personalizar a ferramenta, cria-se um arquivo de configuração o qual permite descrever métricas a serem descartadas, conforme o trecho de código destacado a seguir:

```
1 metric_relabel_configs:
2   - source_labels: [ __name__ ]
3     regex: 'a_metric_to_drop'
4     action: drop
```

Nesse caso, a métrica, cujo *label "__name__"* seja compatível com a *regex* informada, será descartada. Uma *regex* para ser válida deve seguir o padrão RE2 (GOOGLE, 2019). A *tag "action"* também pode assumir o valor *"keep"*, em que apenas as métricas, cujos valores das *labels* determinadas na *tag source_labels* forem compatíveis com a *regex* informada, serão mantidas, descartando-se as demais.

Esse descarte não impede a métrica de ser gerada, mas fará com que ela não seja coletada. A razão para que ela não deixe de ser gerada se deve ao fato de que não é o *Prometheus* o responsável pela geração, mas, apenas, pela coleta. Para gerar as métricas, foi instalado um *exporter* de dados para *Prometheus* nas máquinas onde o serviço testado foi hospedado.

As métricas coletadas pelo *Prometheus* podem ser armazenadas localmente, em disco, ou pode-se configurar um ambiente remoto para isso. Esse ambiente remoto pode ser um banco de dados, entre eles um *PostgreSQL* Group (1996) ou *InfluxDB* InfluxData (2017), para os quais a escrita e leitura pode ser feita diretamente pelo software, ou um sistema de barramento como *Kafka* Apache (2017), mediante a utilização de um adaptador *Prometheus-Kafka* (QIAN, 2018). A lista completa de sistemas que podem ser integrados diretamente com o *Prometheus* está disponível em Prometheus (2014a), e a de lista de *exporters* que podem ser usados para mediar a integração do software com outros sistemas está disponível em (PROMETHEUS, 2014b).

Além disso, o arquivo de configuração permite que se determine um tempo de coleta, seja para todas as métrica ou para um conjunto de métricas específico, denominado de *job*.

O trecho a seguir demonstra como o arquivo de configuração pode determinar um tempo de coleta global, isto é, um tempo de coleta para todas as métricas. No exemplo dado, o tempo de coleta foi definido para 1 *segundo*.

```
global:
  scrape_interval: 1s
```

Ou ainda pode-se definir um intervalo de coleta específico para um conjunto de métricas de um mesmo *job*. O trecho a seguir demonstra como determinar esse intervalo para um *job* de nome *node*. O leitor poderá encontrar mais informações sobre as propriedades de configuração e como utilizá-las em[1].

```
scrape_configs:
  - job_name: node
    scrape_interval: 1s
```

4.3.1.1 Geração de métricas de infraestrutura

Para que o *Prometheus* colete as métricas, é necessário ter, na infraestrutura, um serviço que gere e exporte esses dados em um formato preestabelecido. A rota para esse serviço é especificada no arquivo de configuração do *Prometheus*.

Na implementação do ambiente de testes, detalhado na próxima seção, foi utilizado um *exporter* que pode ser encontrado em (KOCHIE, 2018). Uma vez instalado na máquina, o *exporter* vira um serviço que expõe uma API de consulta, permitindo a coleta dos dados.

Dentre as métricas geradas pelo serviço do *exporter*, muitas estão relacionadas à rede, tais como: *"node_sockstat_TCP_alloc"*, que diz quantos *sockets Transmission Control Protocol* (TCP) estão alocados atualmente; *"node_sockstat_TCP_inuse"*, que representa quantos *sockets* TCP estão em uso atualmente. Muitas outras relacionadas a uso de CPU, a memória RAM, a armazenamento, escrita e leitura em disco também são coletadas. O número total de métricas pode variar de acordo com a infraestrutura. No contexto deste trabalho, o número total de métricas foi de 706. As métricas geradas são listadas quando se consulta o *endpoint* "*/metrics*", também exposto pelo serviço. O resultado é uma lista com todas as métricas disponíveis, bem como uma breve explicação do que se trata cada uma.

4.3.2 Geração de métricas do serviço

Conforme descrito no Capítulo 3, uma das formas de se conseguir o conjunto Y para o aprendizado de máquina é instanciar um cliente do serviço para que se colete as métricas

[1] https://prometheus.io/docs/prometheus/latest/configuration/configuration/

desejadas. Desta forma, no cenário desta dissertação, criou-se um cliente para o serviço da DHT executado em uma máquina virtual do *OpenStack*. E, para que se aproxime de uma situação real de uso, um gerador de carga, implementado utilizando a linguagem Python, também foi executado em outra *Virtual Machine* (VM). Esse gerador aplicou uma carga segundo dois padrões.

Nos primeiros testes, o padrão segue uma distribuição de *Poisson* para a chegada de novos clientes, aplicando uma sinusoide para determinar λ, que neste caso representa o número de clientes ativos. Iniciou-se o teste com $\lambda = 6$ clientes ativos (5 criados pelo gerador de carga, mais o cliente para coleta de Y). A sinusoide possui período de sessenta minutos e amplitude de 5 clientes, ou seja, o número de clientes ativos durante o tempo de uma hora oscila entre 11 e 2 clientes, sendo que cada um deles escreve e lê dados na DHT.

Ao utilizar esse padrão buscou-se testar o serviço nos mais diversos estados de uso possíveis, isto é, deixando-o ocioso em alguns momentos dado o baixo número de clientes, e elevando esse número até o máximo suportado pela infraestrutura. Importante ressaltar que esse número máximo de clientes, que causou o estresse da infraestrutura, foi medido em testes preliminares, não detalhados aqui por estarem fora da proposta deste trabalho. Portanto, com esse padrão de carga foi possível coletar métricas em vários cenários de uso diferentes, o que pode favorecer o aprendizado de máquina, uma vez que assim a infraestrutura suportou várias formas de carga. A Figura 5 ilustra o padrão de criação dos clientes no gerador de carga. O período é de uma hora.

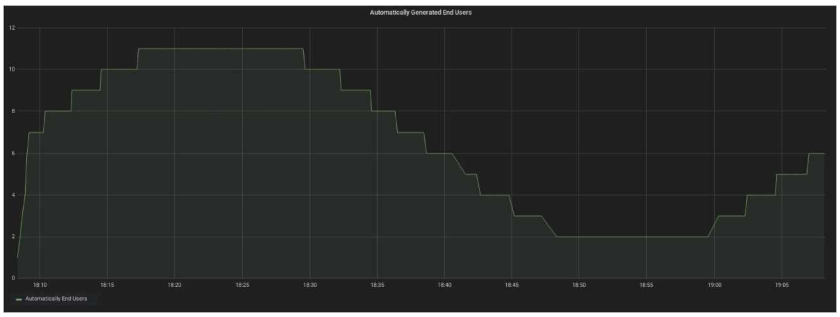

Figura 5 – Sinusoide que expressa a criação de clientes para o serviço da DHT.

Nos testes posteriores, aplicou-se um padrão diferente, aqui denominado Padrão 2, que instancia clientes com um intervalo regular de tempo até se atingir o número máximo, mantendo essa quantidade máxima por um tempo maior que na sinusoide. De forma análoga funciona a retirada de clientes, que ocorreu com intervalos regulares, até se atingir o número mínimo, que foi mantido por um tempo maior. O tempo total para todas essas operações foi de 20 minutos, dos quais 5 para instanciar todos os clientes, 5 foram mantendo a carga máxima ativa, 5 para pará-los e 5 mantendo o número mínimo.

O intuito desse padrão de carga é bastante similar ao primeiro padrão: estressar a infraestrutura e deixá-la ociosa, transitando em todos os cenários suportados. A redução

do período de 1 hora para 20 minutos foi com o objetivo de provocar mais alterações de uso do serviço, permitindo que um mesmo padrão de carga fosse observado com maior frequência em determinado intervalo de tempo. Isso permitiu que mais métricas referentes a um mesmo estado da infraestrutura fossem coletadas ao longo do teste. Ao invés de o cenário se repetir a cada 1 hora, ele se repetiu a cada 20 minutos o que também pode favorecer o aprendizado de máquina.

A Figura 6 ilustra o padrão aplicado aos demais testes, no qual a criação / parada de clientes é determinado por tempo, e não por distribuição de *Poisson*. Uma outra diferença é que este padrão foi aplicado com um período de 20 minutos. A partir do cliente que roda em uma VM separada, coleta-se as métricas do serviço que, como já mencionado, podem ser tempos de leitura, escrita, número de operações por segundo, média ou percentis dos tempos de leitura ou escrita, entre outras.

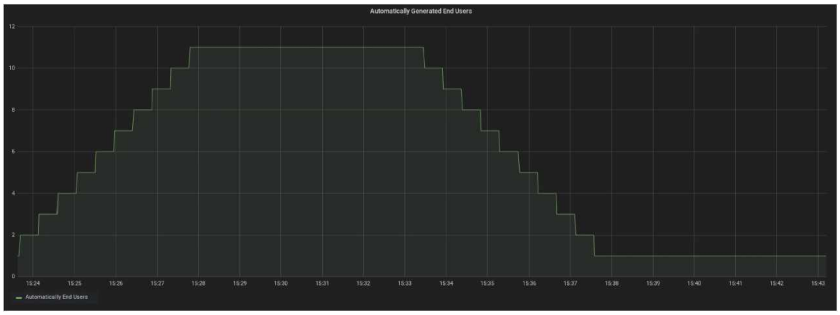

Figura 6 – Padrão de criação determinado pelo autor do projeto.

4.3.3 *Scikit Learn*

A implementação de todo o sistema foi feita utilizando a linguagem *Python* nas versões 3.5 e 2.7 (FOUNDATION, 2020). Fator fundamental para a escolha da linguagem foi o pacote Scikit Learn (INRIA, 2018). Este pacote oferece, de forma prática, a implementação de diversos algoritmos de análise de dados, de aprendizagem de máquina, de regressão, classificação, o que aumenta significativamente a produtividade de trabalhos como este. Dentre os diversos algoritmos implementados, dois foram utilizados na implementação da proposta, visando a possibilidade de comparação entre os diferentes resultados obtidos. Ambos estão descritos e explicados nas duas subseções seguintes.

O uso dos algoritmos descritos a seguir permitiu duas análises para a avaliação dos resultados: 1) essa abordagem permitiu amenizar as dificuldades da solução inicial proposta no contexto do NECOS, descritas no Capítulo 3; 2) usar o algoritmo de correlação e o do *Random Forest* significa executar um algoritmo que não possui aprendizado de máquina com um que possui. Isso pode permitir que se tire algumas conclusões sobre o uso de inteligência artificial na seleção de características no que diz respeito, por exemplo, à eficiência do aprendizado de máquina para o objetivo traçado.

4.3.4 O algoritmo de correlação

O primeiro algoritmo testado foi o de correlação encontrado no método *SelectKBest* (INRIA, 2019c). Neste caso, o algoritmo leva em consideração a correlação existente entre as métricas em X com a métrica em Y e cria um *ranking*, no qual a primeira é a métrica de X mais correlacionada com a métrica em Y e a última é a menos correlacionada. Os parâmetros passados para o método mencionado são: um classificador - que neste trabalho foi utilizado um classificador padrão denominado *f_classif* -, e o número K de métricas a serem selecionadas. Com o objeto retornado pelo método, é possível recuperar os índices das métricas selecionadas. O trecho de código abaixo demonstra como executar o método e recuperar os índices das métricas selecionadas.

```
1 import pandas as pd
2
3 fields = ['timestamp', yField]
4 xRaw = pd.read_csv(pathToXFile)
5 yRaw = pd.read_csv(pathToYFile, skipinitialspace=True, usecols=fields)
6 selector = SelectKBest(f_classif, k=int(K))
7 _ = selector.fit_transform(xRaw, yRaw[yField])
8 idxSelected = selector.get_support(indices=True)
```

Código 4.1 – Código em Python para executar a correlação

Aqui, *xRaw* representa um arquivo de extensão CSV contendo todas as métricas coletadas da infraestrutura. O cabeçalho, contido na primeira linha do arquivo, é organizado de forma que a primeira coluna representa o *timestamp* em que a métrica foi coletada, e as demais colunas representam o nome das métricas. As demais linhas do arquivo são preenchidas com o valor correspondente ou do *timestamp* - no caso da primeira coluna - ou da métrica em si. De forma análoga, pode-se entender a variável *yRaw*: a primeira coluna do arquivo é o *timestamp* em que a métrica do serviço foi coletada, e a segunda possui o valor da métrica propriamente dito.

O método *SelectKBest* constrói um *selector* que posteriormente é aplicado às métricas X e Y (KPI).

A variável *yField* utilizada no momento do *fit_transform* determina o nome da segunda coluna contida no arquivo Y. Desta forma, *yRaw[yField]* elimina o *timestamp*, mantendo apenas o que interessa: a métrica do serviço em questão.

Como resultado final tem-se o vetor *idxSelected*, que contém os índices das métricas selecionadas. O arquivo contendo as métricas X, como dito, é um CSV em que cada coluna representa uma métrica. Os índices retornados são construídos com base nessa ordem determinada pelas colunas no arquivo. A métrica da primeira coluna é o índice 0 e a última possui o índice $N-1$, considerando N métricas. Portanto, os índices retornados

são as primeiras K posições do *ranking* criado pelo algoritmo, o que caracteriza a técnica *filter* de seleção de características.

Este não é exatamente um algoritmo de inteligência artificial, razão pela qual foi utilizado. Dada a simplicidade de seu uso e de já estar implementado no pacote *scikit-learn*, optou-se por utilizá-lo com intuito também de comparar abordagens com e sem aprendizado de máquina. Os resultados obtidos com esse algoritmo serão melhor detalhados no Capítulo 5.

4.3.5 O algoritmo SFS

O segundo algoritmo testado foi o SFS em conjunto com o algoritmo *Random Forest* Breiman (2001) e possui uma complexidade muito superior ao de correlação. O pacote *scikit* também traz o *Random Forest* implementado, tanto para regressão, quanto para classificação (INRIA, 2019b) (INRIA, 2019a). Para este trabalho, utilizou-se o regressor, dado que a métrica a ser estimada é composta de valores reais que dizem respeito a qualidade do serviço hospedado. O trecho de código abaixo demonstra a construção e utilização de um regressor baseado no algoritmo *Random Forest*:

```
1  from sklearn.ensemble import RandomForestRegressor
2  from sklearn.model_selection import train_test_split
3
4  yField='W_95'
5
6  fields = ['timestamp', yField]
7  x_raw = pd.read_csv(pathToXFile)
8  y_raw = pd.read_csv(pathToYFile, skipinitialspace=True, usecols=fields)
9
10 X_train, X_test, y_train, y_test = train_test_split(x_raw, \
11              y_raw[yField], test_size=0.3, random_state=11)
12
13 regr = RandomForestRegressor(max_depth = 10, n_estimators = 5, bootstrap
       = False)
14 regr.fit(X_train, y_train)
15 y_predicted = regr.predict(X_test)
```

Código 4.2 – Código em Python para executar a estimativa com base no Random Forest.

Nesse caso, o regressor foi utilizado diretamente para realizar a estimativa dos dados $\hat{Y}$. Diz-se que o algoritmo estima a KPI do serviço. De forma análoga ao que ocorreu no uso do algoritmo de correlação, *xRaw* e *yRaw* representam os arquivos de extensão CSV que contém as métricas X e Y, respectivamente. A variável *yField* indica o nome da métrica do arquivo Y que será considerada como a métrica de QoS, ou KPI, para ser estimada. No código, divide-se as métricas X e Y em duas partes: uma para treinamento

do modelo, isto é, para a construção da função F de regressão, e outra para o teste e cálculo de erro. A porcentagem de divisão é estipulada pelo parâmetro *test_size*, que neste caso, indica que o teste contém 30% dos dados coletados, ficando os outros 70% para o treinamento.

Cria-se, então, o regressor com a chamada do método *RandomForestRegressor*, cujos parâmetros são assim definidos:

- O parâmetro *max_depth* indica a profundidade máxima de cada árvore utilizada no algoritmo;
- O parâmetro *n_estimators* indica o número de árvores a serem construídas no algoritmo.
- O parâmetro *bootstrap* quando colocado com valor *False* indica que todo o conjunto de dados será usado para a construção de cada árvore.

Esta dissertação não detalhará o algoritmo, mas o leitor poderá ter mais informações em Breiman (2001).

Desta forma, implementou-se o *Forward Stepwise Feature Selection* usando o *Random Forest* da seguinte forma:

```
X_train_full, X_test, y_train_full, y_test = train_test_split(X_raw,
    y_raw['W_95'], test_size=0.3, random_state=11)
for t in ts:
    selectedMetrics = []
    bestMetrics = []
    possibleMetrics = [x for x in list(range(1, len(list(X_train))))] #
   gera indices das metricas no vetor list(X_train)
    lenPossibleMetrics = len(possibleMetrics)
    minNMAEGlobal = 10000.00
    yTestMean = y_test.mean()
    regr = RandomForestRegressor(max_depth=10, n_estimators=5, bootstrap
   =False)
    count = 0
    stop = False
    while(not stop):
        actualBestMetric = -1
        minNMAELocal = 10000.00
        for k in possibleMetrics:
            indexes = selectedMetrics + [k] #inclui a metrica de indice
   'k' no conjunto de metricas a serem consideradas
            X_train_new = X_train.iloc[:, indexes] #filtra o conjunto de
    treino considerando apenas os indices selecionados
            X_test_new = X_test.iloc[:, indexes] #filtra o conjunto de
   teste considerando apenas os indices selecionados
```

```
19            regr.fit(X_train_new, y_train) #monta o modelo de regressao
20            pred = regr.predict(X_test_new) #executa estimativa com base
     no modelo criado
21            nmae_Selected = (abs(pred - y_test).mean())/yTestMean #
    calcula o NMAE para o conjunto selecionado
22
23            if nmae_Selected < minNMAELocal: #se houver melhora no NMAE
    calculado para essa rodada faz as devidas substituicoes
24                minNMAELocal = nmae_Selected
25                actualBestMetric = k
26
27        selectedMetrics = selectedMetrics + [actualBestMetric]
28
29        if minNMAELocal - minNMAEGlobal >= 0.000001: # se o minimo
    encontrado for pelo menos 0.000001 menor
30            count = count + 1  # que o minimo tido ate o momento, faz a
    substituicao
31            if count == 10:
32                stop = True # se 10 metricas testadas aumentam o menor
    erro obtido, interrompe-se o algoritmo
33        else: # do contrario, faz as devidas substituicoes e zera o
    contador
34            minNMAEGlobal = minNMAELocal
35            bestMetrics = selectedMetrics
36            count = 0
37
38        possibleMetrics.pop(actualBestMetric) #retira a metrica
    selecionada do conjunto de metricas ainda disponiveis
39
40        lenPossibleMetrics = lenPossibleMetrics - 1
41
42        if lenPossibleMetrics <= 0:
43            stop = True # se nao houver mais metricas a serem testadas,
    interrompe-se o algoritmo
```

Código 4.3 – Código em Python para executar a seleção com base no Forward Stepwise Feature Selection.

Ao final da execução deste algoritmo, tem-se em *bestMetrics* os índices das métricas que devem compor o conjunto selecionado.

No caso do uso do algoritmo SFS a técnica de seleção de características utilizada foi a *wrapper*, o que significa que o algoritmo usado para selecionar as métricas também foi utilizado para a estimativa da métrica Y, uma vez que é justamente com base no cálculo da estimativa que se seleciona a melhor métrica.

Capítulo 5

Experimentos e Análise dos Resultados

Este capítulo detalha a fórmula de cálculo de erro, os experimentos realizados e apresenta a avaliação dos resultados, demonstrando a validação das Hipóteses.

5.1 Método para a Avaliação

Para a avaliação dos resultados, calculou-se o Erro Absoluto Médio Normalizado (*Normalized Mean Absolute Error - (NMAE)*), que é expressado por:

$$NMAE_m = \frac{1}{\overline{y}}(\frac{1}{m}\sum_{i=1}^{m}|y_i - \hat{y}_i|),$$

onde $\hat{y}_i$ é o valor estimado da métrica obtido pelo regressor no i-ésimo segundo, y_i é o valor real coletado no cliente no i-ésimo segundo, e $\overline{y}$ é a média dos valores y_i do conjunto de teste, cujo tamanho é igual a m.

A partir desse cálculo, foi possível comparar os resultados obtidos nos diversos cenários de teste executados, confrontando as técnicas *filter* e *wrapper*, quando aplicadas com o algoritmo de correlação e um algoritmo de aprendizagem supervisionada, respectivamente. Esse confronto de técnicas e algoritmos permitiu que se definisse uma sugestão de abordagem para o contexto do NECOS, que é um dos objetivos deste trabalho.

Um outro parâmetro de avaliação foi o volume de dados trafegados quando se coleta o conjunto completo de métricas de infraestrutura e quando se coleta o conjunto selecionado. Quanto maior a redução, sem perda de precisão na estimativa das métricas, melhor.

Desta forma, as hipóteses levantadas no Capítulo 1 puderam ser validadas, pois os experimentos responderam as perguntas feitas.

5.2 Experimentos

Essa seção apresenta os resultados obtidos nos testes, em que foi possível validar as hipóteses levantadas no Capítulo 1.

5.2.1 Cenários de teste

Três cenários de teste foram escolhidos para aplicar os algoritmos de correlação e o SFS. O primeiro, denominado Cenário 1, é caracterizado por ter o serviço da DHT sendo executado em uma infraestrutura sem restrição na placa de rede. Isso significa que a placa de rede operou na sua capacidade máxima, que neste caso é de 1 Gbit/s.

No segundo, denominado Cenário 2, foi aplicada uma restrição de $30Mbit/s$ na placa de rede. Essa restrição foi resultado de testes preliminares que demonstraram que próximo a $30Mbit/s$ o serviço começava a sofrer algum tipo de degradação na qualidade ofertada, ou seja, com restrição próximo a $30Mbit/s$ a DHT começava a aumentar os tempos de leitura e escrita.

E, por fim, no terceiro, denominado Cenário 3, foi aplicada uma restrição de $10Mbit/s$ na placa de rede. O intuito foi degradar ainda mais o serviço, fazendo com que ele operasse com maior estresse, o que significa que os tempos de resposta das operações de leitura e escrita ficaram notoriamente maiores.

Com isso, pretendeu-se demonstrar que, não obstante a infraestrutura estivesse diferente, as métricas selecionadas no Cenário 1 ainda poderiam ser utilizadas sem perda de precisão nos demais Cenários. No entanto, também foram testados mais duas possibilidades de uso de métricas:

- usar o conjunto selecionado no Cenário 2 para estimar as métricas do Cenário 1 e 3;
- usar o conjunto selecionado no Cenário 3 para estimar as métricas do Cenário 1 e 2.

Desta forma, foi possível observar o comportamento do *SIMON* quando aplicado a três situações que podem dar-se em um contexto real. Será verdade que, tendo necessidade de redução de recursos, as métricas selecionadas em uma infraestrutura mais robusta continua valendo para uma infraestrutura com menos recursos? E, o contrário? As métricas obtidas em uma infraestrutura mais simples continuam sendo boas para uma infraestrutura mais complexa?

Evidentemente muitas são as formas de restringir a infraestrutura utilizada nos testes. Poderia-se restringir memória RAM, limitar o uso de CPU's, forçar uma leitura e escrita mais lenta no disco, enfim, são variadas as restrições que se podem aplicar nas máquinas do *cluster* para observar uma degradação no serviço. Há, inclusive, a possibilidade de combinar várias dessas degradações, o que aproximaria ainda mais os testes de um cenário real. Todavia, aqui não foram feitas combinações de degradações, pois buscou-se dar início a uma investigação sobre a flexibilidade dos algoritmos ante uma mudança da infraestrutura, razão pela qual impor outras restrições, bem como combinações de restrições de infraestrutura são algumas das ideias de trabalhos futuros propostas no Capítulo 6.

Em todos os Cenários, foram testados quatro intervalos de coleta de métricas da infraestrutura: intervalo de 1 segundo, 5 segundos, 10 segundos e 30 segundos. O intuito foi testar se os algoritmos conseguem dar um bom conjunto de métricas ainda que as informações sejam mais escassas. Nesse caso, quanto maior o intervalo de coleta das métricas, consequentemente menor o volume de dados trafegados e analisados. Se mesmo assim os algoritmos forem capazes de gerar um bom conjunto de métricas, então significa que a proposta consegue ajudar ainda mais na redução dos dados trafegados entre o IMA e os VIM/WIM das *slices*.

5.2.2 Métrica analisada da qualidade do serviço

Dentre as várias métricas que poderiam ser analisadas, o 95 percentil dos tempos de escrita foi considerado nos testes. Essa métrica indica que 95% das operações de escrita realizadas naquele segundo tiveram tempo menor ou igual ao seu valor. A escolha do 95 percentil se deu em razão de representar um dos piores tempos de resposta da DHT, mas não o pior, e, portanto, buscar métricas que estejam relacionadas a um dos piores estados do serviço é parte do objetivo de manter a qualidade da aplicação. Poderia-se, por exemplo, buscar estimar a média aritmética dos tempos de escrita, mas o fato de ela estar dentro de um limite aceitável não significa que o serviço esteja bom, motivo pelo qual não foi considerada aqui.

Havia também a possibilidade de usar a métrica 99 percentil e mesmo a 99.99 percentil, mas são métricas muito sensíveis e que podem degradar mesmo com picos esporádicos. Desta forma, usar uma destas duas métricas poderia fazer com que os algoritmos buscassem estimar e se adaptar a casos isolados, o que pode não ser bom em um contexto geral. Por conta disso, optou-se pela métrica de 95 percentil.

Com relação a ser o tempo de escrita e não de leitura, não há razões especiais. O serviço oferece métricas de tempos de leitura, de forma análoga às métricas de escrita, mas optou-se pelo tempo de escrita.

A Figura 7 apresenta o 95 percentil dos tempos de escrita na DHT observado no cliente onde a KPI é coletada, durante um experimento de duas horas aproximadamente e sem restrição de rede. É possível notar as oscilações causadas no tempo de resposta, reflexo do número de clientes ativos que foram controlados pelo gerador de carga, tendo aplicado o Padrão 2 descrito anteriormente.

Nota-se que os tempos de resposta ficaram, em sua maioria, contidos no intervalo de tempo de $0ms$ a aproximadamente $90ms$. Nos instantes de maior carga, refletido no aumento do tempo de resposta, é possível perceber que a métrica se manteve, em sua maioria, entre 40 e $90ms$. Alguns picos, como o tempo de $140ms$ no começo são identificados, mas não são frequentes.

O eixo das abscissas representa o tempo em segundos do teste. Como foram duas horas de teste, há 7200 segundos ilustrados no gráfico.

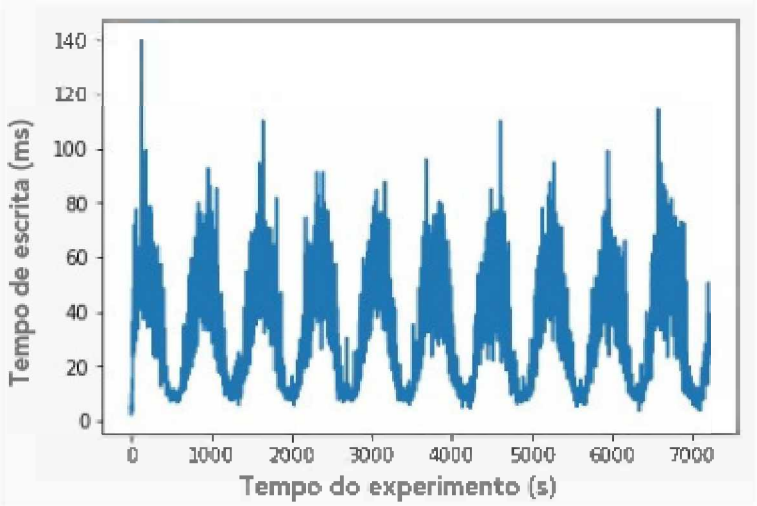

Figura 7 – 95 percentil do tempo de escrita na DHT ao longo de duas horas de experimentação.

Nos Cenários 2 e 3, optou-se por fazer experimentos com uma hora a mais de duração. Isso se deve ao fato de que buscou-se também testar se uma maior quantidade de informações permite melhor análise. Uma vez que os algoritmos se utilizam das informações passadas para identificar quais as métricas mais relacionadas com a qualidade do serviço, não será verdade que quanto mais informação passada, melhor a precisão da estimativa? É uma possibilidade a ser investigada aqui.

A Figura 8 apresenta o 95 percentil dos tempos de escrita na DHT observado no cliente onde a KPI é coletada, durante um experimento de três horas aproximadamente e com restrição de $30Mbit/s$ na placa rede. Também, foi aplicado o mesmo padrão de carga denominado aqui de Padrão 2.

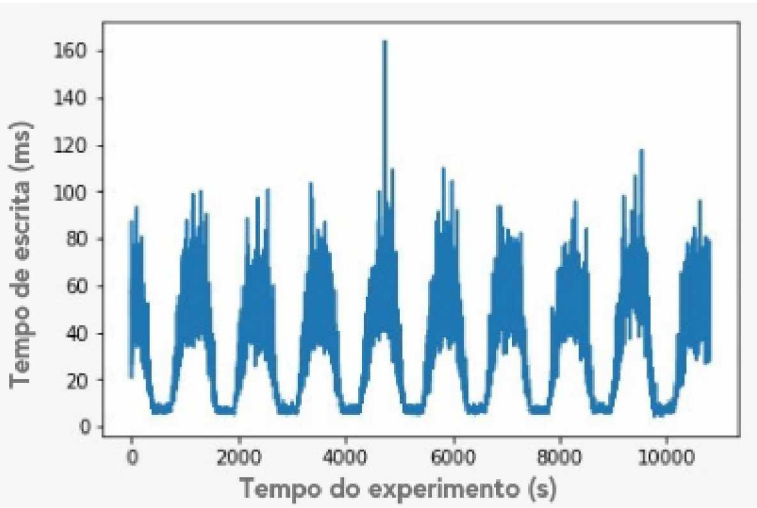

Figura 8 – 95 percentil do tempo de escrita na DHT ao longo de três horas de experimentação e restrição de rede em $30Mbit/s$.

Com a restrição de $30Mbit/s$ na placa de rede, nota-se uma leve degradação do serviço,

mas nada muito expressivo. Os tempos de resposta nos períodos de maior carga na aplicação alcançam os $100ms$ mais frequentemente que antes e houve também um pico, mas aqui de $160ms$ contra $140ms$ no Cenário 1. Também, é importante notar que esse experimento teve 3 horas de duração, o que explica o eixo das abscissas ter 10800 segundos.

A Figura 9 apresenta o 95 percentil dos tempos de escrita na DHT observado no cliente onde a KPI é coletada, durante um experimento de três horas aproximadamente e com restrição de $10Mbit/s$ na placa rede. Nesse caso, também foi aplicado o mesmo padrão de carga dos Cenários anteriores.

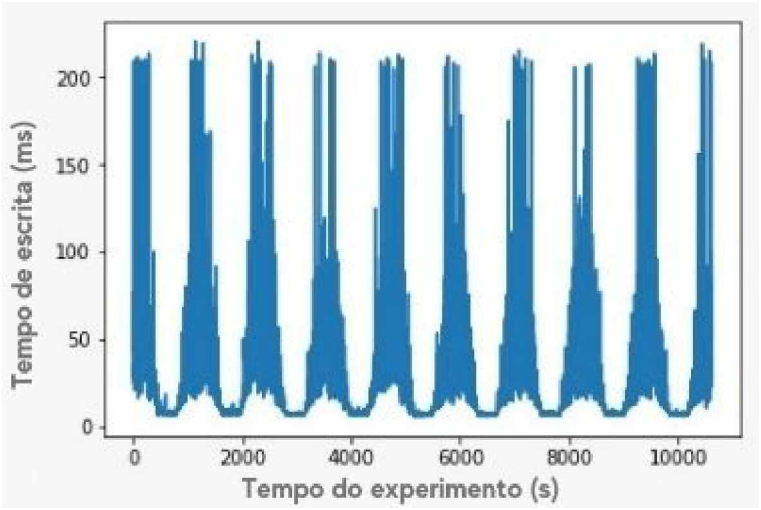

Figura 9 – 95 percentil do tempo de escrita na DHT ao longo de três horas de experimentação e restrição de rede em 10Mbit.

Nesse caso a degradação do serviço fica evidente. Os tempos de resposta alcançam facilmente os $200ms$ e nota-se que esse tempo é bem frequente quando a carga de clientes está no máximo permitido pelo Padrão aplicado. Percebe-se também que os tempos menores ocorrem em períodos mais curtos que nos Cenários anteriores, demonstrando que o serviço sofreu grandes variações para atender à carga aplicada tendo a restrição de rede imposta.

Nos três casos ilustrados, o intervalo de coleta foi de 1 segundo. Entretanto, foi feita uma filtragem nessas informações, visando testar os algoritmos nos quatro intervalos de coleta. Isso significa que, embora estivessem disponíveis os dados de 1 em 1 segundo, eles foram filtrados resultando em dados de 5 em 5, 10 em 10 e 30 em 30 segundos. Desta forma, os quatro intervalos puderam ser analisados.

5.2.3 O algoritmo de correlação

O primeiro algoritmo testado foi o de correlação que busca elencar as métricas que se comportam de maneira similar à métrica em Y. A partir dos dados coletados do serviço,

bem como das métricas da infraestrutura, o método *SelectKBest* foi utilizado para a seleção de características.

Dentro de cada Cenário, foram testados quatro intervalos de coleta, conforme dito anteriormente. Para cada intervalo de coleta - $1s$, $5s$, $10s$ e $30s$-, criou-se todos os *rankings* possíveis com as métricas coletadas, isto é, para cada intervalo de coleta testou-se a estimativa desde 1 métrica até as 706 possíveis. Então, montou-se um gráfico ilustrando a precisão com 1 métrica, com 706 métricas e com uma quantidade K que foi a quantidade com a melhor precisão dentre todas possíveis para aquele intervalo.

A estimativa foi feita com o mesmo regressor usado no algoritmo SFS (explicado na Subseção 5.2.4). Feita a estimativa, calculou-se o $NMAE$. O algoritmo de aprendizado de máquina utilizado para a regressão foi o *Random Forest*, utilizando o método *RandomForestRegressor* explicado anteriormente. Os parâmetros de configuração foram: $max\_depth = 10$, $n\_estimators = 5$ e $bootstrap =$ False. Nesta avaliação, utilizou-se 70% dos dados coletados para o treinamento do regressor e os outros 30% para avaliar a acurácia deste.

A Figura 10 ilustra os resultados obtidos quando utilizado o algoritmo de correlação no Cenário 1. Nota-se que para cada intervalo de coleta há 3 pontos no gráfico: o primeiro é referente ao $NMAE$ com a métrica que ficou no topo do *ranking*; o segundo é o melhor $NMAE$ observado dentre todos os 706 calculados, o que também define o melhor K para esse algoritmo nesse Cenário; e, por fim, o terceiro ponto mostra o erro da estimativa quando tendo todas as 706 métricas envolvidas.

Com o intervalo de $1s$ de coleta, o melhor $NMAE$ se deu com $K = 23$ e $NMAE = 18.91\%$. Para o intervalo de $5s$, o melhor $NMAE$ se deu com $K = 39$ e $NMAE = 22.38\%$. No intervalo de $10s$, o melhor $NMAE$ se deu com $K = 650$ e $NMAE = 23.40\%$. E, por fim, com o intervalo de $30s$, o melhor $NMAE$ se deu com $K = 268$ e $NMAE = 25.67\%$.

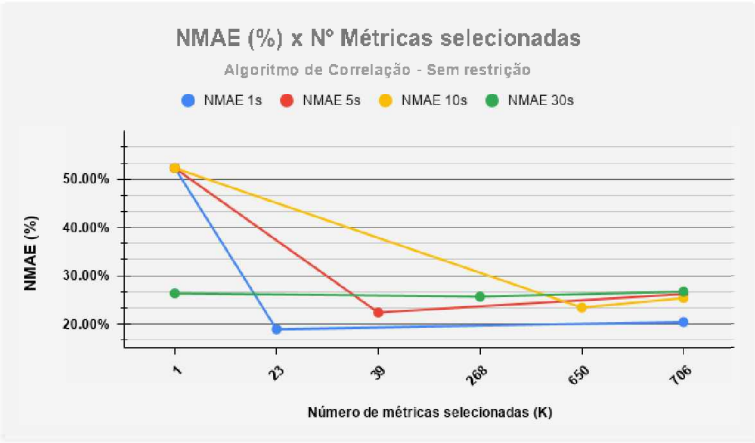

Figura 10 – $NMAE$ observada para os diferentes valores de K utilizados para seleção de métricas que compõem o conjunto $X_{Selecionado}$, com diferentes intervalos de coleta no Cenário 1.

A Figura 11 ilustra os resultados obtidos quando utilizado o algoritmo de correlação no Cenário 2. Onde se nota também que para cada intervalo de coleta há 3 pontos no gráfico de forma análoga aos três pontos do gráfico anterior.

Com o intervalo de $1s$ de coleta, o melhor $NMAE$ se deu com $K = 19$ e $NMAE = 15.83\%$. Para o intervalo de $5s$, o melhor $NMAE$ se deu com $K = 4$ e $NMAE = 16.62\%$. No intervalo de $10s$, o melhor $NMAE$ se deu com $K = 94$ e $NMAE = 20.50\%$. E, por fim, com o intervalo de $30s$, o melhor $NMAE$ se deu com $K = 546$ e $NMAE = 21.98\%$.

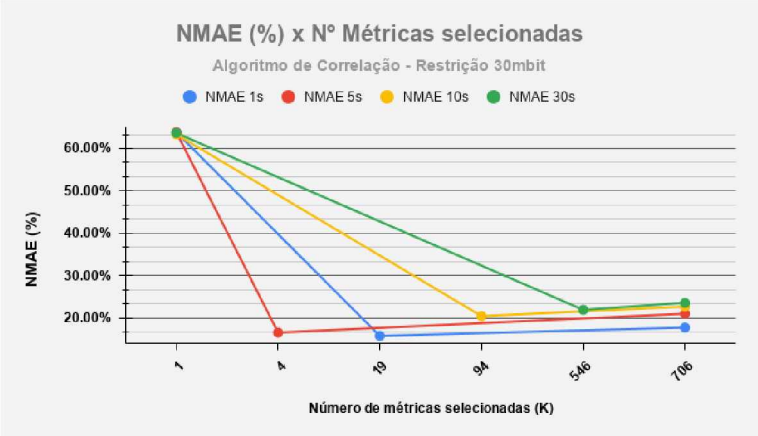

Figura 11 – $NMAE$ observada para os diferentes valores de K utilizados para seleção de métricas que compõem o conjunto $X_{Selecionado}$, com diferentes intervalos de coleta no Cenário 2.

A Figura 12 ilustra os resultados obtidos quando utilizado o algoritmo de correlação no Cenário 3. Aqui também, nota-se que para cada intervalo de coleta há 3 pontos no gráfico que funcionam de forma análoga aos gráficos anteriores.

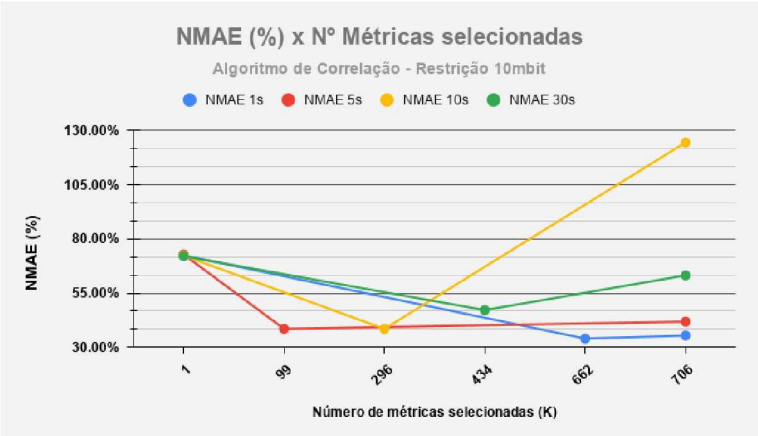

Figura 12 – $NMAE$ observada para os diferentes valores de K utilizados para seleção de métricas que compõem o conjunto $X_{Selecionado}$, com diferentes intervalos de coleta no Cenário 3.

Com o intervalo de $1s$ de coleta, o melhor $NMAE$ se deu com $K = 662$ e $NMAE = 34.04\%$. Para o intervalo de $5s$, o melhor $NMAE$ se deu com $K = 99$ e $NMAE = 38.54\%$.

No intervalo de $10s$, o melhor $NMAE$ se deu com $K = 296$ e $NMAE = 38.50\%$. E, por fim, com o intervalo de $30s$, o melhor $NMAE$ se deu com $K = 434$ e $NMAE = 47.17\%$.

5.2.3.1 As métricas do algoritmo de correlação

Por terem sido selecionadas muitas métricas em todos os Cenários, optou-se por colocar aqui apenas as duas primeiras métricas selecionadas de cada Cenário quando o intervalo de coleta foi de $1s$. No entanto, o leitor poderá encontrar a relação de todas as métricas selecionadas, nos três Cenários e para os quatro intervalos de coleta, no GitHub do autor desta dissertação[1].

A Figura 13 ilustra os gráficos das duas métricas mais correlacionadas à métrica de tempo de resposta do Cenário 1, isto é, do Cenário em que não houve restrição na placa de rede. De fato, nota-se a grande similaridade entre as duas métricas apresentadas com o gráfico apresentado na Figura 7.

As métricas mais correlacionadas ao tempo de resposta são, respectivamente, a métrica *node_scrape_collector_duration_seconds*, que indica o tempo em segundos gasto para a coleta de métricas naquele nó específico do *cluster*, e a *node_sockstat_TCP_alloc*, que calcula o número de *sockets* TCP abertos naquele segundo em determinado nó.

É importante ressaltar que os cinco nós do *cluster* geram as mesmas métricas, uma vez que possuem o mesmo *exporter* de dados, e por isso é importante diferenciar de onde veio qual métrica. Neste caso em especial ambas as métricas vieram do nó 1 do *cluster*, que também é o nó que inicia toda a instanciação do *cluster*.

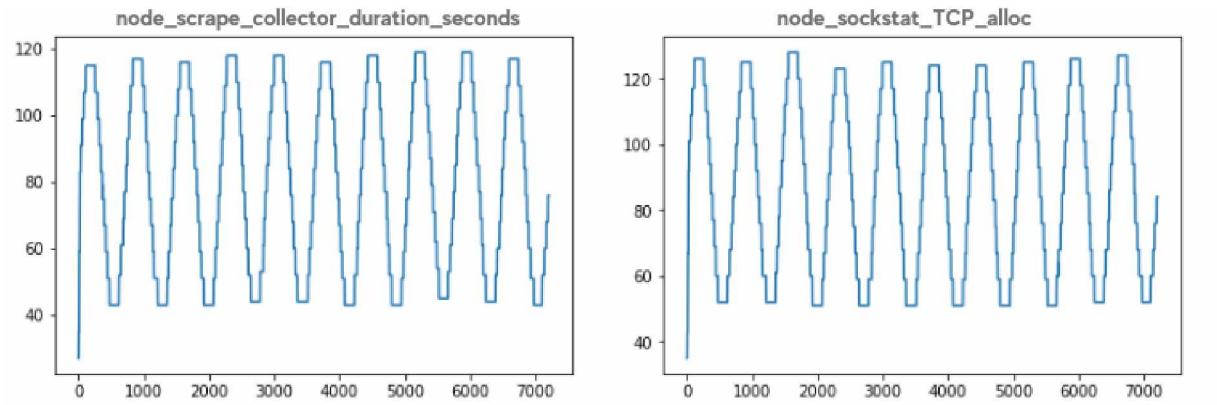

Figura 13 – As duas métricas mais correlacionadas ao tempo de escrita observado no Cenário 1 segundo o algoritmo de correlação.

A Figura 14 ilustra os gráficos das duas métricas mais correlacionadas à métrica de tempo de resposta do Cenário 2, isto é, do Cenário em que se impôs a restrição de $30Mbit/s$ na placa de rede. Também aqui, nota-se que, de fato, há grande similaridade entre as duas métricas apresentadas com o gráfico apresentado na Figura 8.

[1] https://github.com/gustavosm/traces-dissertacao-ufu

Neste caso, ambas as métricas mais correlacionadas ao tempo de resposta do Cenário 2 são a *node_netstat_Tcp_CurrEstab*, que indica o número de conexões TCP que estejam no *status: ESTABLISHED* ou *CLOSE-WAIT*. Todavia, a primeira foi coletada no nó 3 do *cluster* enquanto que a segunda foi coletada no nó 1.

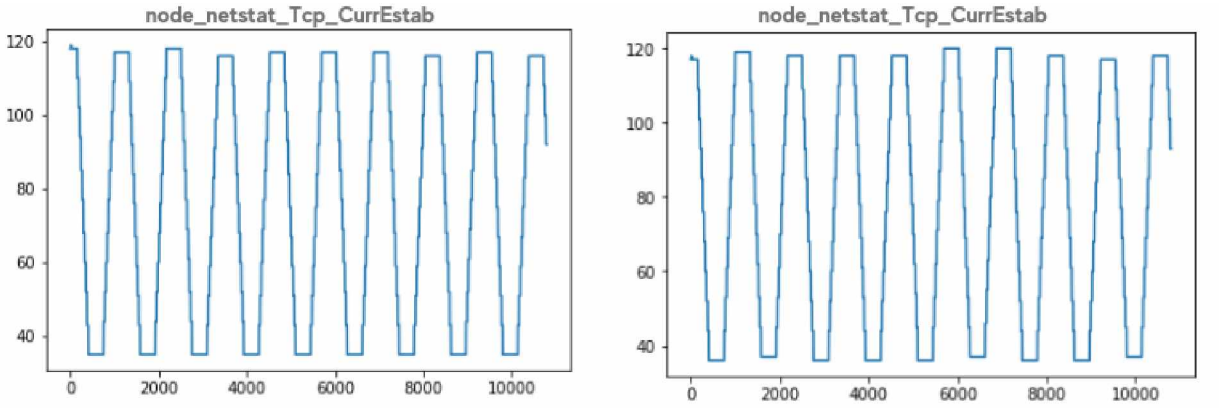

Figura 14 – As duas métricas mais correlacionadas ao tempo de escrita observado no Cenário 2 segundo o algoritmo de correlação.

A Figura 15 ilustra os gráficos das duas métricas mais correlacionadas à métrica de tempo de resposta do Cenário 3, isto é, do Cenário em que se impôs a restrição de $10Mbit/s$ na placa de rede. Todavia, neste caso, nota-se que a segunda métrica, tida como mais correlacionada ao tempo de resposta, apresentada na Figura 9, não segue exatamente o padrão observado. Isso é relevante, pois indica uma possível razão para que a precisão nesse caso tenha sido tão diferente da precisão obtida nos demais Cenários.

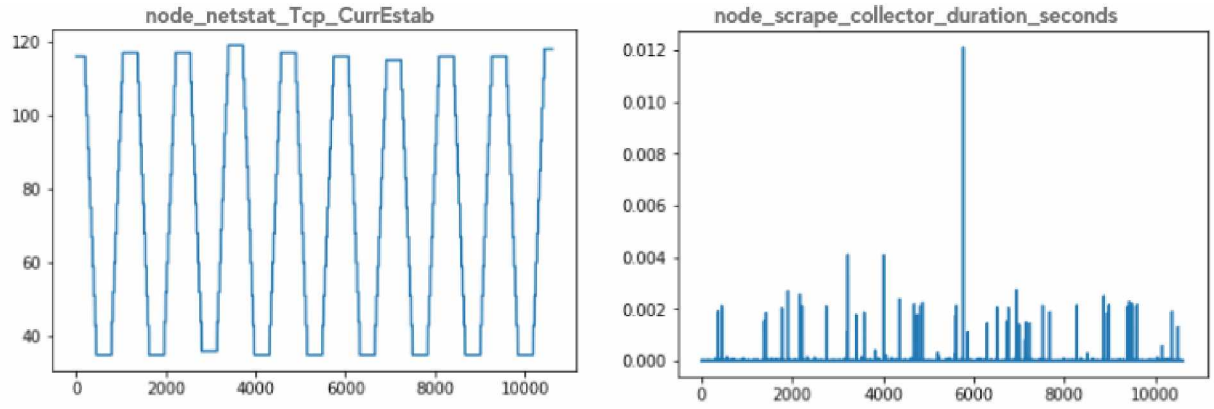

Figura 15 – As duas métricas mais correlacionadas ao tempo de escrita observado no Cenário 3 segundo o algoritmo de correlação.

As métricas mais correlacionadas ao tempo de resposta são, respectivamente: a métrica *node_netstat_Tcp_CurrEstab*, que indica o número de conexões TCP que estejam no *status: ESTABLISHED* ou *CLOSE-WAIT*; e, a *node_scrape_collector_duration_seconds*,

que indica o tempo em segundos gasto para a coleta de métricas naquele nó específico do *cluster*.

A análise dos resultados apresentados se encontra na Seção 5.3.

5.2.4 O algoritmo SFS

O segundo algoritmo testado, o SFS, foi aplicado em conjunto com o *Random Forest*. Caracterizando a técnica *wrapper*, utilizou-se este último para selecionar e para estimar o valor das métricas.

A abordagem utilizada foi a *Forward Stepwise Feature Selection*, mas com uma importante diferença. O conjunto completo de métricas, em alguns testes, chegou ao número de 706 métricas, o que inviabiliza completamente a utilização do algoritmo puro como está descrito no Capítulo 2, em virtude da sua alta complexidade computacional. Por conta disso, foi necessário fazer uma adaptação. Sempre que se insere uma nova métrica no conjunto, faz-se o cálculo da estimativa e espera-se que a precisão desta melhore. Entretanto, é de esperar que a partir de um determinado ponto a precisão apenas piore, devido a inserção de ruídos.

Então, pode-se indagar: que ponto é esse? Não se conhece. Esperava-se que a função de erro do $NMAE$ se comportasse como uma parábola, mas os testes realizados para definição dos parâmetros demonstrou que isso não era verdade. Buscou-se também definir uma função, e a partir daí se poderia encontrar os máximos e mínimos locais e globais para estudar qual o melhor K, mas também não foi uma abordagem possível.

Portanto, admitiu-se que, após a primeira estimativa que gerasse um erro 0.000001 maior que o anterior, iria se testar a inserção de apenas mais 10 métricas. Se, em algum momento da inserção dessas próximas 10 métricas, o erro fosse pelo menos 0.000001 menor que o menor erro obtido, recomeçaria-se a contagem. O número 10 foi escolhido com base em testes que mostraram que, a partir do primeiro momento em que a estimativa piora, não foi necessário inserir mais que 5 métricas para que a estimativa voltasse a melhorar. Dando, portanto, uma margem de erro, estipulou-se a tentativa em 10 métricas.

A justificativa para que o erro melhore, depois de piorar, se deve ao fato de que algumas métricas são complementares umas às outras. Desta forma, a inserção de apenas uma pode ser ruim, mas essa em conjunto com outra que ainda será inserida, pode melhorar a estimativa.

A Figura 16 ilustra os $NMAE$ obtidos nos testes do Cenário 1, isto é, sem restrições na placa de rede. Nesse gráfico, será possível notar que os testes pararam em determinado valor para K diferentes a depender do intervalo de coleta. Isso porque a execução parou ao atingir 10 métricas sem melhora na precisão da estimativa. Portanto, para saber o melhor K obtido para certo intervalo de coleta, é preciso considerar o K listado 10 números antes do último.

Com o intervalo de $1s$ de coleta, o melhor $NMAE$ se deu com $K = 12$ e $NMAE = 17.39\%$. Para o intervalo de $5s$, o melhor $NMAE$ se deu com $K = 7$ e $NMAE = 17.90\%$. No intervalo de $10s$, o melhor $NMAE$ se deu com $K = 7$ e $NMAE = 18.33\%$. E, por fim, com o intervalo de $30s$, o melhor $NMAE$ se deu com $K = 7$ e $NMAE = 20.04\%$.

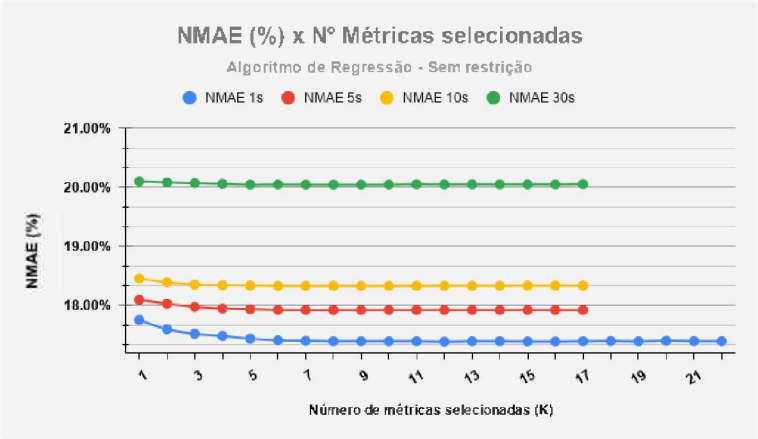

Figura 16 – $NMAE$ observada para os diferentes valores de K utilizados para seleção de métricas que compõem o conjunto $X_{Selecionado}$, com diferentes intervalos de coleta no Cenário 1.

A Figura 17 ilustra os $NMAE$ obtidos nos testes do Cenário 2, isto é, com restrição de $30Mbit/s$ na placa de rede. Como no gráficos anterior, aqui também há, para cada intervalo de coleta, 10 números K a mais do que o K ideal.

Com o intervalo de $1s$ de coleta, o melhor $NMAE$ se deu com $K = 16$ e $NMAE = 15.72\%$. Para o intervalo de $5s$, o melhor $NMAE$ se deu com $K = 8$ e $NMAE = 16.43\%$. No intervalo de $10s$, o melhor $NMAE$ se deu com $K = 3$ e $NMAE = 16.66\%$. E, por fim, com o intervalo de $30s$, o melhor $NMAE$ se deu com $K = 3$ e $NMAE = 16.92\%$.

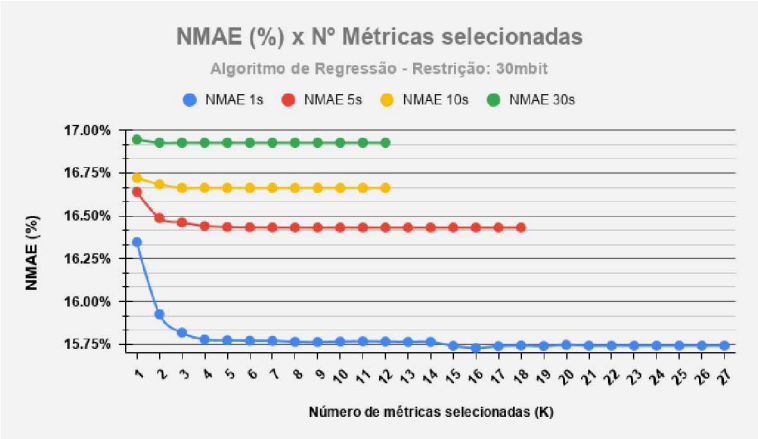

Figura 17 – $NMAE$ observada para os diferentes valores de K utilizados para seleção de métricas que compõem o conjunto $X_{Selecionado}$, com diferentes intervalos de coleta no Cenário 2.

A Figura 18 ilustra os $NMAE$ obtidos nos testes do Cenário 3, isto é, com restrição de $10Mbit/s$ na placa de rede. Como nos gráficos anteriores, aqui também há, para cada intervalo de coleta, 10 números K a mais do que o K ideal.

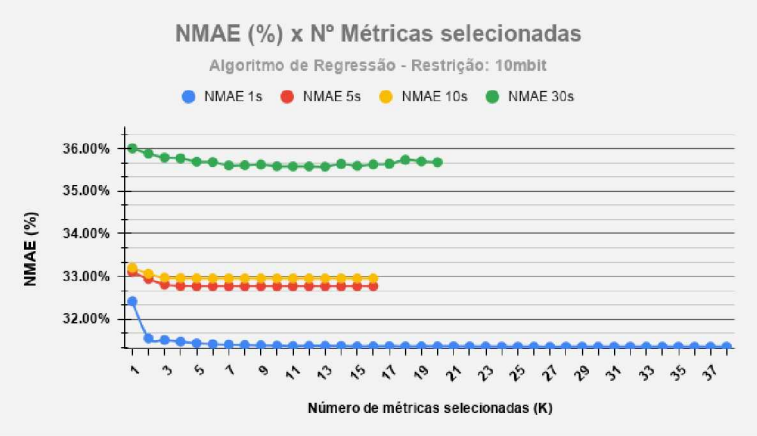

Figura 18 – $NMAE$ observada para os diferentes valores de K utilizados para seleção de métricas que compõem o conjunto $X_{Selecionado}$, com diferentes intervalos de coleta no Cenário 3.

Com o intervalo de $1s$ de coleta, o melhor $NMAE$ se deu com $K = 27$ e $NMAE = 31.35\%$. Para o intervalo de $5s$, o melhor $NMAE$ se deu com $K = 6$ e $NMAE = 32.77\%$. No intervalo de $10s$, o melhor $NMAE$ se deu com $K = 6$ e $NMAE = 32.95\%$. E, por fim, com o intervalo de $30s$, o melhor $NMAE$ se deu com $K = 11$ e $NMAE = 35.57\%$.

5.2.4.1 As métricas do algoritmo SFS

Assim como no caso do algoritmo de correlação foram detalhadas as duas primeiras métricas selecionadas, a seguir tem-se as duas primeiras métricas selecionadas pelo algoritmo SFS nos três Cenários de testes. Mesmo com o algoritmo SFS obteve-se um número de métricas que não seria adequado ilustrar aqui. No entanto, o leitor poderá encontrar as métricas selecionadas, bem como seus valores no Github[2] do autor da dissertação.

A Figura 19 ilustra os gráficos das duas primeiras métricas selecionadas para o Cenário 1, isto é, do Cenário em que não houve restrição na placa de rede. As métricas selecionadas são, respectivamente, a métrica $node\_sockstat\_TCP\_alloc$, que calcula o número de *sockets* TCP alocados naquele segundo em determinado nó e a $node\_filefd\_allocated$, que indica o número de descritores de arquivos alocados naquele segundo.

A primeira métrica foi coletada no nó 1 do *cluster* e a segunda no nó 3. Nota-se que a segunda métrica, embora tenha comportamento similar ao observado no tempo de resposta apresentado na Figura 7, ela não é exatamente a mais correlacionada diretamente com a métrica Y. Isso é relevante, porque o algoritmo de correlação não deu a mesma

[2] https://github.com/gustavosm/traces-dissertacao-ufu

importância para essa métrica, no entanto, ela de fato está correlacionada à carga aplicada ao serviço da DHT.

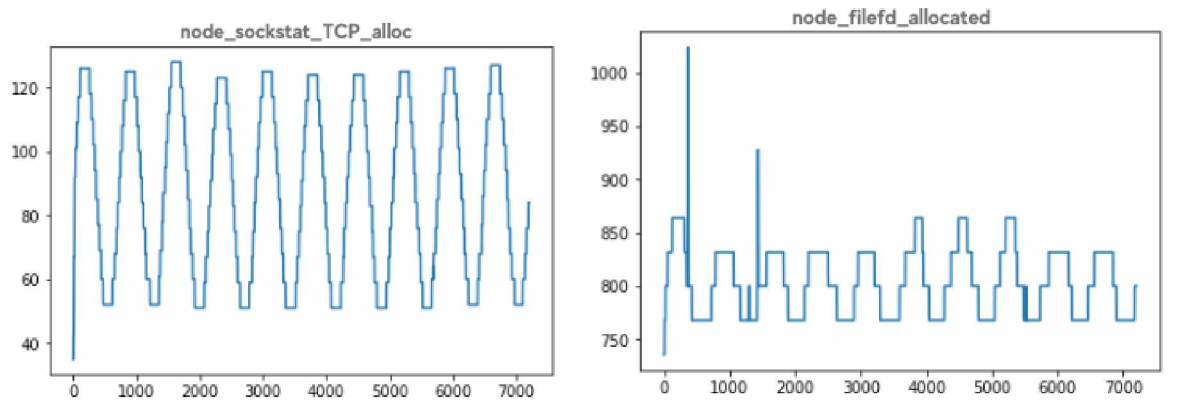

Figura 19 – As duas primeiras métricas selecionadas no Cenário 1 segundo o algoritmo SFS

A Figura 20 ilustra os gráficos das duas primeiras métricas selecionadas para o Cenário 2, isto é, do Cenário em que se impôs a restrição de $30Mbit/s$ na placa de rede. As métricas selecionadas, neste caso, são a *node_sockstat_TCP_inuse*, que indica o número de *sockets* TCP em uso naquele segundo, e a métrica *go_memstats_buck_hash_sys_bytes*, que indica o número de *bytes* utilizados pelos *buckets* da tabela *hash* dos perfis criados pelo *exporter*.

Nota-se a grande diferença de comportamento existente entre a segunda métrica selecionada e o tempo de resposta apresentado na Figura 8.

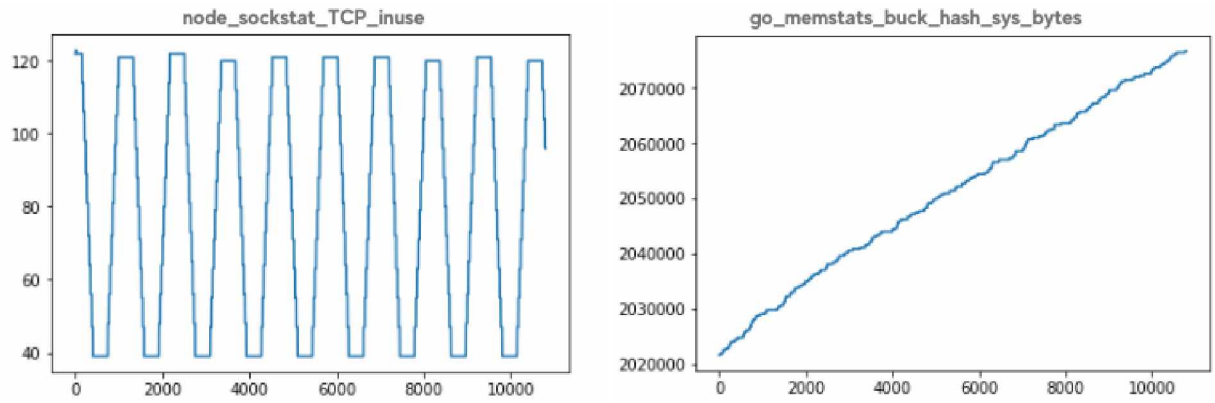

Figura 20 – As duas primeiras métricas selecionadas no Cenário 2 segundo o algoritmo SFS.

A Figura 21 ilustra os gráficos das duas primeiras métricas selecionadas para o Cenário 3, isto é, do Cenário em que se impôs a restrição de $10Mbit/s$ na placa de rede.

As métricas selecionadas são, respectivamente: a métrica *node_sockstat_sockets_used*, que indica o número de *sockets* utilizados naquele segundo, independentemente do pro-

tocolo; e, a *go_memstats_last_gc_time_seconds*, que indica o tempo em segundos, a partir de 1970, desde a última execução do *garbage collector*.

Como observado no Cenário 2, nota-se aqui que a segunda métrica selecionada pelo algoritmo SFS não possui um comportamento similar ao do tempo de resposta apresentado na Figura 9. É interessante observar que o comportamento dessa segunda métrica se assemelha ao da segunda métrica selecionada no Cenário 2.

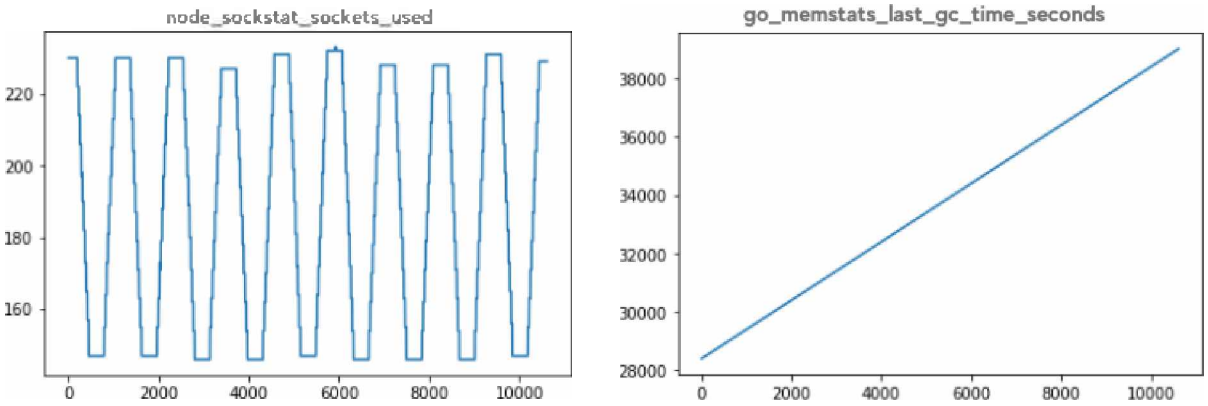

Figura 21 – As duas primeiras métricas selecionadas no Cenário 3 segundo o algoritmo SFS.

5.3 Avaliação dos Resultados

Essa seção analisa os resultados obtidos tanto pelo algoritmo de correlação quanto pelo SFS.

5.3.1 Comparativo entre os dois algoritmos

Nesta subseção estabeleceu-se a comparação entre os dois algoritmos, confrontando os resultados, erros e tempos de execução, com intuito de definir se há um entre os dois que seja melhor dentro do contexto do NECOS, ou se tanto um quanto o outro poderiam ser usados.

A Tabela 1 ilustra uma comparação entre os dois algoritmos para o Cenário 1. Observando a taxa de erro $NMAE$, bem como o número K de métricas selecionadas, nota-se alguns pontos importantes descritos a seguir:

- o primeiro é que o algoritmo SFS não sofreu uma grande perda de precisão quando o intervalo de coleta subiu de $1s$ para $5s$, nem mesmo quando subiu para $10s$. A taxa $NMAE$ variou de 17.39% para 18.33%, sendo menos de 1% de diferença. No entanto, no algoritmo de correlação a taxa $NMAE$ sofreu uma variação de mais de 4% também considerando os três primeiros intervalos de coleta. Desta forma, dado

o padrão de carga adotado, o algoritmo SFS não sofre grandes perdas de precisão, mesmo com os dados coletados em intervalos maiores.

- um segundo ponto importante a ser destacado é a grande diferença no número K observado entre os dois algoritmos. O algoritmo SFS se manteve com menos de 12 métricas, não obstante a diferença de tempo de coleta, enquanto que o algoritmo de correlação chegou a usar quase que a totalidade das métricas disponíveis no caso do tempo de coleta de $10s$.

- por fim, nota-se que o algoritmo de correlação teve um desempenho pior que o algoritmo SFS em todos os intervalos de coleta, com o detalhe de que o intervalo de $10s$ para o algoritmo SFS gerou um resultado ligeiramente melhor que o intervalo de $1s$ do de correlação, o que propõe que o primeiro trabalha melhor que o segundo mesmo tendo menos dados.

Esses três pontos levantados são relevantes para esta dissertação, porque um dos objetivos é reduzir o volume de dados trafegados entre o IMA e os demais módulos do NECOS. Nesse sentido, é importante perceber que a coleta com intervalo de $10s$ gera menos dados que a coleta de $1s$, e mesmo nesse cenário, dado o padrão de carga aplicado, o algoritmo SFS sofre pouca perda. Neste caso, os testes indicam que este algoritmo pode ser a melhor opção dentre os dois que foram testados para a implementação do *SIMON* no contexto do NECOS.

	Correlação	SFS
1s	K=23, NMAE=18.91%	K=12, NMAE=17.39%
5s	K=39, NMAE=22.38%	K=7, NMAE=17.90%
10s	K=650, NMAE=23.40%	K=7, NMAE=18.33%
30s	K=268, NMAE=25.67%	K=7, NMAE=20.04%

Tabela 1 – Tabela de comparação entre os algoritmos de correlação e SFS, confrontando o número K obtido, bem como a taxa de erro $NMAE$ para o Cenário 1.

A Tabela 2 ilustra a comparação entre os dois algoritmos para o Cenário 2. Aqui, nota-se que o algoritmo de correlação se mantém bem próximo do SFS, chegando a selecionar menos métricas para o intervalo de coleta $5s$, todavia, quando o intervalo sobe para $10s$ e $30s$ a perda de precisão é significativa, aumentando em mais de 5%, enquanto a variação no algoritmo SFS é pouco mais de 1%, quando se compara o tempo de coleta de $1s$ com o de $30s$.

Novamente, o quadro observado no Cenário 1 se mantém mesmo com a infraestrutura alterada. Entretanto, resta ainda observar a mudança obtida no Cenário 3, pois conforme mostrado anteriormente, a degradação do serviço foi mais expressiva quando aplicada a restrição de $10Mbit/s$ na placa de rede.

	Correlação	SFS
1s	K=19, NMAE=15.83%	K=16, NMAE=15.72%
5s	K=4,NMAE=16.62%	K=8,NMAE=16.43%
10s	K=94,NMAE=20.50%	K=3,NMAE=16.66%
30s	K=546,NMAE=21.98%	K=3,NMAE=16.92%

Tabela 2 – Tabela de comparação entre os algoritmos de correlação e SFS, confrontando o número K obtido, bem como a taxa de erro $NMAE$ para o Cenário 2.

A Tabela 3 mostra um comparativo entre os dois algoritmos quando no Cenário 3. Definitivamente, este foi o pior Cenário para ambos os algoritmos. A taxa $NMAE$ foi a mais alta - sempre acima de 30% -, e para o algoritmo SFS, foi o Cenário em que o tempo de coleta de $1s$ demandou um maior número de métricas.

Entretanto, ainda aqui nota-se que o algoritmo SFS manteve uma baixa variação da taxa $NMAE$, não obstante a mudança de intervalos de coleta, à exceção do intervalo de $30s$ que aqui, como nos dois Cenários anteriores, a variação foi maior. Mesmo assim, a variação observada no algoritmo SFS é notoriamente menor que a observada no algoritmo de correlação, não importando qual o intervalo de coleta considerado.

É importante ressaltar, também, que a degradação causada no serviço prejudicou consideravelmente a taxa de erro de ambos os algoritmos. Se antes as taxas atingiram pouco mais de 20% a 25%, aqui elas chegaram a 35% e 47%.

	Correlação	SFS
1s	K=662, NMAE=34.04%	K=27, NMAE=31.35%
5s	K=99,NMAE=38.54%	K=6,NMAE=32.77%
10s	K=296,NMAE=38.50%	K=6,NMAE=32.95%
30s	K=434,NMAE=47.17%	K=11,NMAE=35.57%

Tabela 3 – Tabela de comparação entre os algoritmos de correlação e SFS, confrontando o número K obtido, bem como a taxa de erro $NMAE$ para o Cenário 3.

Esses comparativos demonstram que, mesmo a taxa $NMAE$ não sendo muito diferente entre os algoritmos, a depender do intervalo de coleta, o número K de métricas selecionadas é consideravelmente menor no algoritmo SFS para a maioria dos casos testados. Isso é importante para que se defina a abordagem com o aprendizado de máquina como aquela em que se validará as hipóteses levantadas no Capítulo 1.

Um fator também importante a ser considerado aqui é o tempo de execução. Apesar de no algoritmo de correlação não se ter colocado a mesma tratativa inserida no SFS - interromper a execução depois de 10 métricas com piora na estimativa -, o tempo de execução de ambos não ficou muito diferente.

A Tabela 4 apresenta um comparativo no tempo de execução de ambos os algoritmos nos três Cenários propostos. Nota-se que, em média, o algoritmo de correlação é mais rápido, mesmo testando todas as métricas possíveis. De fato, a complexidade computacional deste é menor, o que faz esperar um tempo de execução igualmente menor. Todavia,

não se considera que o tempo de execução seja um fator para descartar esse algoritmo, sobretudo tendo em vista os resultados apresentados sobre o $NMAE$ e a quantidade K de métricas selecionadas. No contexto do NECOS, os tempos de execução apresentados são aceitáveis, dado que o serviço de seleção pode funcionar de forma assíncrona, e não será executado com frequência que possa ser impactada pelo tempo gasto na seleção.

A grande diferença do tempo de execução notada no Cenário 1 para os demais se justifica no fato de que nesse caso o experimento durou 2 horas e não 3 como nos demais.

	Correlação	SFS
Cenário 1	17m22s	16m
Cenário 2	26m10s	34m20s
Cenário 3	25m30s	31m23s

Tabela 4 – Tabela de comparação entre do tempo de execução dos algoritmos de correlação e SFS nos três Cenários propostos.

5.3.2 Sobre as Hipóteses

No Capítulo 1 levantou-se três hipóteses a serem validadas ou refutadas neste trabalho. Tais hipóteses estão reescritas a seguir:

- **H1**. Existe um conjunto de características assertivo, isto é, um conjunto reduzido em relação ao conjunto original (completo) de características que permita uma boa predição de métricas de qualidade de serviço.

- **H2**. Existe um conjunto de características que permite boas predições de métricas, ainda que a infraestrutura sofra alterações nos recursos ofertados.

- **H3**. As ferramentas são capazes de prover dados em granularidade suficiente e permitir a filtragem de características.

Uma vez que o algoritmo SFS se mostrou melhor nos comparativos com o algoritmo de correlação, optou-se por utilizá-lo na validação das Hipóteses levantadas. Desta forma, buscou-se responder as perguntas feitas para cada Hipótese tendo por base os resultados obtidos com esse algoritmo.

5.3.2.1 A Hipótese H1

Para a Hipótese **H1**, questionou-se o seguinte:

P1.1 **É possível selecionar os dados do conjunto completo que mais interferem na qualidade dos serviços analisados?**

Aqui, pode-se responder afirmativamente, pois o algoritmo foi capaz de gerar um subconjunto, consideravelmente menor que o conjunto completo, que se mostrou intimamente relacionado com a métrica de qualidade de serviço escolhida.

P1.2 **Comparando o resultado obtido através do conjunto completo de características com o conjunto reduzido, a predição das métricas sofre alteração significativa, ou seja, a margem de acerto varia de forma a tornar inviável a redução do conjunto?**

Considerando o Cenário 3 como parâmetro, por ter sido o pior Cenário para o algoritmo SFS, veremos que: com intervalo de coleta de $1s$, o $NMAE$ observado no conjunto completo foi de 34.2%, enquanto que no conjunto selecionado de 27 métricas foi de 31.35%. Uma ligeira redução em termos de erro, porém significativa em termos de quantidade de métrica. No tempo de coleta de $5s$, o $NMAE$ do conjunto completo foi de 41.08%, contra 32.77% no conjunto com apenas 6 métricas. Com o intervalo de $10s$, o $NMAE$ do conjunto completo foi de 124.47% enquanto que no conjunto reduzido de 6 métricas foi de 32.95%. Por fim, no último intervalo de coleta testado de $30s$, observou-se um $NMAE$ de 51.64% no conjunto completo, e no conjunto de 11 métricas foi de 35.57%. É importante ressaltar que o conjunto completo nesse caso possui 706 métricas.

Portanto, conclui-se que não apenas não houve pioras na estimativa, como houve significativas melhoras. E, não apenas em termos de taxa de $NMAE$, onde percebe-se uma precisão consideravelmente melhor, como também na quantidade de métricas. A redução nos conjuntos para esse Cenário, chegou a mais de 98%. Desta forma, a seleção se mostrou uma abordagem viável do ponto de vista de estimativas e redução de dados trafegados, e com um tempo de execução, para o contexto do NECOS, aceitável.

P1.3 **Não havendo alteração significativa e comparando o resultado obtido através do conjunto completo de características com o conjunto reduzido, o tempo de execução para predição das métricas caiu de maneira satisfatória a ponto de ser necessária a redução?**

Conforme mostrado na resposta anterior, é possível afirmar que sim, houve queda significativa, e torna a seleção necessária. A Figura 22 demonstra a notória redução de dados trafegados, o que contribui ainda mais para que a seleção de métricas seja realizada.

5.3.2.2 A Hipótese H2

Para a Hipótese H2, fez-se o seguinte questionamento:

P2.1 **Será que, alterando os recursos da máquina hospedeira do serviço, não será necessário refazer a seleção novamente?**

Para validar esta Hipótese e responder a pergunta, aplicou-se duas restrições na placa de rede das máquinas do *cluster*. A primeira reduziu a capacidade da placa de $1Gbit/s$ para $30Mbit/s$, e posteriormente aplicou-se a segunda que reduziu de $30Mbit/s$ para $10Mbit/s$. Com isso, foi possível realizar três comparações que permitem uma análise da Hipótese.

Considerando o intervalo de coleta de $1s$, buscou-se estimar as métricas de dois Cenários utilizando o conjunto selecionado no Cenário restante. O intuito aqui é concluir se

a estimativa sofre pioras significativas, tendo mantido o conjunto selecionado e realizado alterações na infraestrutura.

Os resultados estão descritos a seguir:

- Ao estimar a KPI do serviço do Cenário 2 com as métricas selecionadas do Cenário 1, obteve-se um $NMAE$ de 16.18%. Nota-se que houve uma piora na estimativa, que havia sido de 15.72%, entretanto ela é inferior a 0.5%.

- Igualmente buscou-se estimar a KPI do serviço do Cenário 3 com o conjunto reduzido do Cenário 1 e obteve-se um $NMAE$ de 30.25%. Interessante que neste caso o $NMAE$ que antes era de 31.35% sofreu uma redução de 1.10%.

- Também se estimou a KPI do serviço do Cenário 1 utilizando-se as métricas selecionadas no Cenário 2. Neste caso, o $NMAE$ que antes havia sido de 17.39% foi de 18.86%, tendo, portanto, um aumento de 1.47%.

- Ao estimar a KPI do serviço no Cenário 3 utilizando as métricas selecionadas no Cenário 2, obteve-se um $NMAE$ igual a 31.25%, tendo uma ligeira redução de 0.1%.

- E por fim, testou-se as métricas selecionadas no Cenário 3 nos dois outros Cenários. Quando no Cenário 1 o $NMAE$ ficou em 18.71% e quando no Cenário 2 ficou em 15.98%, apresentando ligeiros aumentos em ambos, mas sempre inferiores a 1.5%.

Portanto, no caso desta proposta de alteração nos recursos de rede da infraestrutura, os testes indicam que um conjunto selecionado em um Cenário continua sendo válido para os outros, ainda que piorando ligeiramente a estimativa. Por isso, conclui-se que a Hipótese é verdadeira para o caso testado. Este resultando é muito relevante, por exemplo, para o orquestrador do NECOS, pois permite que o orquestrador continue empregando os modelos de inteligência artificial que foram treinados para uma determinada configuração da *slice* em outras configurações, após ações de elasticidade, até que modelos mais adequados/atualizados sejam treinados para os novos arranjos da *slice*.

Todavia, aqui ainda será necessário testar outras formas de alteração na infraestrutura, bem como fazer uma combinação destas, para que se possa ter mais garantias de que se pode manter o conjunto selecionado mesmo que a infraestrutura sofra alterações. Tal proposta será melhor detalhada como trabalho futuro no Capítulo 6.

5.3.2.3 A Hipótese H3

Para a última Hipótese, fez-se a seguinte pergunta:

P3.1 **Existe pelo menos uma ferramenta que cumpre a hipótese H3?**

A ferramenta testada neste trabalho foi o *Prometheus*. Conforme dito, esse sistema de coleta de dados possui um arquivo de configuração que permite ajustar as métricas que

serão ou que não serão coletadas. Tal configuração não impede que a métrica seja gerada, mas evita o seu tráfego para os níveis acima da camada em que se executa o *Prometheus*.

Nesse sentido, feita a seleção de características, duas instâncias do *Prometheus* foram iniciadas: uma coletando o conjunto completo de dados, isto é, coletando todas as 706 métricas exportadas pelo *node exporter*, e outra para coleta apenas do conjunto reduzido de métricas construído no Cenário 1, ambos com o tempo de coleta em $1s$. A Figura 22 ilustra a diferença do volume total de dados trafegados na rede.

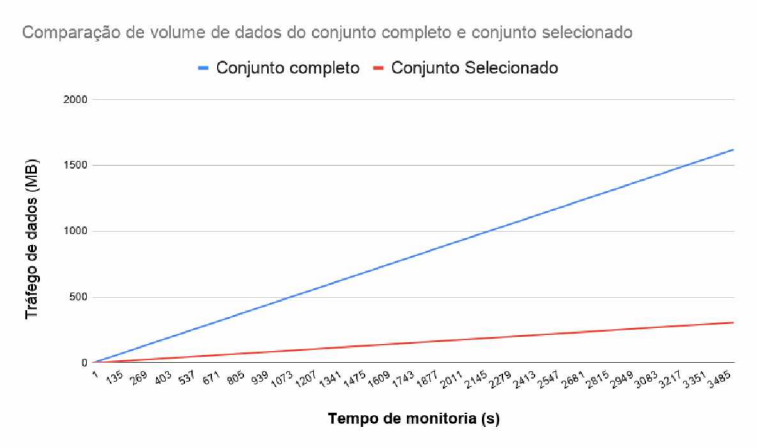

Figura 22 – Diferença entre o volume de tráfego entre a coleta de conjunto completo de métricas da infraestrutura com o conjunto selecionado.

É possível notar a diferença de crescimento das retas, que indica o volume total de dados trafegados. Ao término da análise, o conjunto completo havia gerado um total de $1,6GB$ de dados, enquanto que o conjunto selecionado gerou $307MB$. Em média, o conjunto completo trafega $450KB$ de dados por segundo, enquanto que o selecionado $85KB$.

Desta forma, mostra-se que o *Prometheus* valida a Hipótese H3, porque oferece mecanismos de descarte de métricas evitando o tráfego dessas informações entre o IMA e os VIM/WIM o que ameniza o problema da sobrecarga na rede de controle caso fosse necessário sempre trafegar o conjunto completo de métricas.

Assim, as três Hipóteses puderam ser analisadas e validadas dentro da proposta feita nesta dissertação.

Capítulo 6

Conclusão

O cenário de investigação do projeto NECOS é desafiador. Por se tratar de um sistema em que a escalabilidade e a disponibilidade são fatores de suma importância, os tempos de resposta de cada módulo devem ser os menores possíveis. Desta forma, o módulo de monitoria possui um requisito de selecionar as características de maneira rápida, sem perder eficácia, para que possibilite aos demais módulos maior velocidade e acurácia na tomada de decisões.

Nesse sentido, a solução inicial do módulo de monitoria proposta no NECOS possivelmente não atenderia os requisitos, em decorrência das dificuldades discutidas no Capítulo 3. Trabalhar com todos os dados da infraestrutura é uma carga alta, que demanda muito processamento e pode afetar, consideravelmente, a precisão das estimativas e predições, conforme demonstrado nos testes, mas também pode afetar a própria disponibilidade do sistema. A demora na coleta dos dados, a sobrecarga na rede de controle e, posteriormente, o esforço computacional para gerenciar e analisar toda essa informação extraída da infraestrutura exigiria um grande tempo de resposta, bem como uma grande demanda de recursos e como consequência há uma lentidão no sistema e eventuais indisponibilidades.

Como alternativa, poderia-se propor um conjunto fixo e reduzido de métricas a serem consideradas, mas essa abordagem não deve ser flexível o bastante, porque não se adapta aos serviços hospedados nas *slices*. Também é evidente que não há condições reais de se saber de antemão qual serviço será hospedado, e ainda que se soubesse, a seleção manual de métricas não pode ser a opção de um sistema que pretende ser escalável.

Assim, o módulo de monitoria tinha a demanda de ser flexível, dinâmico, rápido, eficaz e viável do ponto de vista de implementação. Esses foram os objetivos principais na construção do *SIMON* e os resultados apresentados indicam que foram atendidos na totalidade.

Apresentando os resultados, o *SIMON* mostrou que a seleção de características pode ser feita sem nenhum conhecimento prévio do serviço, nem mesmo da infraestrutura. Deem-lhe as métricas e ele retorna o conjunto reduzido que ajudará a estimar as métricas de qualidade do serviço. Isso é um ponto muito importante, porque em um cenário real

dificilmente será possível saber qual serviço está sendo hospedado em tal *slice*.

Como resultado da seleção de características, um volume menor de métricas passa a ser fornecido aos módulos, fator central para a escalabilidade de todo o sistema. Conforme os resultados apresentados, a seleção de características contribui, inclusive, para uma melhor estimativa das métricas Y. Portanto, apesar de o cenário ser bastante desafiador, os primeiros testes são promissores e indicam a importância do sistema inteligente de monitoramento apresentado neste trabalho. Além disso, o presente trabalho cumpriu todos os objetivos a que se propôs, conforme especificado na seção a seguir, e que foram detalhados no Capítulo 1.

6.1 Dos objetivos

O presente trabalho se propôs a atingir um objetivo geral e quatro objetivos específicos, detalhados no Capítulo 1.

O objetivo geral foi assim descrito: construir um módulo de monitoramento que identifica e coleta um conjunto reduzido e essencial de características de cada *slice* utilizando aprendizado de máquina. Este conjunto essencial de características deve suportar as diferentes demandas de gerenciamento das *slices*, buscando manter a qualidade dos serviços nelas instanciados.

Conforme mostrado nos testes, a seleção de características feita pelo *SIMON* através do algoritmo SFS foi capaz de:

- construir um conjunto reduzido de características que permitiu reduzir o $NMAE$ do conjunto completo, além de permitir uma boa redução no conjunto de dados a serem monitorados;

- construir um conjunto reduzido de características que manteve a taxa de erro $NMAE$ próxima à observada nas diferentes configurações da infraestrutura mesmo tendo sido gerado em uma configuração específica. Assim é que, por exemplo, observou-se que o conjunto gerado em uma infraestrutura sem restrição na placa de rede manteve o $NMAE$ próximo ao do conjunto gerado no cenário com restrição de $30mbit/s$ quando ambos são aplicados às métricas desse mesmo cenário. Isso significa que o conjunto selecionado em uma configuração de infraestrutura poderia ser mantido ainda que ocorresse alguma das alterações propostas.

Desta forma, pode-se dizer que o objetivo geral do trabalho foi cumprido na sua totalidade. Quanto aos objetivos específicos, foram listados:

1. Avaliar ferramentas de coletas dos dados no sentido de granularidade dos dados coletados e possibilidade de filtragem dos dados, isto é, capacidade de descartar

informações não relevantes, sendo considerado tão melhor quanto mais próximo da fonte dos dados estas informações possam ser descartadas;

2. Avaliar se os algoritmos retornam um bom conjunto reduzido de características, ou seja, se as características selecionadas permitem uma boa estimativa de qualidade dos serviços analisados;

3. Avaliar o quanto o volume de dados trafegados na rede é reduzido após a seleção;

4. Avaliar se o conjunto de características selecionadas continua sendo bom, isto é, continua permitindo boas predições, ainda que a infraestrutura sofra alterações.

Nesse sentido, o primeiro objetivo específico foi cumprido pela a avaliação da ferramenta *Prometheus*. Com ela, mostrou-se que os dados puderam ser coletados em granularidades específicas, bem como o seu descarte ocorreu o mais próximo possível da sua fonte. O *Prometheus* não impede que uma métrica seja gerada, embora esteja marcada para descarte, todavia isso é um ponto positivo, dado que, eventualmente, sendo necessário refazer a seleção de características, todas as métricas precisam ser consideradas novamente. A ferramenta apenas não coleta a métrica, o que reduz o tráfego na rede de controle.

Com relação ao segundo objetivo, a avaliação dos resultados mostrou que os conjuntos selecionados obtiveram uma boa taxa de erro, ficando, na maioria dos cenários em torno dos 17%. Embora este número esteja ainda relativamente distante dos resultados obtidos em (PASQUINI; STADLER, 2017), é importante observar que não se considera o $NMAE$ em 17% como uma taxa alta de erro, no contexto do NECOS. Além disso, é importante notar que no trabalho referido testou-se outras ferramentas de coleta, outras aplicações, mas principalmente o padrão de carga aplicado foi bastante diferente do aplicado nesta dissertação.

O terceiro objetivo foi alcançado demonstrando-se pela Figura 22. A quantidade de dados trafegados pelo conjunto completo foi reduzida em mais de 75% quando comparada com um dos conjuntos selecionados nos testes.

Por fim, o quarto e último objetivo foi demonstrado quando se utilizou as métricas selecionadas no Cenário 1 (sem restrição de largura de banda) nos Cenários 2 e 3 (com restrições de largura de banda) e vice-versa. A taxa $NMAE$ se manteve, não obstante as alterações feitas na infraestrutura, o que mostra que o conjunto construído é estável e continua sendo útil, mesmo com recursos alterados.

6.2 Principais Contribuições

As principais contribuições deste trabalho são:

- ❑ Mostrar a viabilidade das melhorias propostas para o sistema de monitoria do NECOS, tornando-o mais robusto do ponto de vista computacional e operacional;

- Oferecer à comunidade científica um sistema de código aberto que integra com o *Prometheus* e realiza a seleção de características, devolvendo um conjunto de métricas menor, mais preciso e dinâmico;

- Oferecer à comunidade científica os dados em repositório aberto no GitHub que permitem a verificação do exposto no Capítulo 5;

- Demonstrar a aplicação das técnicas de seleção de características em mais um cenário de monitoria de infraestrutura, ao lado de outros trabalhos que também se lançaram nessa proposta;

- Avaliar a tecnologia de monitoramento *Prometheus* em conjunto com outras ferramentas de muito uso atualmente como *Kubernetes* e *OpenStack*, apontando seus benefícios e auxiliando quem deseje utilizá-la.

6.3 Trabalhos Futuros

O conjunto de trabalhos futuros inclui:

1. experimentar outros serviços, que não o da DHT, conforme indicado na Seção 2. Serviços ofertados atualmente como de *streaming* de vídeos com o protocolo *DASH* podem ser uma boa alternativa, porque sofrem alterações dinâmicas na qualidade do serviço o que pode representar mais um desafio para os algoritmos de aprendizado de máquina;

2. analisar os resultados obtidos com outros algoritmos em conjunto com as técnicas de *filter* e *wrapper* que não os de correlação e o *Stepwise Foward Selection*, com intuito de buscar melhorar ainda mais o conjunto selecionado e a precisão das estimativas;

3. testar a viabilidade de se manter o mesmo conjunto de métricas selecionadas, provocando outras alterações de recursos na infraestrutura que não a feita na placa de rede, bem como no número de instâncias do serviço. Buscar realizar alterações de memória RAM, de número de CPU's, aumentar ou diminuir o número de réplicas no *cluster* da aplicação;

4. testar outros padrões de carga que não sejam tão suaves na variação de carga como o padrão aplicado. Padrões como *flashcrowd* como aplicado em (PASQUINI; STADLER, 2017) aplicam cargas repentinas no serviço, com intuito de simular um cenário em que o sistema é acessado em massa com um intervalo pequeno de tempo. Assim, poderia se testar mais um cenário real possível de acontecer no dia-a-dia de uma aplicação hospedada com o NECOS.

6.4 Contribuições em Produção Bibliográfica

O presente trabalho resultou em dois artigos, descritos a seguir:

- MARQUES, G.; REZENDE, A.; CUNHA, I.; FIALHO, R.; PASQUINI, R. "Arcabouço de um Sistema Inteligente de Monitoramento para Cloud Slices". In: Anais do I Workshop de Teoria, Tecnologias e Aplicações de Slicing para Infraestruturas Softwarizadas (WSlice). SBRC, 2019. p. 56-68, Gramado/RS, Brazil.
- MARQUES, G.; PASQUINI, R; FIALHO, R. "Towards Intelligent Monitoring of Cloud Slices". In: LANCOMM Student Workshop. SBRC, 2019. Gramado/RS, Brazil.

Referências

APACHE. **Apache Cassandra**. 2016. <http://cassandra.apache.org/>. Acessado em: 17/03/2019.

_____. **Introduction**. 2017. <https://kafka.apache.org/>. Acessado em: 21/01/2020.

BREIMAN, L. Random forests. **Machine Learning**, v. 45, n. 1, p. 5–32, Oct 2001. ISSN 1573-0565. Disponível em: <https://doi.org/10.1023/A:1010933404324>.

CEILOMETER. **Ceilometer**. 2018. <https://docs.openstack.org/ceilometer/latest/>. Acessado em: 21/12/2018.

CHANDRASHEKAR, G.; SAHIN, F. A survey on feature selection methods. **Computers & Electrical Engineering**, v. 40, n. 1, p. 16 – 28, 2014. ISSN 0045-7906. 40th-year commemorative issue. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0045790613003066>.

FOUNDATION, P. S. 2020. <https://docs.python.org/3/>. Acessado em 16/01/2020.

FULMARI1, A.; CHANDAK, M. B. **A Survey on Supervised Learning for Word Sense Disambiguation**. 2013. <https://pdfs.semanticscholar.org/58bb/8f4b9a0e7257ca15555e505e9fd35992f66c.pdf>. Acessado em: 17/03/2019.

GOOGLE. **RE2**. 2019. <https://github.com/google/re2/wiki/Syntax>. Acessado em 17/01/2020.

GROUP, T. P. G. D. **PostgreSQL**. 1996. <https://www.postgresql.org/>. Acessado em: 21/01/2020.

INFLUXDATA. **Influx DB**. 2017. <https://portal.influxdata.com/>. Acessado em: 21/01/2020.

INRIA. **Scikit Learn**. 2018. <https://scikit-learn.org/stable/>. Acessado em: 12/12/2018.

_____. **RandomForestClassifier method**. 2019. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier>. Acessado em: 17/01/2020.

_____. **RandomForestRegressor method**. 2019. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html?highlight=random%20forest#sklearn.ensemble.RandomForestRegressor>. Acessado em: 16/01/2020.

_____. **SelectKBest method**. 2019. <https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html#sklearn.feature_selection.SelectKBest>. Acessado em: 16/01/2020.

JOHN, G. H.; KOHAVI, R.; PFLEGER, K. Irrelevant features and the subset selection problem. In: **ICML 1994**. [S.l.: s.n.], 1994.

KIRA, K.; RENDELL, L. A. The feature selection problem: Traditional methods and a new algorithm. In: . [S.l.: s.n.], 1992. p. 129–134.

KOCHIE, B. **Prometheus node-exporter**. 2018. <https://github.com/prometheus/node_exporter/releases>. Acessado em: 06/02/2019.

KUBERNETES. **Production-Grade Container Orchestration**. 2018. <https://kubernetes.io/>. Acessado em: 12/12/2018.

_____. **What is Kubernetes**. 2018. <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>. Acessado em: 21/01/2020.

KäCHELE, S. et al. Beyond iaas and paas: An extended cloud taxonomy for computation, storage and networking. In: **2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing**. [S.l.: s.n.], 2013. p. 75–82.

LITJENS, G. et al. **A Survey on Deep Learning in Medical Image Analysis**. 2017. <https://arxiv.org/pdf/1702.05747.pdf>. Acessado em: 15/01/2020.

NECOS. **Motivation and Vision**. 2017. <http://www.h2020-necos.eu/motivation-and-vision/>. Acessado em: 04/12/2018.

_____. **D3.2: NECOS System Architecture and Platform Specification. V2**. 2019. <http://www.maps.upc.edu/public/necos_d3.2.v4.11_final_web.pdf>. Acessado em: 12/03/2019.

_____. **D5.1: Architectural update, Monitoring and Control Policies Frameworks**. 2019. <http://www.maps.upc.edu/public/D5.1_final.pdf>. Acessado em: 12/03/2019.

PASQUINI, R.; STADLER, R. Learning end-to-end application qos from openflow switch statistics. In: **2017 IEEE Conference on Network Softwarization (NetSoft)**. [S.l.: s.n.], 2017. p. 1–9.

PROMETHEUS. 2014. <https://prometheus.io/docs/operating/integrations/#remote-endpoints-and-storage>. Acessado em: 21/01/2020.

_____. 2014. <https://prometheus.io/docs/instrumenting/exporters/>. Acessado em: 21/01/2020.

_____. **Prometheus**. 2014. <https://prometheus.io/>. Acessado em: 12/12/2018.

QIAN, D. **Kafka exporter for Prometheus**. 2018. <https://github.com/danielqsj/kafka_exporter>. Acessado em: 21/01/2020.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. [S.l.]: Prentice-Hall, 1995.

SAR. **SAR**. 2018. <https://linux.die.net/man/1/sar>. Acessado em: 21/12/2018.

VEXXHOST. **OpenStack**. 2020. <https://www.openstack.org/bare-metal/>. Acessado em 18/01/2020.

_____. **OpenStack**. 2020. <https://www.openstack.org/software/>. Acessado em 22/07/2020.

VOLDEMORT. **Voldemort Project**. 2016. <http://www.project-voldemort.com/voldemort/>. Online; acessado em 20/03/2019.

YANG, K.; YOON, H.; SHAHABI, C. A supervised feature subset selection technique for multivariate time series. In: **In Proceedings of the Workshop on Feature Selection for Data Mining: Interfacing Machine Learning with Statistics, 92–101**. [S.l.: s.n.], 2005.

ZHU, X. **Semi-Supervised Learning Literature Survey**. 2005. <https://minds.wisconsin.edu/bitstream/handle/1793/60444/TR1530.pdf?sequence=1&isAllowed=y>. Acessado em: 15/01/2020.