Arcabouço de um Sistema Inteligente de Monitoramento para *Cloud Slices*

Gustavo S. Marques[1], Aryadne G. P. Rezende[1], Ian R. da Cunha[1], Raquel F. Q. Lafetá [1], Rafael Pasquini[1]

[1]Faculdade de Computação – Universidade Federal de Uberlândia (UFU)
Caixa Postal 593 – 38.400-902 – Uberlândia – MG – Brazil

{guto_silveira_, aryadne.guardieiro, ianresende}@ufu.br

{raquel.lafeta, rafael.pasquini}@ufu.br

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 investigates 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. An innitial experiment in our testbed is presented and some benefits of feature selection techniques can already be seen.*

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 estuda um sistema inteligente de monitoramento capaz de selecionar dinamicamente 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. Um primeiro experimento em nosso protótipo é apresentado e alguns dos benefícios da seleção de características já podem ser observados.*

1. Introdução

Nos últimos anos, computação em nuvem se tornou um assunto bastante relevante no meio tecnológico. Entre as suas contribuições pode-se destacar algumas, tais como: permitir processamentos pesados, maior capacidade de armazenamento de informação, acesso a partir de qualquer localidade e etc.

Empresas como Google, Microsoft, Amazon, IBM e outras, disponibilizam seus servidores (nuvem) oferecendo aos seus usuários uma variedade 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]. Estes serviços demonstram a versatilidade da computação em nuvem, a qual será mais ou menos aproveitada dependendo da necessidade do usuário. Alguns exemplos destes serviços são: hospedagem de sites, serviços de e-mail, serviços de armazenamento de dados e etc.

Entretanto, embora a computação em nuvem traga muitos benefícios, ela traz igualmente alguns desafios. Nesse contexto, pode-se levantar alguns questionamentos: 1) um único provedor de servidores de nuvem é sempre capaz de suprir todas as demandas de seus clientes? 2) Não sendo capaz, o que pode ser feito? 3) Seria possível estabelecer comunicação entre diversos provedores, de maneira que, um provedor possa contratar serviços de terceiros para ofertar ao seu cliente uma solução completa, de maneira transparente, quando não for capaz de o atender integralmente?

Nesse contexto, o projeto NECOS (*Novel Enablers for Cloud Slicing*) [Necos 2017c] propõe um novo paradigma definido como *slice-as-a-service*. Neste cenário, fatias (*slices*) são provisionadas entre provedores federados que fornecem diferentes tipos de infraestrutura, isto é, entre provedores de nuvem, de rede, etc. Portanto, uma *slice* é um conjunto de recursos de infraestrutura ofertado por vários provedores e que atende a necessidade de um usuário do sistema (aqui chamado de *Tenant*). Um *Tenant* pode ser entendido como um usuário final de serviços de nuvem, mas também como os próprios provedores e empresas.

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 2017a] [Necos 2017b]. 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*. Em interação com os 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.

Entretanto, dada a variedade de recursos que compõem as *slices*, o sistema de monitoramento pode sofrer problemas de escalabilidade, devido a grande quantidade de dados disponíveis na infraestrutura. Considerando a relevância da monitoria para o bom funcionamento da *slice*, se faz necessário pensar num sistema inteligente de monitoramento, capaz de não apenas coletar dados, mas também determinar um conjunto essencial de métricas, que realmente estão relacionadas ao bom funcionamento do serviço sendo ofertado. Idealmente, essa seleção precisa se dar de maneira automática, uma vez que esse conjunto pode ser diferente para cada serviço e para cada composição da *slice*.

Nesse contexto desafiador, o principal objetivo deste trabalho é investigar algoritmos de aprendizado de máquina, aplicados em conjunto com as técnicas de seleção de características, para a construção de um sistema inteligente de monitoramento. Feita essa seleção, há também o objetivo de evitar o tráfego de informações desnecessárias, o que significa que, uma vez selecionadas as métricas mais relevantes, é importante que as

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

demais deixem de ser trafegadas o mais próximo possível da sua fonte. Resultados iniciais coletados em nosso protótipo demonstram a eficácia da seleção automática de características, não apenas reduzindo o volume de métricas a serem coletadas, como também permitindo estimativas mais acuradas quanto ao serviço em execução.

O restante deste artigo está organizado da seguinte forma: a Seção 2 apresenta alguns trabalhos relacionados, nos quais também se fez necessária a seleção de características; a Seção 3 introduz a fundamentação teórica, resumindo conceitos de aprendizado de máquina e de seleção de características; a Seção 4 detalha o sistema inteligente de monitoramento em construção; a Seção 5 apresenta os resultados coletados em nosso primeiro experimento; e, por fim, a Seção 6 traz as conclusões e trabalhos futuros.

2. Revisão da Literatura Correlata

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 neste artigo.

No trabalho [Yang et al. 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). Em [Yang et al. 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. Isso também ocorre neste artigo, uma vez 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 de vídeo sob demanda, por exemplo, onde há mais de um servidor sendo utilizado para ofertá-lo, haverá uma série temporal multivariada, o que permite que o *Corona* seja testado, embora o cenário seja diferente.

No trabalho [Kira and A. 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. Portanto, o resultado obtido pelos autores pode servir de base de comparação para os que serão obtidos no trabalho aqui apresentado.

Em [John et al. 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, é válido tentar aplicar o método proposto em [John et al. 1994] para executar os algoritmos de seleção de características no sistema proposto neste trabalho.

Em [Pasquini and 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 (*Distributed hash table*). 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.

Futuramente, os dois serviços apresentados em [Pasquini and Stadler 2017] também serão avaliados no contexto deste trabalho, porém com duas importantes modificações: 1) no serviço de vídeo sob demanda, o protocolo DASH [Encoding.com 2016] será utilizado com intuito de analisar se os algoritmos de aprendizado continuam sendo capazes de construir bons conjuntos de métricas a partir das do serviço monitorado. Esse protocolo gera uma dificuldade a mais, pois altera dinamicamente a qualidade do vídeo de acordo com a carga aplicada ao servidor e a velocidade de banda do cliente. Portanto, suspeita-se que a predição de qualidade de serviço seja dificultada; e, 2) no serviço de DHT, será avaliado o serviço do Cassandra [Apache 2016] e não o Voldemort [Voldemort 2016].

3. Fundamentação Teórica

Esta Seção introduz algumas definições sobre os métodos de seleção de características, bem como conceitos sobre algoritmos de aprendizado de máquina.

Em [Chandrashekar and Sahin 2014] os autores apresentam uma revisão da literatura abordando os algoritmos reconhecidos como estado da arte de seleção de características. Além disso, é apresentada uma classificação das técnicas, dividindo-as segundo a estratégia de busca implementada para selecionar as características, que podem ser classificadas em: 1) *wrapper*, métodos que utilizam o mesmo algoritmo de aprendizado de máquina tanto no processamento/mineração dos dados quanto para seleção de características; 2) *filter*, métodos que primeiro constroem um *ranking* das características baseado em algum critério, para utilizá-lo posteriormente selecionando as características de acordo com a posição no *ranking*; 3) *embedded*, métodos que selecionam as melhores características a medida em que se cria o modelo de aprendizado.

Nesse sentido, todas as classes podem ser aplicadas. Conforme se vê em [Pasquini and Stadler 2017], tanto se pode utilizar os algoritmos de aprendizado de

máquina para a seleção de características, quanto se pode construir um modelo de seleção independente do algoritmo utilizado para o aprendizado. Sendo assim, qualquer uma das estratégias de busca podem ser exploradas. No contexto do trabalho aqui apresentado, explorou-se a estratégia *filter*, portanto, criou-se um *ranking* e em seguida selecionou-se as K primeiras características deste *ranking*.

No que tange a algoritmos de aprendizado de máquina, pode-se estabelecer a seguinte divisão: algoritmos de aprendizado não-supervisionado, de aprendizado semi-supervisionado e de aprendizado supervisionado.

No primeiro caso, são algoritmos aplicados quando não há qualquer informação sobre os dados coletados e se deseja aprender algo sobre eles [Fulmari1 and Chandak 2013]. Diz-se que não se tem os respectivos rótulos dos dados coletados. Um exemplo, seria um cenário em que se desejasse estabelecer relação entre pessoas com base nas suas características. Neste caso, poderia se aplicar um algoritmo de aprendizado não-supervisionado para agrupar (*clustering*) e detectar, por exemplo, que pessoas com determinado estilo de vida também são mais propensas a desenvolver determinada doença, sem que essa condição seja conhecida anteriormente.

No segundo caso, são algoritmos em que os dados em sua maioria não possuem os respectivos rótulos. Segundo [Zhu 2005], algoritmos deste tipo buscam construir classificadores. Ainda segundo [Zhu 2005], o fato de esses algoritmos trabalharem com dados não rotulados faz com que esta abordagem seja interessante em muitos casos, pois definir tais rótulos é uma tarefa frequentemente árdua.

No terceiro caso, 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 and 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. 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 este trabalho serão utilizados algoritmos de aprendizado supervisionado, buscando executar uma regressão nos dados, para que se possa estimar as métricas do serviço em questão. Esses algoritmos também poderão, eventualmente, ser utilizados para fazer a seleção de características em nossos experimentos. A técnica supervisionada se adéqua melhor ao cenário deste trabalho, pois as métricas coletadas possuem seus rótulos e estes são conhecidos do sistema de monitoramento. O protótipo desenvolvido é capaz de coletar métricas relativas ao funcionamento dos serviços, de tal forma que as coletadas na infraestrutura são automaticamente rotuladas. Esta é uma característica essencial do protótipo, pois, conforme dito anteriormente, obter rótulos é uma tarefa geralmente custosa.

4. Monitoramento Inteligente

O NECOS auxilia os *Tenants* a obterem suas *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 compõem as *slices*, ou até 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 da *slice*. O sistema também é responsável por realizar as alterações nas *slices*, visando manter os *Service Level Agreements* (SLAs) [Necos 2017a].

Desta forma, para possibilitar que tais funcionalidades existam, o NECOS possui um sistema de monitoramento contido no módulo IMA (*Infrastructure & Monitoring Abstraction*). Atualmente, o IMA é capaz de coletar métricas a partir dos diferentes provedores de infraestrutura presentes na federação, trabalhando com diferentes tecnologias e atuando em diferentes granularidades de coleta.

Neste projeto, o IMA é complementado com a capacidade de coletar dados dos serviços sendo ofertados nas *slices*. Os dados do serviço, conforme mencionado anteriormente, constituem uma alternativa para composição do conjunto Y de dados para o aprendizado supervisionado neste artigo. Além da coleta de Y relativo ao serviço, outra contribuição deste trabalho está em melhorar o atual sistema de coleta de dados do NECOS, adicionando uma seleção inteligente de características. O cenário ideal consiste em reduzir significativamente o conjunto de métricas coletadas por *slice*, mantendo a precisão das tarefas de gerenciamento.

A Figura 1 ilustra o fluxo de monitoramento 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*. De acordo com a requisição do *Tenant*, uma grande quantidade de tecnologias pode ser instanciada nos provedores para compor a *slice*. Isso implica que o IMA precisa se comunicar com diferentes provedores, que estabelecem padrões de comunicação diferentes uns dos outros, diferença essa que pode se dar tanto na forma de requisitar os dados, quanto na formatação da resposta. Portanto, o IMA implementa *Adaptors* (AD) para que seja possível comunicar-se com os

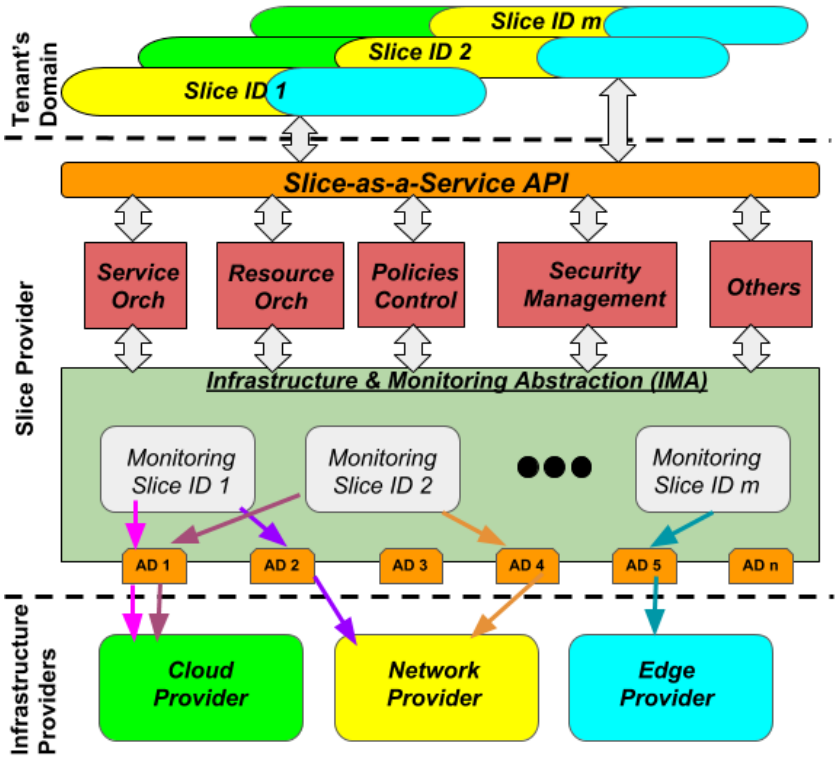

Figura 1. Visão geral do monitoramento no NECOS. O módulo IMA usa adaptadores para coletar métricas de diferentes tecnologias que os *Tenants* requisitam para construção das suas *slices*, instanciadas na infraestrutura federada de provedores. As métricas coletadas são fornecidas para as diferentes tarefas de gerenciamento desempenhadas durante o ciclo de vida das *slices*.

diversos provedores e, assim, coletar as métricas. Os *Adaptors* realizam a tradução da solicitação do IMA para uma sintaxe que o provedor aceite e vice-versa. Além disso, no IMA, instâncias dos módulos de monitoramento são geradas para cada *slice*.

Tão logo as métricas sejam coletadas da infraestrutura que compõe a *slice*, os módulos de gerenciamento, tais como o de Segurança, o de Controle de Políticas, o Orquestrador de Serviços e o de Recursos, etc, consomem essas informações durante o gerenciamento das *slices* para que decisões possam ser tomadas com base nelas. Por exemplo, após analisar os dados de monitoramento, o Orquestrador de Recursos pode concluir que determinada *slice* necessita de aumento no número de CPUs. Neste caso, o Orquestrador executa a ampliação do recurso da *slice*.

Finalizada a instanciação, 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/WAN Infrastruture Managers* (VIMs/WIMs) que fazem parte da *slice*. Usando tais pontos, o IMA é capaz de monitorar todas as métricas disponíveis.

Os módulos de gerenciamento indicam ao IMA qual Y a ser utilizado na 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/interesse. Além de fornecer o conjunto Y, os módulos podem especificar um valor K, que representa o tamanho do conjunto de métricas que deve ser produzido pela seleção de características. Existem técnicas de seleção nas quais o valor K não precisa ser informado.

A Figura 2 ilustra o Orquestrador de Recursos solicitando a seleção de características ao módulo de monitoramento (IMA). Neste exemplo, o Orquestrador de Recursos envia um conjunto Y ao módulo de monitoramento da *slice* (passo 1). Por sua vez, o módulo de monitoramento recupera todas as métricas $X_{Completo}$ de infraestrutura relativas à *slice* (passo 2). Na sequência, executa-se o algoritmo de seleção de características para construção de um conjunto reduzido de métricas $X_{Selecionado}$ (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.

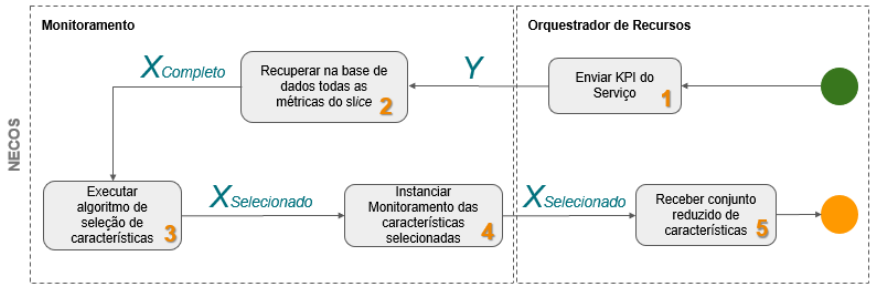

Figura 2. Fluxo proposto do sistema inteligente de monitoramento considerando solicitações vindas do módulo Orquestrador de Recursos.

Feita a seleção, o IMA passa a monitorar as métricas selecionadas mais frequentemente que as demais, o que significa que todas as métricas continuam a ser coletadas, porém em intervalos maiores. A frequência na coleta das métricas selecionadas $X_{Selecionado}$ respeita a solicitação do *Tenant*. A frequência de coleta para $X_{Completo}$ é um parâmetro a ser investigado neste trabalho. É preciso ter uma frequência de coleta que não prejudique a eficácia de seleção. Ao manter ambas as coletas, o presente trabalho permite que o conjunto $X_{Selecionado}$ possa ser renovado dinamicamente, o que pode ser necessário devido a mudanças as quais os serviços e *slices* estão sujeitos.

5. Avaliação

Os resultados apresentados nesta Seção foram coletados durante um experimento no qual um serviço de *Key-Value Store* foi monitorado. Para a construção do serviço, foram instanciadas 5 máquinas virtuais que hospedam um *cluster* do serviço Cassandra $v3.11.3$ [Apache 2016]. Além disso, outras duas máquinas virtuais complementam os experimentos: 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 Cassandra.

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, número de operações realizadas e tempos de resposta, incluindo, por exemplo, média e percentis dos tempos de resposta. Ao utilizarmos o gerador de carga, o objeto é conduzir o serviço do Cassandra através de diferentes níveis de carga. Como resultado, o protótipo desenvolvido consegue coletar as flutuações causadas nas métricas observadas no cliente que coleta Y, gerando um cenário ideal para rotular amostragens de X.

Para a coleta de dados da infraestrutura X utilizou-se uma ferramenta denominada *Prometheus* em conjunto com um *exporter* de dados [Kochie 2018], que foi instalado em cada uma das 5 máquinas que hospedam o serviço do Cassandra. Isso permitiu que métricas sobre os mais diversos aspectos fossem coletadas, incluindo métricas de uso de CPU, memória, tráfego em *bytes* na rede, etc. Para este experimento[2], o intervalo de coleta de tais métricas, tanto as do conjunto X quanto as do conjunto Y, foi de um segundo. Relativamente ao conjunto X, 374 métricas foram coletadas, desta forma, temos que o conjunto $X_{Completo}$ é formado por todas estas métricas coletadas a partir das 5 máquinas onde o serviço do Cassandra é executado.

Toda a infraestrutura é virtualizada no OpenStack. Cada uma das 5 máquinas virtuais com o Cassandra que estão envolvidas no teste possuem a seguinte configuração: 1 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 Ubuntu Server 16.04. Os mecanismos de aprendizado de máquina para seleção de características foram executadas em um *notebook* com processador Intel i7 de sétima geração e 16GB de memória RAM, num ambiente com sistema operacional Ubuntu 18.04 LTS.

O experimento para a coleta dos conjuntos $X_{Completo}$ e Y teve duração total de três horas. O gerador de carga, implementado utilizando a linguagem Python, segue uma distribuição de *Poisson* para a chegada de novos clientes, aplicando uma sinusoide para determinar λ. Iniciou-se o teste com $\lambda = 11$ clientes ativos (10 criados pelo gerador de carga, mais o cliente para coleta de Y). A sinusoide possui período de uma hora e amplitude de 5 clientes, ou seja, o número de clientes ativos durante o período de uma hora oscila entre 16 e 6 clientes. Desta forma, o experimento coletou dados referentes a três sinusoides completas.

A Figura 3 apresenta o 95 percentil do tempo de escrita na DHT observado no cliente onde o Y é coletado, durante o experimento de três horas. É possível notar as oscilações causadas no tempo de resposta, reflexo do número de clientes ativos que foram controlados pelo gerador de carga.

O objetivo central deste artigo é avaliar a eficácia da seleção de características, aplicando diferentes valores de K para a composição do conjunto $X_{Selecionado}$. Porém, para permitir uma melhor ilustração dos resultados da seleção de características, os experimentos aplicaram também um regressor para estimar o 95 percentil dos tempos de resposta das operações de escrita efetuadas pelo cliente coletor de Y. Este seria, por exemplo, a tarefa de um módulo orquestrador que precisa avaliar se acordos de nível de

[2]Os dados do experimento podem ser acessados em: *https://github.com/gustavosm/traces-wslice-2019*

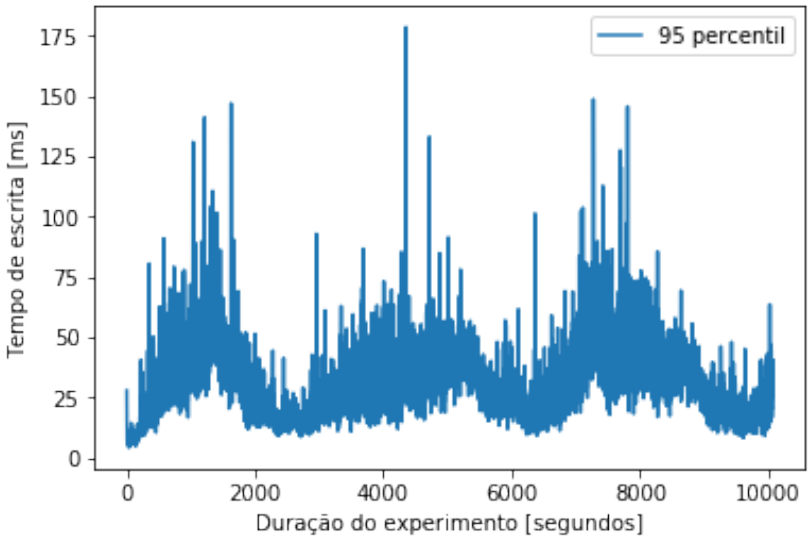

Figura 3. 95 percentil do tempo de escrita na DHT ao longo de três horas de experimentação.

serviço estão sendo honrados na *slice*.

O algoritmo de aprendizado de máquina utilizado para a regressão foi o *Random Forest*, utilizando-se a implementação oferecida no pacote de *machine learning* para Python denominado *scikit-learn* [INRIA 2018]. Os parâmetros de configuração foram: *max_depth* $= 10$, *random_state* $= 17$ e *n_estimators* $= 120$. Nesta avaliação, utilizou-se 70% dos dados coletados para o treinamento do regressor e os outros 30% para avaliar a acurácia deste. A acurácia é apresentada em termos do Erro Absoluto Médio Normalizado (*Normalized Mean Absolute Error - ($NMAE$)*), que é expressada por: $\frac{1}{\overline{y}}(\frac{1}{m}\sum_{i=1}^{m}|y_i - \hat{y}_i|)$, onde $\hat{y}_i$ é estimativa obtida pelo regressor, y_i é o valor coletado no cliente, e $\overline{y}$ é a média dos valores y_i do conjunto de teste, cujo tamanho é igual a m.

A Figura 4 apresenta duas séries de dados referentes ao valor amostrado de Y (conjunto de teste com 30% das amostras) pelo cliente e o valor estimado $\hat{Y}$ pelo regressor, a partir das métricas $X_{Completo}$ observadas na infraestrutura onde o serviço Cassandra é executado. A taxa de erro das estimativas feitas pelo regressor corresponde a $NMAE = 20,07\%$. Conforme ilustra a Figura 3, as estimativas conseguem seguir o comportamento dos valores amostrados, ou seja, a partir das métricas $X_{Completo}$, é possível estimar o comportamento do serviço Cassandra que está sendo entregue aos clientes.

Conforme mencionado anteriormente, o conjunto $X_{Completo}$ contém todas as métricas da infraestrutura, coletadas pelo *Prometheus*. O tamanho deste conjunto de dados varia conforme a composição da infraestrutura, podendo representar um volume pesado de dados a ser coletado. Além disso, o conjunto $X_{Completo}$, geralmente, possui métricas (ruídos) que podem degradar o comportamento, por exemplo, dos regressores. Outro fator importante é que, para cada serviço diferente ofertado em uma *slice*, a composição ideal de X para representar o comportamento do serviço pode variar.

A Figura 5 apresenta o primeiro resultado do mecanismo inteligente de seleção de características desenvolvido neste trabalho. Os resultados apresentam a $NMAE$ obtida

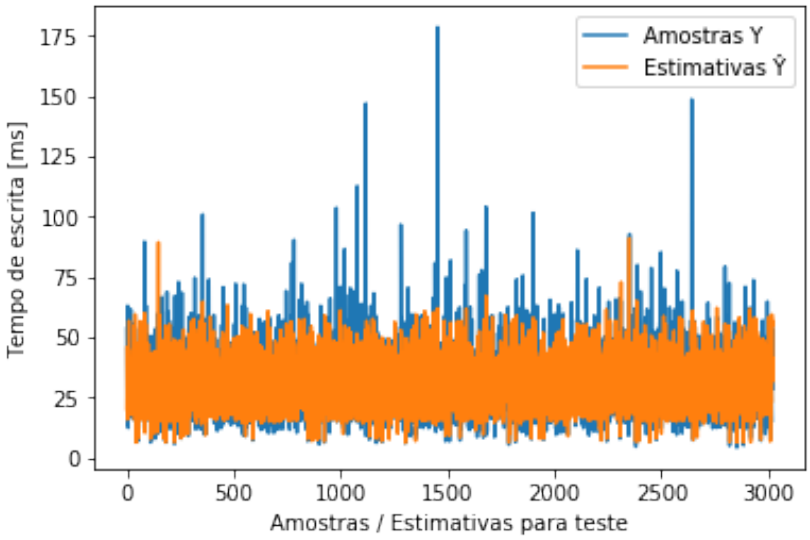

Figura 4. Valores amostrados de Y pelo cliente do Cassandra e estimativas $\hat{Y}$ obtidas pelo regressor *Random Forest*.

para regressores treinados com diferentes conjuntos $X_{Selecionado}$, considerando os valores $K = \{50, 100, 150, 200\}$ para a seleção. A estratégia escolhida para seleção de características foi a *filter* e, portanto, as K métricas representam as primeiras posições de um *ranking* que elege as métricas mais correlacionadas com o Y considerado.

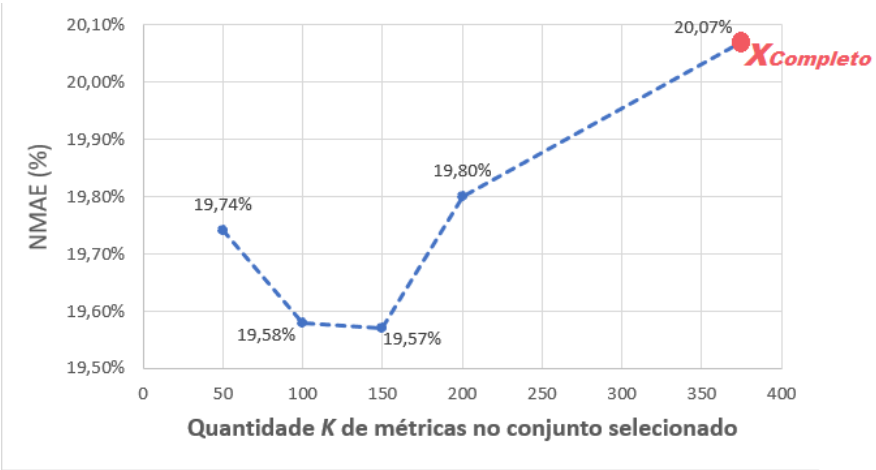

Figura 5. $NMAE$ observada para os diferentes valores de K utilizados para seleção de métricas que compõem o conjunto $X_{Selecionado}$.

Como pode ser observado na Figura 5, os diferentes conjuntos $X_{Selecionado}$ são capazes de melhorar a acurácia do regressor, apresentando melhores valores de $NMAE$. O gráfico indica que, neste cenário de experimentação, valores para K entre 100 e 150 são os mais indicados. Primeiramente, este resultado indica a eficácia da seleção de características, permitindo que um conjunto menor de métricas sejam coletadas a partir da infraestrutura, evitando a movimentação desnecessária de grandes volumes de dados.

Outro aspecto importante do resultado da Figura 5 refere-se a extração de ruído

presente no conjunto $X_{Completo}$. Vale destacar que o valor $K = 50$ apresentou $NMAE$ maior do que $K = 100$ e, conforme o *ranking*, as primeiras 50 métricas são as mesmas em ambos os conjuntos $X_{Selecionado}$. Entretanto, além de eliminar ruído, o valor adequado para K deve ser capaz de produzir um conjunto $X_{Selecionado}$ que represente adequadamente o comportamento de Y.

Finalmente, a redução no número de métricas permite o treinamento de regressores em um menor tempo. Por exemplo, considerando o conjunto $X_{Selecionado}$ para $K = 150$, o regressor do *random forest* foi treinado em 27.73 segundos, enquanto que para o conjunto $X_{Selecionado}$ com $K = 100$ este tempo foi de 15.51 segundos.

6. Conclusões e trabalhos futuros

O cenário de investigação do projeto NECOS é desafiador. Por se tratar de um sistema que deve ser altamente escalável, 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, para que possibilite aos demais módulos maior velocidade na tomada de decisões. 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. Entretanto, apesar de o cenário ser bastante desafiador, os primeiros testes são promissores e indicam a importância do sistema inteligente de monitoramento apresentado neste trabalho.

O conjunto de trabalhos futuros inclui: 1) experimentar outros serviços conforme indicado na Seção 2; 2) avaliar coletas de métricas $X_{Completo}$ em intervalos diferentes do atual (um segundo); 3) aplicar diferentes padrões de carga aos serviços, além do atual baseado na sinusoide; 4) utilizar outras ferramentas de coleta de dados, tais como *Ceilometer* [Ceilometer 2018] e *SAR* [SAR 2018]; e 5) aplicar outras metodologias de seleção de características, que não a *filter*, conforme discutido na Seção 3.

Agradecimentos

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.

Referências

Apache (2016). Apache cassandra. `http://cassandra.apache.org/`. Acessado em: 17/03/2019.

Ceilometer (2018). Ceilometer. `https://docs.openstack.org/ceilometer/latest/`. Acessado em: 21/12/2018.

Chandrashekar, G. and Sahin, F. (2014). A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16 – 28. 40th-year commemorative issue.

Encoding.com (2016). Mpeg-dash an overview. `https://www.encoding.com/mpeg-dash/`. Acessado em: 17/03/2019.

Fulmari1, A. and Chandak, M. B. (2013). A survey on supervised learning for word sense disambiguation. https://pdfs.semanticscholar.org/58bb/8f4b9a0e7257ca15555e505e9fd35992f66c.pdf. Acessado em: 17/03/2019.

INRIA (2018). Scikit learn. https://scikit-learn.org/stable/. Acessado em: 12/12/2018.

John, G. H., Kohavi, R., and Pfleger, K. (1994). Irrelevant features and the subset selection problem. In *ICML 1994*.

Kira, K. and A. Rendell, L. (1992). The feature selection problem: Traditional methods and a new algorithm. In *The Feature Selection Problem: Traditional Methods and a New Algorithm.*, pages 129–134.

Kochie, B. (2018). Prometheus node-exporter. https://github.com/prometheus/node_exporter/releases. Acessado em: 06/02/2019.

Kächele, S., Spann, C., Hauck, F. J., and Domaschka, J. (2013). 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*, pages 75–82.

Necos (2017a). D3.1: Necos system architecture and platform specification. v1. http://www.maps.upc.edu/public/NECOS%20D3.1%20final.pdf. Acessado em: 12/03/2019.

Necos (2017b). D5.1: Architectural update, monitoring and control policies frameworks. http://www.maps.upc.edu/public/D5.1_final.pdf. Acessado em: 12/03/2019.

Necos (2017c). Motivation and vision. http://www.h2020-necos.eu/motivation-and-vision/. Acessado em: 04/12/2018.

Pasquini, R. and Stadler, R. (2017). Learning end-to-end application qos from openflow switch statistics. In *2017 IEEE Conference on Network Softwarization (NetSoft)*, pages 1–9.

SAR (2018). Sar. https://linux.die.net/man/1/sar. Acessado em: 21/12/2018.

Voldemort (2016). Voldemort Project. http://www.project-voldemort.com/voldemort/. Online; acessado em 20/03/2019.

Yang, K., Yoon, H., and Shahabi, C. (2005). 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*.

Zhu, X. (2005). Semi-supervised learning literature survey. https://minds.wisconsin.edu/bitstream/handle/1793/60444/TR1530.pdf?sequence=1. Acessado em: 17/03/2019.