Pós-Graduação em Ciência da Computação

Wellison Raul Mariz Santos

Adaptação de aplicações baseadas em microsserviços usando aprendizagem de máquina

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

Recife
2020

Wellison Raul Mariz Santos

Adaptação de aplicações baseadas em microsserviços usando aprendizagem de máquina

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: *Redes de Computadores e Sistemas Distribuídos*
Orientador: *Nelson Souto Rosa*
Co-Orientador: *George Darmiton da Cunha Cavalcanti*

Recife
2020

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

S237a Santos, Wellison Raul Mariz
Adaptação de aplicações baseadas em microsserviços usando aprendizagem de máquina / Wellison Raul Mariz Santos. – 2020.
89 f.: il., fig., tab.

Orientador: Nelson Souto Rosa.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2020.
Inclui referências e apêndices.

1. Redes de computadores. 2. Aprendizagem de máquina. I. Rosa, Nelson Souto (orientador). II. Título.

004.6 CDD (23. ed.) UFPE - CCEN 2020 - 76

Wellison Raul Mariz Santos

“Adaptação de aplicações baseadas em microsserviços usando aprendizagem de máquina”

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 27 de fevereiro de 2020.

BANCA EXAMINADORA

_____

Prof. Dr. Vinícius Cardoso Garcia
Centro de Informática / UFPE

_____

Prof. Dr. Fernando Antônio Aires Lins
Departamento de Estatística e Informática/UFRPE

_____

Prof. Dr. Nelson Souto Rosa
Centro de Informática / UFPE
(Orientador)

Eu dedico esse trabalho a minha mãe, que sempre me incentivou, encorajou e nunca mediu esforços para me ajudar independentemente da situação.

AGRADECIMENTOS

Agradeço a minha família, meu pai **Paulo**, minha mãe **Maria de Fátima**, meu irmão **Wesley**, por todo o apoio durante esses anos e em especial ao meu irmão Augusto, pela companhia e parceria durante os anos dessa pesquisa.

À **Mayane** minha noiva, que pacientemente me incentivou todos os dias os quais estamos juntos e mostrou-se ser companheira para as horas fáceis e difíceis da vida. Obrigado pelo carinho, a paciência e por sua capacidade de me trazer paz na correria de cada semestre.

A meu orientador, **Dr. Nelson Souto Rosa**, pela confiança, disponibilidade, compreensão, apoio e pelas riquíssimas discussões que direcionaram da melhor forma possível o desenvolvimento deste trabalho e compartilhamento de seus conhecimentos num ambiente ético, profissional e acima de tudo humano, sempre visando o crescimento.

A meu co-orientador, **Dr. George Darmiton da Cunha Cavalcanti**, por sua excelente orientação, paciência, disposição, assistência e apoio.

Aos amigos pesquisadores do Centro de Informática aos quais destaco **Eraylson Galdino**, **Josival Santos**, **Amarildo Lucena**. Tenho certeza de que a qualidade deste trabalho não seria a mesma sem a ajuda de vocês. Grato.

Agradeço aos meus amigos do laboratório de pesquisa: **Bruno**, **Francisco**, **Flávio**, **Gustavo**, **Gunnar**, **Leandro**, **Milton**, **Osmar**, **Rafael**, **Reinaldo** e **Vandenberg**, que por muitas vezes me ajudaram durante todo o meu mestrado e tornaram o meu dia a dia mais divertido.

À **Fundação de Amparo à Ciência e Tecnologia do Estado de Pernambuco** (FACEPE), que me promoveu recursos financeiros para execução deste trabalho.

A todos participantes do grupo de pesquisa, agradeço a todos pela contribuição direta ou indireta. Minha gratidão a todos colegas do grupo do **GFADS**.

Agradeço aos **Professores do Centro de Informática da UFPE**, por todo o conhecimento compartilhando, que foram fundamentais para minha formação como Mestre.

À **UFPE** pela infraestrutura para a realização do trabalho.

RESUMO

Os microsserviços têm sido amplamente utilizados para criar aplicações complexas baseadas na nuvem. Nos sistemas baseados em nuvem, um dos requisitos essenciais é a escalabilidade. Sistemas escaláveis se adaptam às mudanças na carga de trabalho usando recursos de hardware adicionais, ou seja, dimensionamento automático. No entanto, o gerenciamento de recursos envolve o monitoramento da carga de trabalho para determinar quando as adaptações são necessárias, seguida de uma ação corretiva rápida e precisa para evitar problemas como indisponibilidade ou baixo desempenho da aplicação. Como consequência, essas aplicações devem ser gerenciadas continuamente para que o ajuste de seus recursos atenda à demanda atual. Dessa maneira, as aplicações buscam manter a qualidade do serviço fornecido. As soluções existentes descrevem vários mecanismos que aplicam técnicas de dimensionamento automático para adaptar *Microservice-Based Applications* (MBAs). Porém, a adaptação somente é efetuada após um problema ocorrer, *e.g.,* a adição de recursos adicionais a um microsserviço operando com consumo de CPU em 95%, quando deveria usar no máximo 80%. No entanto, poucos trabalhos tratam o gerenciamento de recursos que ajusta o sistema antes que o problema ocorra e as consequências de sua adoção. Nesse contexto, este trabalho propõe um novo ambiente proativo que usa modelos de aprendizagem de máquina para prever cargas de trabalho no futuro próximo, e os usa para ajudar nas decisões de provisionamento de recursos. Esses modelos foram integrados a um *feedback loop* MAPE-K, responsável pela adaptação da MBA. Essa pesquisa serve ainda como um passo-a-passo de como usar modelos de aprendizado de máquina para o desenvolvimento de soluções adaptativas proativas no mundo de microsserviços. Para avaliar o ambiente proposto, experimentos foram realizados para compará-la com os mecanismos reativos existentes no Kubernetes. No final, o ambiente proposto mostrou um melhor desempenho para adaptar MBAs.

Palavras-chave: Microsserviços. Aprendizagem de Máquina. Previsão de Recursos.

ABSTRACT

Microservices have been used extensively to create complex cloud-based applications. In cloud-based systems, one of the essential requirements is scalability. Scalable systems adapt to changes in workload using additional hardware resources, i.e., autoscaling. However, resource management involves monitoring the workload to determine when adaptations are needed, followed by quick and accurate corrective action to avoid problems such as unavailability or poor application performance. As a consequence, these applications must be managed continuously so that the adjustment of their resources meets the current demand. In this way, applications seek to maintain the quality of the service provided. Existing solutions describe various mechanisms that apply autoscaling techniques to adapt the Microservice-Based Applications (MBAs). However, the adaptation only is made after a problem occurs, e.g., the addition of additional resources to a microservice operating with 95% CPU consumption, when it should use a maximum of 80%. However, few of them address the use of resource management to adjust the system before the problem occurs and the consequences of their adoption. In this context, this work proposes a new proactive environment that uses machine learning models to predict near-future workloads and uses them to help in resource provisioning decisions. These models were integrated into a MAPE-K feedback loop, responsible for adapting the MBA. This research also serves as a step-by-step guide on how to use machine learning models for the development of proactive adaptive solutions in the world of microservices. The proposed environment was evaluated through experiments that compared it to the existing reactive mechanisms in Kubernetes. In the end, the proposed environment showed the better performance to adapt MBAs.

Keywords: Microservices. Machine Learning. Forecasting.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

AM	Aprendizagem de Máquina
API	*Application Programming Interface*
ARFIMA	Autoregressive Fractionally Integrated Moving Average
ARIMA	Autoregressive Integrated Moving Average
ES	Exponential Smoothing
FCC	Ferramenta de *Clustering* para Contêineres
GARCH	Generalized Autoregressive Conditional Heteroskedasticity
HPA	Horizontal Pod Autoscaler
LR	Linear Regression
LSTM	Long Short-Term Memory
MBA	*Microservice-Based Application*
MLP	Multilayer Perceptron
MLR	Multinomial Logistic Regression
QoS	*Quality of Service*
RBF	Radial Basis Function
RF	Random Forest
RMSE	Root Mean Square Error
RNA	Rede Neural Artificial
RTT	Round Trip Time
SARIMA	Seasonal Autoregressive Integrated Moving Average
SSA	Singular Spectrum Analysis
SVM	Support Vector Machine
SVR	Support Vector Regression
VM	*Virtual Machine*
XGBoost	Extreme Gradient Boosting

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

Um dos recursos essenciais de uma aplicação em nuvem é a escalabilidade. Essas aplicações se adaptam às mudanças em sua carga de trabalho usando recursos de hardware adicionais para manter seu desempenho em um nível adequado (Herbst *et al.*, 2013). Relacionado a esse fato, os microsserviços receberam muita atenção nos últimos anos devido a sua adequação na construção de sistemas altamente escaláveis e flexíveis. Eles oferecem benefícios significativos ao desenvolvimento, em oposição à abordagem monolítica padrão (Taibi *et al.*, 2017). Como consequência, a adoção de microsserviços está começando a se tornar uma tendência no desenvolvimento de aplicações em nuvem. Empresas como Amazon, Netflix, LinkedIn, SoundCloud e Uber já estão desenvolvendo soluções baseadas no uso de microsserviços (Hassan *et al.*, 2017; Taibi *et al.*, 2017).

O estilo arquitetural dos microsserviços é uma abordagem para desenvolver aplicações como um conjunto de serviços autônomos. Cada serviço executa seu processo e interage com outros microsserviços através de interfaces padronizadas (Lewis & Fowler, 2014; Hassan & Bahsoon, 2016). *Microservice-Based Applications* (MBAs) são sistemas distribuídos cujos componentes são altamente dinâmicos no sentido de que os microsserviços podem ser atualizados, removidos, adicionados, replicados enquanto o sistema é executado. Essa estratégia de desenvolvimento simplifica o dimensionamento da aplicação através de operações de *scaling-in/out* na quantidade de réplicas, delimita as responsabilidades de cada componente e permite a implantação independente de microsserviços (Taibi *et al.*, 2017).

MBAs admitem alterações no contexto de execução da aplicação, alterações nas condições ambientais, *e.g.,* variação da carga de trabalho ou qualidade da rede (Toffetti *et al.*, 2015). Adaptações frequentes podem causar violações dos requisitos especificados em tempo do projeto (Toffetti *et al.*, 2015). Uma maneira de identificar violações é monitorar os atributos de qualidade das MBAs (*e.g.,* disponibilidade e desempenho) e verificar os níveis desejados. Se uma violação for detectada, as ações corretivas poderão ser aplicadas automaticamente, com mínima intervenção humana, para manter a aplicação funcionando corretamente (Salehie & Tahvildari, 2009).

Existem vários tipos de pesquisa sobre gerenciamento de MBAs. As estratégias existentes incluem otimizar a localização dos microsserviços implantados (Gabbrielli *et al.*, 2016; Sampaio *et al.*, 2019), o controle de atualizações efetuadas em produção (Rajagopalan & Jamjoom, 2015), melhorias nos componentes de comunicação (Kookarinrat & Temtanapat, 2016) e gerenciamento do consumo de recursos das aplicações (Florio & Nitto, 2016; Toffetti *et al.*, 2017; Zhang *et al.*, 2018). Essas pesquisas são consideradas reativas por suas soluções efetuarem intervenções na MBA apenas após a ocorrência do problema. Florio & Nitto (2016), por exemplo, observam se a demanda por recursos nos microsserviços está sendo violada, *e.g.,* um microsserviço operando com consumo de CPU em 90%, quando o limite pré-definido é 80%. Desta forma, após a detecção da violação, uma nova réplica do microsserviço é implantada para suprir a demanda.

1.1 MOTIVAÇÃO E JUSTIFICATIVA

Embora as soluções existentes tenham abordado o gerenciamento de recursos usados pelas MBAs, pouca atenção foi dada à criação de soluções proativas. Uma solução proativa age antes que ocorra um problema na execução de uma MBA, *e.g.,* antes do microsserviço ficar sobrecarregado, a solução cria uma réplica. No entanto, de acordo com o que foi identificado nos trabalhos observados na seção anterior, a maioria das soluções são reativas, pois só agem quando um problema já ocorreu.

O tempo entre detectar um problema e corrigi-lo pode levar a uma deterioração da qualidade do serviço ou até sua própria interrupção. Portanto, soluções proativas podem ser mais eficazes em determinadas situações, em oposição às reativas (Salehie & Tahvildari, 2009). Como as soluções proativas podem antecipar possíveis violações, elas podem propor rapidamente uma modificação que lide com os problemas. Por exemplo, a proatividade beneficia as MBAs, permitindo adaptações antes de uma mudança repentina na carga de trabalho recebida.

Trabalhos anteriores foram propostos para lidar proativamente com o gerenciamento de recursos de MBAs. Essas soluções aplicam algoritmos de Aprendizagem de Máquina (AM) para prever a demanda de recursos futura (Alipour & Liu, 2017; Wong, 2018). No entanto, as abordagens existentes não demonstram se o uso de uma solução proativa para microsserviços melhora ou não o processo de adaptação.

Alipour & Liu (2017), por exemplo, não comparam a sua solução proativa à nenhuma outra existente. Wong (2018), por outro lado, apresenta a previsão de recursos por meio de modelos de AM, como uma alternativa para melhorar um escalonador híbrido de microsserviços nomeado de HyScale. No entanto, os modelos de AM estão fora do processo adaptativo do HyScale. Logo, Wong (2018) apresenta apenas uma solução adaptativa reativa que altera as MBAs em tempo de execução.

Outra característica desses trabalhos são os algoritmos de AM escolhidos. Alipour & Liu (2017) utilizam os algoritmos Multinomial Logistic Regression (MLR) e Linear Regression (LR), enquanto Wong (2018) aplica o Long Short-Term Memory (LSTM). Porém, na literatura os algoritmos comuns para previsão de informações (*e.g.,* consumo de CPU, tempo de resposta, taxa de transferência) são o Multilayer Perceptron (MLP) e o Support Vector Regression (SVR) (Ajila & Bankole, 2016; Baldan *et al.*, 2018; Nikravesh *et al.*, 2017).

Diante disto, surge a necessidade de investigar como realizar a verificação e adaptação de MBAs de modo proativo. Esse processo de investigação deve considerar a análise do desempenho da solução proativa quando comparada a uma outra reativa. Além disso, assim como os trabalhos anteriores propostos, essa dissertação deve optar por algoritmos de AM como um recurso para alcançar a proatividade na solução. Porém, considerando algoritmos comuns de previsão.

1.2 OBJETIVOS

O objetivo principal é investigar a viabilidade da aplicação de algoritmos de AM em soluções adaptativas proativas que gerenciam os recursos de MBAs. Para isto, será proposto um ambiente que deve monitorar e identificar violações aos limites pré-definidos para o consumo de recursos e agir para corrigi-los em tempo de execução, com o mínimo de intervenção humana.

A fim de atingir o objetivo principal, os seguintes objetivos específicos foram definidos:

1. Realizar um estudo sobre a adoção de ambientes adaptativos na arquitetura de microsserviços;
2. Revisar a literatura de AM para identificar os principais algoritmos que podem ser aplicados dentro do contexto de microsserviços;
3. Definir uma arquitetura conceitual do ambiente baseado em um *feedback loop*;
4. Implementar o ambiente e os cenários de simulação;
5. Validar o ambiente através de experimentos.

1.3 CONTRIBUIÇÃO

A contribuição principal desta dissertação é o desenvolvimento de um ambiente holístico genérico capaz de gerenciar o consumo de CPU de MBAs. Para isto, é proposto o ML-Adapt (***M**achine **L**earning **Adapt**ation of microservice-based applications*), um ambiente adaptativo proativo que usa algoritmos para a aprendizagem de padrões e produz previsões, em tempo de execução, com base no consumo de CPU da MBA. Para esse fim, ele monitora constantemente o consumo de CPU dos contêineres implantados. Nos cenários avaliados nesta dissertação, um contêiner comporta apenas um único microsserviço. Logo, a previsão do consumo dos contêineres equivale ao consumo da MBA.

O ML-Adapt é construído sobre o *feedback loop* MAPE-K (***M**onitor*, ***A**nalyser*, ***P**lanner*, ***E**xecutor* e o ***K**nowledge*). Ele usa a previsão de demanda do consumo de CPU para decidir se uma adaptação é necessária ou não. Se necessária, a adaptação consiste em executar operações de *scaling-in/out* nos microsserviços da MBA. Por consequência, nos cenários avaliados neste trabalho, ele consegue manter a *Quality of Service* (QoS) provida e a eficiência no uso de recursos, mesmo em situações de alteração da demanda.

Além da contribuição principal, algumas contribuições secundárias deste trabalho incluem:

1. Um conjunto abrangente de etapas e tecnologias úteis para a construção de soluções adaptativas proativas por meio de algoritmos de AM no contexto de MBAs;

2. O desenvolvimento de sensores e atuadores capazes de estabelecer comunicação com diferentes tecnologias. Os sensores, por exemplo, são capazes de coletar, minerar e processar métricas de desempenho, em tempo de execução, e disponibilizá-las para o ML-Adapt. Os atuadores podem alterar a quantidade de réplicas implantadas por microsserviço.

1.4 ORGANIZAÇÃO DO TRABALHO

Os próximos capítulos estão organizados como segue:

- Capítulo 2 introduz os conceitos básicos necessários ao entendimento deste trabalho;
- Capítulo 3 apresenta a proposta desse trabalho com foco no projeto e implementação;
- Capítulo 4 demonstra os experimentos realizados para a avaliação de desempenho do ML-Adapt;
- Capítulo 5 expõe os trabalhos relacionados e faz uma análise comparativa com o que foi proposto nesta dissertação;
- Capítulo 6 descreve as principais contribuições, limitações e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os conceitos necessários ao entendimento deste trabalho. A Seção 2.1 introduz conceitos básicos sobre softwares adaptativos, com ênfase na arquitetura MAPE-K. A Seção 2 apresenta uma introdução a microsserviços, o processo de implantação no contexto das *Virtual Machines* (VMs) e dos contêineres e discute as diferentes Ferramentas de *Clustering* para Contêineres (FCCs) existentes. Por fim, a Seção 2.3 apresenta conceitos de séries temporais, um conjunto de etapas referentes ao processo de previsão e os algoritmos de AM.

2.1 SOFTWARES ADAPTATIVOS

Os sistemas computacionais alcançaram um nível de complexidade que o esforço humano necessário para colocá-los e mantê-los em funcionamento é custoso (Huebscher & McCann, 2008). Dessa forma, se torna necessário que processos sejam propostos para auxiliar o gerenciamento desses sistemas. Kephart & Chess (2003) destacam que a computação autonômica é a única opção para tornar as aplicações mais gerenciáveis. A computação autonômica busca melhorar os sistemas computacionais, tornando-os adaptativos, com o objetivo de diminuir o envolvimento humano (Huebscher & McCann, 2008).

A adaptação é um processo pela qual um organismo recebe alterações em sua estrutura ou funcionamento que o torna mais adequado ao ambiente (Da *et al.*, 2011). Logo, sistemas adaptativos são capazes de se adaptar às mudanças que ocorrem no ambiente de execução para continuar a atingir seus objetivos (Weyns *et al.*, 2013). Segundo Salehie & Tahvildari (2009), essas mudanças são derivadas, principalmente, da complexidade de gerenciar os sistemas (*e.g.,* heterogeneidade entre componentes que compõem uma aplicação), robustez no tratamento de condições inesperadas (*e.g.,* falhas), e a busca por resiliência para lidar com alterações constantes nos objetivos/requisitos, *e.g.,* o tempo de resposta da aplicação deve ser no máximo 200ms.

Huebscher & McCann (2008) e Kephart & Chess (2003) classificam as soluções adaptativas em quatro principais grupos, observando suas propriedades de autogerenciamento:

- A autoconfiguração permite que um sistema adaptativo seja capaz de configurar-se conforme objetivos de alto nível definidos. No entanto, isso não significa que a solução está ciente de como executar reconfigurações, apenas que ela conhece o que é desejado. Por exemplo, quando um componente é introduzido, ele se incorpora perfeitamente, enquanto o resto do sistema se adapta à sua presença, de modo que, ao expor suas capacidades, os outros componentes possam usá-las.

- A auto-otimização permite que um sistema busque continuamente oportunidades para melhorar seu desempenho e eficiência. Isso significa que este detecta mudanças no seu ambiente (*e.g.,* uma nova versão de um componente) e se modifica para otimizar as funções providas. Por exemplo, quando um sistema detecta variações na demanda

de recursos de um determinado componente, ele é capaz de alocar réplicas, com a mesma função, a fim de suprir a demanda solicitada.

- A autocura proporciona à uma solução adaptativa a capacidade de detectar e diagnosticar problemas. Essas soluções podem, portanto, detectar diversos problemas, como falha em hardware (*e.g.,* um disco sólido redundante parou de funcionar), ou software, *e.g.,* quando um determinado serviço não está respondendo. Porém, além de resolver o problema, é importante que essa ação não acarrete a ocorrência de outros problemas.

- A autoproteção possibilita que um sistema autônomo se proteja de ataques maliciosos ou evite a ocorrência de falhas em cascatas. Por exemplo, ele garante que um determinado usuário não possa remover um arquivo essencial para o funcionamento do sistema. Outro aspecto dessa propriedade é a capacidade de determinar, com base em relatórios anteriores, futuras violações que prejudiquem o sistema.

De acordo com Salehie & Tahvildari (2009), uma maneira de identificar os requisitos de uma aplicação auto adaptável é através do uso de seis perguntas:

1. Onde a adaptação ocorre? Esta pergunta está associada à localização do problema que precisa ser resolvido pela adaptação, identificando quais artefatos e em que camada estes se encontram, *e.g.,* ocorre após uma mudança repentina do consumo de recursos por um microsserviço.

2. Quando a adaptação ocorre? Esta pergunta lida com os aspectos temporais da solução adaptativa, abordando pontos como: quando uma mudança precisa ser aplicada, se a mudança é aplicada a qualquer momento, com que frequência essa mudança ocorre e se as ações adaptativas realizadas são reativas ou proativas, *e.g.,* uma ação corretiva é aplicada ao perceber uma degradação de desempenho do microsserviço.

3. O que é adaptado? Essa pergunta identifica os atributos que podem ser alterados por meio de ações adaptativas e o que precisa ser alterado em cada situação, *e.g.,* o elemento alterado é o microsserviço que recebe recursos adicionais.

4. Por que é adaptado? Esse ponto determina as motivações da construção de uma aplicação de software adaptável, *e.g.,* um microsserviço gerenciado é capaz de operar com um tempo de resposta menor.

5. Como é a adaptação? Essa pergunta define como a solução adaptativa altera o sistema e quais ações são aplicadas conforme as condições encontradas, *e.g.,* os microsserviços são adaptados por meio de técnicas de dimensionamento automático que são capazes de aumentar e diminuir a quantidade de réplicas.

6. Quem adapta? Esse item aborda o nível de automação e envolvimento humano necessários para a solução adaptativa, *e.g.,* o gerente humano precisa intervir no número máximo de réplicas por microsserviço, quando novos equipamentos de hardware estão disponíveis.

2.1.1 Feedback Loop

Feedback Loop é um elemento crucial para a desenvolvimento de soluções adaptativas (Salehie & Tahvildari, 2009; Weyns *et al.*, 2013). Esse elemento compõe um modelo referencial constituído de uma sequência de etapas cíclicas. A Figura 1 apresenta os componentes do MAPE-K, cujas funções são descritas a seguir (Kephart & Chess, 2003; Salehie & Tahvildari, 2009; Computing *et al.*, 2006).

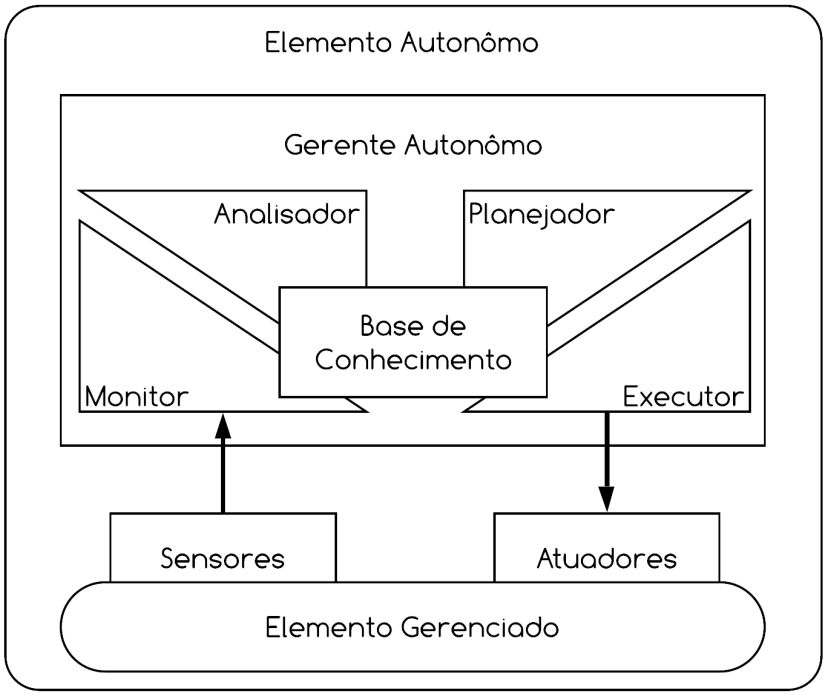


Figura 1: Arquitetura MAPE-K. **Fonte:** Adaptado de (Huebscher & McCann, 2008).

- *Elemento gerenciado*: é um software que atua em tempo de execução e que é gerenciado pelo *Gerente Autônomo*.
- *Monitor*: é o elemento que tem a função de coletar, correlacionar, agrupar e filtrar informações do Elemento Gerenciado através de *Sensores*, entidades que efetuam a extração de dados. Essas informações são enviadas para o Analisador.
- *Analisador*: é um processo para verificação das informações selecionadas na etapa de monitoramento. A verificação possibilita ao sistema adaptativo detectar quando

uma mudança é necessária, assim como, permite identificar a fonte do problema encontrado. Todas as informações encontradas nesta etapa, são despachadas para o Planejador.

- *Planejador*: é um mecanismo que estrutura ações necessárias para que os objetivos do sistema sejam atingidos. Logo, é de sua responsabilidade, determinar os componentes que precisam ser alterados no Elemento Gerenciado e como essas alterações devem ocorrer. De modo que seja criado um plano de adaptação, com ações que modificam o sistema, conforme os seus objetivos. Esse plano de adaptação é destinado ao Executor.

- *Executor*: é o componente responsável por aplicar o conjunto de ações definidas pelo Planejador. O Executor se comunica com o Elemento Gerenciado por meio de *Atuadores*, entidades que executam as ações desejadas.

- *Base de Conhecimento*: é um elemento passivo que armazena informações utilizadas pelas quatro etapas do Gerente Autônomo (Monitor, Analisador, Planejador e Executor). Ele é capaz de manter informações como históricos, métricas de desempenho, políticas de uso de recursos, entre outros.

2.2 MICROSSERVIÇOS

Os microsserviços são uma inovação na área de construção de aplicações distribuídas (Lewis & Fowler, 2014). A sua arquitetura foca na modelagem em torno do domínio de negócio que, por sua vez, facilita a integração de diferentes técnicas de desenvolvimento e tecnologias, superando problemas encontrados em arquiteturas tradicionais como as aplicações monolíticas (Lewis & Fowler, 2014; Newman, 2015).

A Figura 2 apresenta a diferença entre aplicações monolíticas e as MBAs. De modo geral, uma aplicação monolítica, geralmente, é criada como uma unidade composta de três principais partes: a interface do usuário, um banco de dados e a aplicação no lado servidor. Logo, quaisquer alterações no sistema envolvem a construção e a implantação de uma nova versão da aplicação no lado do servidor (Lewis & Fowler, 2014). Um microsserviço é um software autônomo, implantado de forma independente, e com uma funcionalidade específica claramente definida (Taibi *et al.*, 2017). Uma MBA é constituída por um conjunto de microsserviços que fornecem um único serviço. A comunicação entre esses elementos ocorre através de mecanismos leves, *e.g.,* como um barramento de mensagens simples do RabbitMQ ou por meio de *Application Programming Interfaces* (APIs) do HTTP (Lewis & Fowler, 2014).

O uso de microsserviços traz vários benefícios (Newman, 2015). Em primeiro lugar, como uma MBA é composta de um conjunto de microsserviços dissociados, torna-se possível implementá-los usando uma variedade de tecnologias. Como consequência, esse fato permite

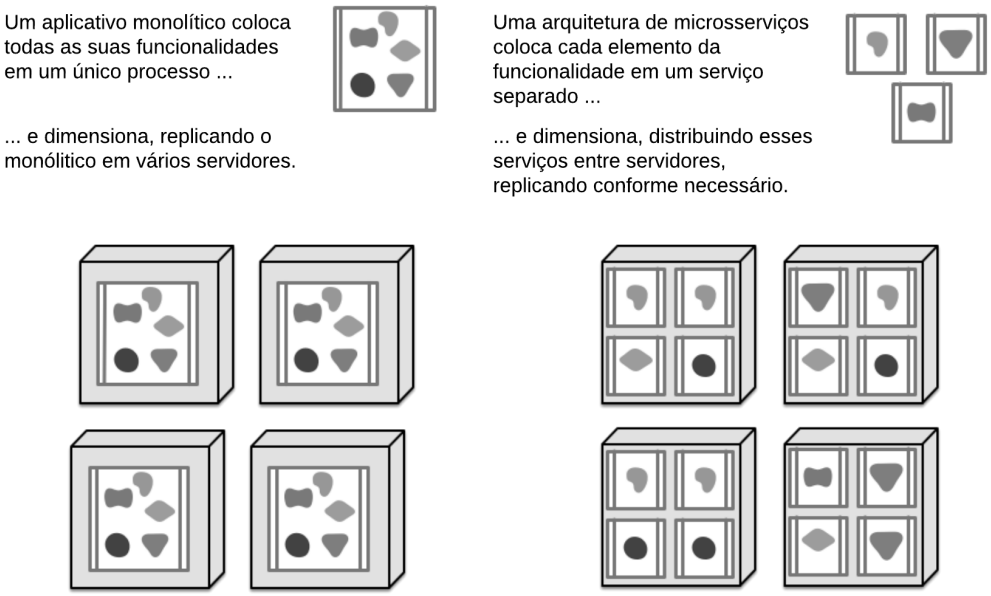


Figura 2: Microsserviços versus Aplicação monolítica. **Fonte:** Adaptado de (Lewis & Fowler, 2014).

que os desenvolvedores apliquem a melhor ferramenta para atingir níveis de desempenho estabelecidos ou resolver problemas específicos do contexto. Em segundo lugar, cada microsserviço tem uma responsabilidade específica em um contexto limitado. No caso de uma falha, apenas parte da aplicação é afetada (*e.g.,* registro do cliente) sem prejudicar todo o sistema tornando a aplicação resistente a falhas. Por fim, a escalabilidade é um recurso de aplicações baseadas em nuvem que permite a adaptação de componentes individuais ou de toda a aplicação a cargas de trabalho variadas. Como os microsserviços são dissociados, a maioria dos recursos do sistema pode ser dimensionada para atender às suas respectivas demandas de tempo de execução.

Quando se trata de aplicações monolíticas, a implementação usando diferentes linguagens é de alto risco e dispendiosa. Assim como, falhas em componentes específicos do sistema acarretam na indisponibilidade de todos os serviços providos. Além disso, por ser construída como um único bloco, a escalabilidade nessa arquitetura exige a replicação de toda aplicação. Logo, os microsserviços se tornaram muito populares na criação de aplicações complexas baseadas em nuvem (Newman, 2015).

2.2.1 Contêineres e Virtual Machines

As aplicações desenvolvidas para a nuvem geralmente usam técnicas de virtualização para alcançar escalabilidade (Pahl *et al.*, 2019). Essas aplicações podem ser implantadas em *Virtual Machines* (VMs) ou contêineres. Uma VM pode ser definida como uma cópia eficiente e isolada de uma máquina real (Laureno, 2006). Um *Hypervisor* é uma camada de software localizada entre a camada de hardware e o sistema operacional que gerencia e aloca recursos

de hardware da máquina real para a VM, permitindo que vários sistemas operacionais sejam executados no mesmo dispositivo (Desai *et al.*, 2013). Os contêineres oferecem um conceito de virtualização semelhante, mas são uma alternativa mais leve porque consomem menos recursos e são mais rápidos de provisionar (Pahl, 2015).

A Figura 3 apresenta uma comparação entre a virtualização utilizando *Hypervisor* e contêineres. A implantação baseada em *Hypervisor* é ideal para cenários em que aplicações na mesma nuvem precisam utilizar diferentes sistemas operacionais, *e.g.,* Ubuntu Linux, Debian Linux, Windows. Por outro lado, os contêineres são capazes de compartilhar o sistema operacional, os binários e as bibliotecas da máquina real.

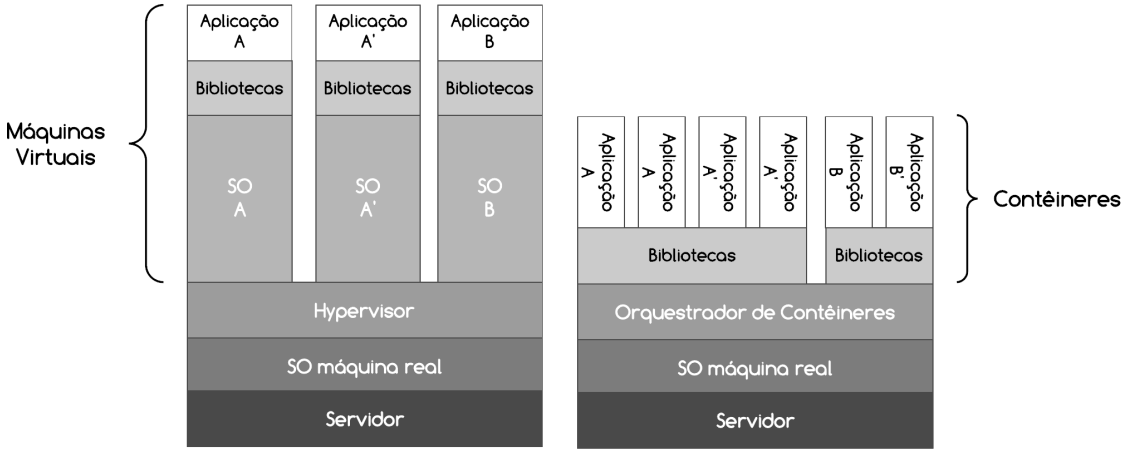


Figura 3: *Hipervisors* vs Contêineres. **Fonte:** Adaptado de (Bernstein, 2014).

A escalabilidade é um fator importante na distinção entre VMs e contêineres. Geralmente, quando uma aplicação precisa ser dimensionada, esse problema está relacionado a sobrecarga de demanda em algumas partes do sistema, não no sistema como um todo (Sampaio *et al.*, 2019). Se a aplicação for monolítica e se encontra implantada em uma VM, todos os componentes do sistema deverão ser replicados, incluindo aqueles que não precisam de recursos extras, gerando um consumo desnecessário de recursos.

MBAs, por outro lado, são soluções construídas por meio de um conjunto de serviços que podem ser implantados separadamente. Nesse caso, é possível lidar com o problema do dimensionamento desnecessário de recursos, por meio da implantação de partes da aplicação em VMs diferentes, *e.g.,* registro de clientes na VM1 e venda de produtos na VM2. Porém, ainda assim, cada VM está associada a um sistema operacional específico que aloca recursos da máquina real, *e.g.,* memória RAM e armazenamento em disco. Como resultado, essa alocação gera um provisionamento lento que varia de um até mais de dez minutos por VM (Pahl, 2015).

Como alternativa a esses problemas, os contêineres estão se destacando como uma tecnologia convencional para implantar MBAs. Isso ocorre porque, como não há sobrecarga de sistemas operacionais, a implantação de microsserviços em contêineres é muito mais eficiente,

pois não aloca recursos desnecessários como nos *Hypervisors*. Como resultado, é possível armazenar centenas de contêineres em um único *host* físico, em oposição a um número limitado de VMs (Bernstein, 2014).

2.2.2 Ferramenta de Clustering para Contêineres

A implantação de MBAs em contêineres é uma alternativa que facilita o dimensionamento e a atualização de determinados componentes da aplicação. Porém, conforme a quantidade de contêineres aumenta, novos problemas surgem que requerem a intervenção humana (Soppelsa & Kaewkasi, 2017), *e.g.,* falhas internas nos microsserviços, problemas de comunicação, indisponibilidade de contêineres. As Ferramentas de *Clustering* para Contêineres (FCCs) como Kubernetes[1] e Docker Swarm[2] surgiram para facilitar o gerenciamento de microsserviços (Florio & Nitto, 2016).

Kubernetes, criado em 2014 pela Google, é um orquestrador de código aberto projetado para implantar aplicações implementadas através de contêineres. Esse software tornou-se a ferramenta padrão para a implantação de aplicações nativas da nuvem, presente em quase todas as nuvens públicas. Há diversos benefícios associados à adoção do Kubernetes como o aumento na velocidade de tarefas de implantação e reparação de aplicações. Além disso, ele permite o dimensionamento automático de software e de equipes facilmente (Hightower *et al.*, 2017).

Docker Swarm, lançado em 2016, é uma ferramenta de código aberto desenvolvida para gerenciar e orquestrar um conjunto de recursos, *e.g.,* contêineres. O Docker Swarm transforma um grupo de nós em um *Cluster Docker*. A ferramenta, por sua vez, abstrai a complexidade de gerenciamento, oferecendo um conjunto de operações (*e.g.,* implantação, atualização e replicação de contêineres) por meio de única API. O Docker Swarm apresenta algumas vantagens como ser nativo no Docker, fácil de configurar e usar, entre outras (Soppelsa & Kaewkasi, 2017).

2.3 PREVISÃO DE SÉRIES TEMPORAIS

Aprendizagem de Máquina (AM) é um campo de estudo que lida com o desenvolvimento de algoritmos capazes de imitar a inteligência humana ao aprender sem serem explicitamente programados (El Naqa & Murphy, 2015; Samuel, 1959). Os algoritmos de AM são aplicados em vários problemas como a tomada de decisões sobre crédito, diagnóstico de dispositivos mecânicos e classificação automática de objetos celestes (Langley & Simon, 1995). Além disso, outra aplicação popular para algoritmos de AM é a previsão de séries temporais (Sapankevych & Sankar, 2009).

[1] https://kubernetes.io/
[2] https://docs.docker.com/engine/swarm/

2.3.1 Séries Temporais

Uma série temporal é composta por um conjunto de observações, medidas tipicamente em instantes de tempo sucessivos espaçados em intervalos de tempo uniformes (Lorido-Botran *et al.*, 2014). Uma série temporal pode ser representada através de notações matemáticas.

$$T = \{t_1, t_2, ..., t_n\} \tag{2.1}$$

Onde,

- t_1, t_2 e t_n, são observações da série temporal;
- n equivale ao tamanho da série temporal.

As séries temporais são contínuas quando as observações são registradas continuamente durante algum intervalo de tempo, *e.g.,* registro da temperatura do Brasil, 24 horas por dia, durante um ano. Nas séries discretas, as observações são feitas em intervalos de tempo fixos, *e.g.,* vendas de alimentos mensais de 1980 até 1990 (Brockwell *et al.*, 1991). Porém, esses termos não se referem a variável observada, que pode ser contínua ou discreta, estando somente relacionada a forma como a observação é conduzida (Ehlers, 2007).

Segundo Ehlers (2007), uma característica importante das séries temporais é que observações vizinhas são dependentes e, portanto, é possível analisar e modelar essa dependência. Para isto, no entanto, é preciso realizar o ordenamento temporal das informações coletadas, característica essa crucial para o processo de análise das séries temporais. A Figura 4 apresenta uma série temporal discreta composta pelo valor histórico da ação da empresa Microsoft. Outros exemplos de séries temporais são: número de passageiros mensais em linhas aéreas internacionais; medições anuais da vazão de rios; consumo anual de gás em um país; taxas de desemprego; valores diários da cotação de moedas e índices da bolsa de valores.

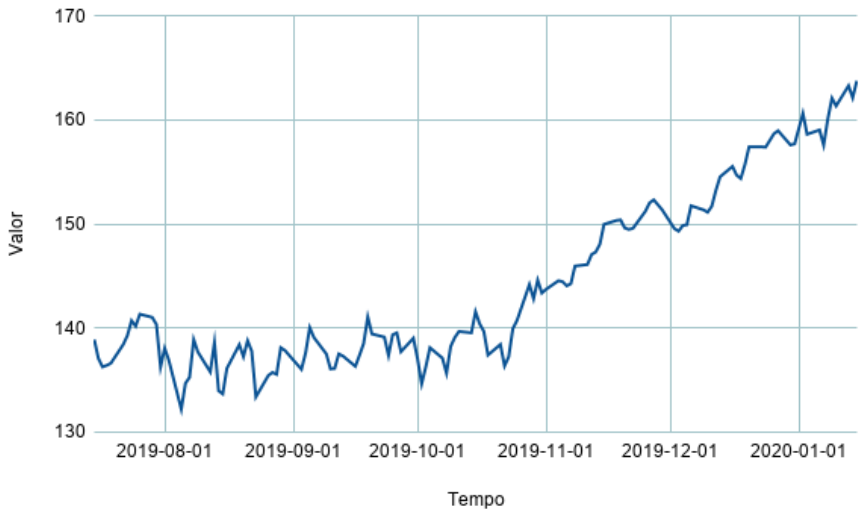


Figura 4: Série da empresa Microsoft na bolsa de valores. **Fonte:** NASDAQ.

Conforme Brockwell *et al.* (1991) e Mahalakshmi *et al.* (2016), a modelagem de séries temporais pode ser aplicada em diversos problemas como associação temporal (Sompolinsky & Kanter, 1986), agrupamento de séries temporais (Kisilevich *et al.*, 2010), classificação de séries temporais (Graves *et al.*, 2006) e previsão de séries temporais (Ajila & Bankole, 2016). A presente pesquisa teve o foco no problema de previsão de séries temporais no contexto de microsserviços. Esse problema consiste em criar um modelo capaz de representar o comportamento de uma determinada série e usá-lo para fazer previsões. Uma maneira comum na literatura para criar modelos representativos de séries temporais é através de algoritmos de Aprendizagem de Máquina.

2.3.2 Construção dos Modelos de Séries Temporais

A metodologia para construção de um modelo de séries temporais através de algoritmos de AM pode ser dividida em quatro etapas fundamentais: aquisição, pré-processamento, treinamento dos algoritmos de AM e previsão da série temporal. A aquisição determina o conjunto de informações alvo utilizado no processo de previsão, *e.g.,* o consumo de CPU, memória ou rede de um microsserviço. Essas informações precisam estar organizadas como uma série temporal (Seção 2.3.1). A Figura 5 apresenta uma informação de consumo de CPU de um determinado microsserviço, organizada como uma série temporal.

Figura 5: Série temporal do consumo de CPU de um microsserviço.

A etapa de pré-processamento inicia-se com a adequação do conjunto de treinamento. Essa adequação lida com duas principais tarefas: a definição da janela temporal e a normalização dos dados. Uma janela temporal é composta por um conjunto de retardos temporais (*lags*). Os retardos temporais são definidos com base na relação existente entre uma observação num determinado instante da série com observações anteriores, *e.g.,* a venda de computadores do mês de março pode estar relacionada a venda do mês de fevereiro. A Figura 6, mostra uma janela temporal composta de 20 retardos temporais.

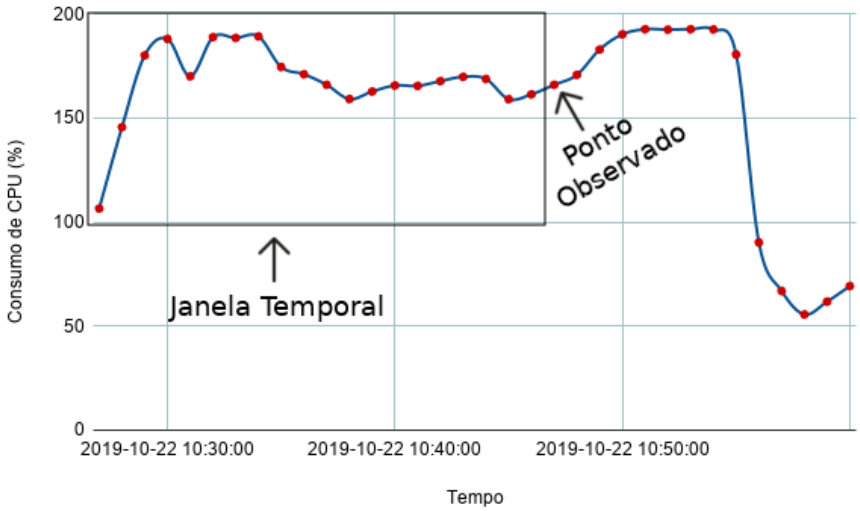


Figura 6: Janela temporal com 20 retardos.

Para determinar os retardos temporais existentes em uma série é possível utilizar a função de auto correlação (Scargle, 1989) ou algoritmos de otimização (Ribeiro *et al.*, 2011). O exemplo da venda de computadores é um caso com retardos temporais contínuos. Porém, é possível se deparar com casos não contínuos. Por exemplo, o número de publicações cientificas no Brasil no mês de julho está relacionado às publicações de maio e março. Logo, não há relação de julho com junho e abril, apesar destes serem retardos temporais próximos. Quando os retardos temporais mais relevantes são escolhidos, toda a série é processada e se transforma em um conjunto de janelas deslizante, como mostra a Figura 7.

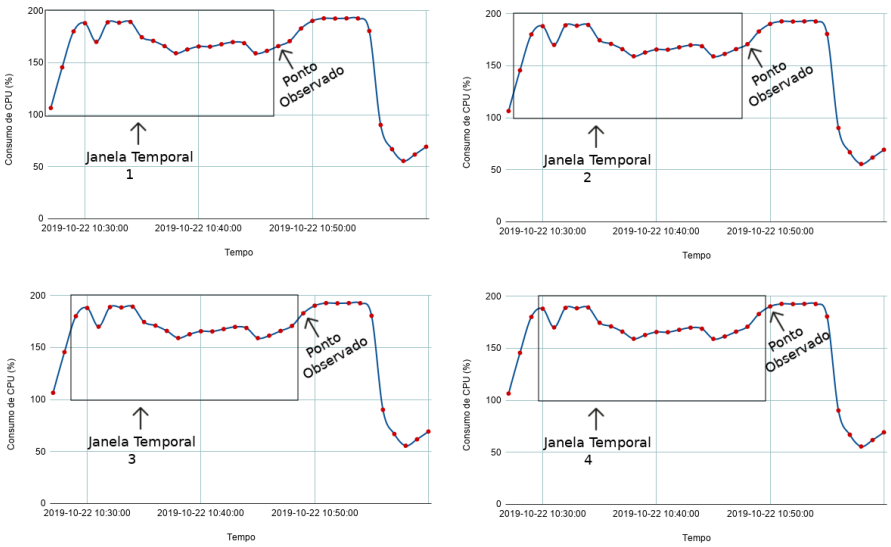


Figura 7: Janela temporal deslizante.

O processo de normalização ou padronização dos dados é utilizado para alterar a escala das informações para intervalos menores. Esse processo pode ser realizado através de diversas técnicas como o *min-max normalization*, *z-score normalization* e a *normalization by decimal*

scaling (Al Shalabi *et al.*, 2006). A normalização da série é importante para facilitar o treinamento de algoritmos que utilizam funções, durante o treinamento, que englobem o intervalo entre [0, 1], como é o caso do MLP (Zhang *et al.*, 2018). Além disso, a normalização é capaz de melhorar a previsão e a eficácia dos algoritmos preditivos (Al Shalabi *et al.*, 2006).

O objetivo da etapa de treinamento é construir um modelo capaz de representar o comportamento dos dados. Para isso, é necessário que as informações normalizadas sejam expostas a este. O processo de treinamento envolve o ajuste de determinados parâmetros (*e.g.,* a função de ativação) do algoritmo de AM, que permitem criar um modelo que represente os dados da série temporal. Esse ajuste é realizado calculando o erro métrico, ou seja, a diferença entre a previsão do modelo treinado e o valor real. Há algumas alternativas para calcular o erro métrico como Root Mean Square Error (RMSE), Symmetric Average Absolute Percentage Error (SMAPE) e o Average Relative Variance (ARV) (Ahmed *et al.*, 2010; de Mattos Neto *et al.*, 2010; Willmott *et al.*, 1985). Na etapa de previsão, o algoritmo está pronto e, portanto, é capaz de fazer previsões em tempo de execução.

2.3.3 Algoritmos para Previsão de Séries Temporais

A atividade de previsão de séries temporais depende de algoritmos capazes de capturar padrões presentes na série. Os algoritmos preditivos podem ser divididos em duas categorias: estatísticos e de AM (Nowicka-Zagrajek & Weron, 2002). Segundo Silva *et al.* (2018), algoritmos estatísticos possuem desempenho limitado para a previsão de séries temporais não lineares no mundo real. Algoritmos de AM, por outro lado, se destacam por seus bons resultados em termos de precisão. Além disso, a previsão de séries temporais usando modelos de AM é a técnica mais amplamente adotada por gerenciadores de aplicações em nuvem (Lorido-Botran *et al.*, 2014). Isto posto, essa seção descreve quatro diferentes algoritmos de AM usados nesta dissertação.

As Redes Neurais Artificiais (RNAs) são algoritmos computacionais de AM inspirados no funcionamento do cérebro humano. Segundo Hawkins (2004), uma RNA é um processador paralelo distribuído, constituído de unidades simples de processamento. Essa técnica se assemelha ao cérebro humano por ser capaz de adquirir conhecimento através de um processo de aprendizado e pela utilização de pesos sinápticos para o armazenamento do conhecimento adquirido.

A Figura 8 apresenta a arquitetura de uma RNA do tipo Multilayer Perceptron (MLP) (Werbos, 1974). O MLP é a classe mais popular das redes *feedforward* multicamada (Jain *et al.*, 1996). Segundo Gardner & Dorling (1998), o MLP consiste em um conjunto de unidades, conhecidas por neurônios, distribuídas ao longo de uma ou mais camadas e interligadas por conexões. As conexões são associadas por pesos e sinais de saída. Os pesos são utilizados como ponderadores da informação a ser processada, enquanto os sinais de saída são processados por funções de ativação, *e.g., sigmoid* ou *exponential linear unit*.

A arquitetura de uma rede MLP é dividida em três camadas: entrada, oculta e saída. A

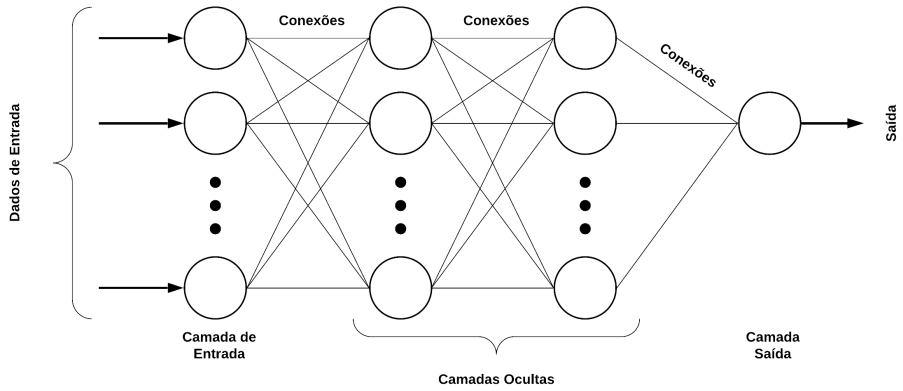


Figura 8: MLP com duas camadas ocultas. **Fonte:** Adaptado de (Gardner & Dorling, 1998)

primeira camada é composta pelo conjunto de neurônios de entrada. No problema abordado por esta dissertação, a quantidade de neurônios na camada de entrada equivale a quantidade de retardos temporais (*lags*) utilizados (Seção 2.3.2). A camada oculta, diferente da camada de entrada e saída, pode ser composta por uma ou mais camada intermediárias. Essas camadas ocultas são compostas por uma quantidade de neurônios, que variam conforme o problema. Geralmente, essa camada contém a maior parte dos neurônios da rede. Além disso, as camadas ocultas são responsáveis por processar as informações inseridas na entrada e direcionar os resultados para a camada de saída. Por fim, a quantidade de neurônios da camada de saída equivale a saída do problema, *e. g,* na previsão do consumo futuro de CPU, há apenas uma única informação de interesse e, portanto, a camada de saída contém apenas um neurônio. Além disso, similar as camadas intermediárias, a camada de saída processa novamente as informações e apresenta o resultado da rede.

Segundo Gardner & Dorling (1998), o treinamento de uma MLP tem como objetivo ajustar os pesos da rede para encontrar a combinação que resulta no menor erro para o modelo. Conforme mencionado na Seção 2.3.2, durante a fase de treinamento a rede precisa ser exposta aos dados, que foram transformados em conjunto normalizado. Cada linha dessa informação contém uma janela temporal (entrada) e um ponto observado (a saída desejada). Os pesos, portanto, são ajustados conforme o erro métrico resultante da diferença entre a saída desejada e o valor previsto. Esse processo é conhecido como treinamento supervisionado e é aplicado no MLP através do algoritmo *backpropagation* (Rumelhart *et al.*, 1986). Os passos de treinamento do *backpropagation* são apresentadas a seguir:

1. Inicializar os pesos da rede usando pequenos valores aleatórios;
2. Apresentar o primeiro vetor de entrada, ou seja, a primeira janela temporal dos dados de treinamento;
3. Propagar o vetor de entrada pela rede para obter uma saída;
4. Calcular o erro métrico do sinal comparando o valor previsto com a saída desejada (valor real);

5. Propagar o sinal de erro de volta pela rede;
6. Ajustar os pesos para minimizar o erro geral;
7. Repetir os passos 2 a 7 com o próximo vetor de entrada, até que o erro geral seja satisfatoriamente pequeno.

Random Forest (RF) (Breiman, 2001) é um método usado para problemas de classificação e regressão (Liaw *et al.*, 2002). Segundo Wang *et al.* (2016), o RF é constituído por um conjunto de árvores de regressão. Cada árvore de regressão representa um conjunto de condições ou restrições organizadas hierarquicamente e aplicadas sucessivamente de uma raiz a uma folha da árvore. Essas árvores são cultivadas a partir de dados extraídos da reamostragem (*bootstrapping*) dos dados originais (Palmer *et al.*, 2007).

A Figura 9 apresenta a topologia de uma árvore de regressão. Nessa Figura, os círculos representam os nós decisórios (X, X_1, X_2, X_3 *e* X_6); os quadrados representam os nós terminais (X_4, X_5, X_7, X_8, X_9, X_{10}); as linhas representam os ramos que ligam dois nós. O X é um nó raiz da árvore que compreende todo o espaço do conjunto de dados. X_1 e X_2 são os nós descendente esquerdo e direito de X, respectivamente. Assim como, X_1 e X_2 são disjuntos, ou seja, nenhum elemento de *X1* está presente em *X2* ($X = A \cup B$). Por fim, abaixo de cada nó terminal encontra-se o resultado correspondente da árvore.

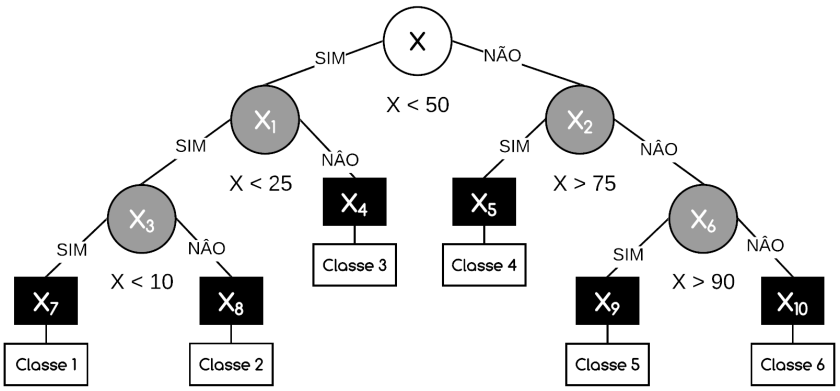


Figura 9: Árvore de regressão. **Fonte:** Adaptado de (Ferreira, 1999)

Para determinar o critério de divisão do nó de uma árvore são analisados os dados de treinamento. Um dado de treinamento é uma das amostras do conjunto de reamostragem. O algoritmo comumente utilizado para a escolha desse critério é o CART (Classification and Regression Tree) (Palmer *et al.*, 2007). O CART determina dentre todos os elementos do conjunto, aquele que gere os dois "melhores" nós descendentes, considerando o resultado de uma métrica de avaliação (Cutler *et al.*, 2012; Palmer *et al.*, 2007). Em seguida, essa atividade é realizada nos nós descendentes de forma recursiva. O algoritmo, por sua vez, é encerrado quando atinge um critério de parada pré-definido. Esse conjunto de tarefas é efetuado para cada árvore do RF. De forma resumida, os passos do treinamento do RF são apresentados a seguir:

1. Colete um vetor de amostras N a partir dos dados originais usando a técnica de *bootstrapping*;
2. Use N_i, uma amostra de N, como dado de treinamento para cultivar uma árvore;
    (a) Inicialmente, todas as observações de N_i estão em um único nó (raiz);
    (b) Repita as etapas a seguir recursivamente para cada nó não dividido até que o critério de parada seja atendido:
        i. Encontre o melhor critério de divisão entre todos os elementos de N_i;
        ii. Divida o nó em dois nós descendentes.
3. Repita o passo 2 até que não haja mais elementos de N.

Finalmente, para prever novos dados usando o RF, é necessário agregar as previsões de cada árvore, ou seja, a previsão final é o resultado da média das previsões de todas as árvores (Cutler *et al.*, 2012).

Support Vector Machine (SVM) (Vapnik, 1963) é um algoritmo de aprendizagem supervisionado utilizado para classificação de dados. O objetivo do SVM é construir um hiperplano para separação de classes de padrões. A Figura 10 apresenta exemplos de hiperplanos com dados linearmente separáveis. Dado que existem diferentes possíveis hiperplanos capazes de separar corretamente um determinado conjunto de dados. O objetivo do SVM é encontrar o hiperplano "ideal". O hiperplano "ideal" é aquele que minimiza ao máximo os erros dos dados de treinamento e contém uma margem de separação alta.

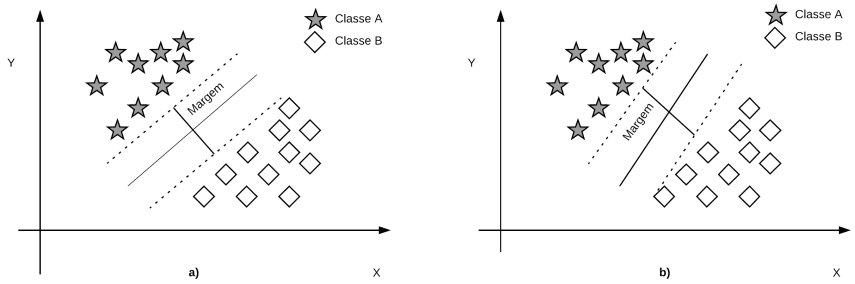


Figura 10: Separação de dados lineares por meio de hiperplanos.

O SVR (Drucker *et al.*, 1997) é um algoritmo estendido da SVM aplicado em problemas de regressão e utilizado para previsão de séries temporais. Assim como o SVM, o objetivo do SVR é encontrar hiperplanos capazes de separar os dados. No entanto, os dados usados pelo SVR geralmente são não linearmente separáveis. Dessa forma, ele precisa realizar um mapeamento de um espaço de entrada para um outro espaço com uma dimensionalidade maior. De acordo com o teorema de Cover, dados não-linearmente separáveis podem ser linearmente separáveis em uma dimensão maior (Haykin, 1999). A atividade de mapeamento é realizada através de funções de

Kernel (Bergman & Schiffer, 1951). Na literatura são encontradas diversas funções de *Kernel*, como Radial Basis Function (RBF), *Polynomial* e *Sigmoid* (Hussain *et al.*, 2011).

O treinamento de uma SVR consiste em encontrar uma função $f(x)$ com liberdade de erro menor ou igual a ε, ou seja, o erro máximo entre a previsão da SVR e a saída desejada é ε (Smola & Schölkopf, 2004). Essa relação pode ser representada matematicamente (Xu *et al.*, 2015):

$$\sum_{n=1}^{N} \langle \omega, x_i \rangle - b - y_i \leq \varepsilon \tag{2.2}$$

Onde,

- ω é um vetor de pesos;
- x_i é um exemplo de treinamento;
- b é um limiar;
- y_i é a saída desejada;
- N é o tamanho do conjunto avaliado.

Extreme Gradient Boosting (XGBoost) (Chen & Guestrin, 2016) é um algoritmo usado no contexto de aprendizagem supervisionada para problemas de classificação e regressão (Chen *et al.*, 2018; Gumus & Kiran, 2017). De modo geral, o XGBoost surge como uma otimização ao método tradicional Gradient Boosting Decision Tree (GBDT) melhorando a velocidade de processamento, o desempenho para generalização e a escalabilidade do método (Chen *et al.*, 2018; Zhang & Zhan, 2017). Ele é utilizado em uma variedade de problemas para obter resultados de ponta em muitos desafios de dados propostos pelo Kaggle[3] e KDDCup[4] (Chen & Guestrin, 2016).

A ideia básica do XGBoost é usar um conjunto de preditores (árvores de regressão) com baixa precisão para construir um preditor de melhor desempenho (Ren *et al.*, 2017; Zhang & Zhan, 2017). Para isso, cada preditor é construído usando a técnica de *Gradient Boosting* (Friedman *et al.*, 2000). Basicamente, o *Gradient Boosting* é capaz de cultivar uma nova árvore para o conjunto se baseando em todas as árvores anteriores. Dessa forma, o algoritmo é capaz de minimizar a função de objetivo a cada nova árvore cultivada (Zhang & Zhan, 2017). A função objetivo $Obj(\Theta)$ (Chen & Guestrin, 2016) é representada matematicamente, a seguir:

$$Obj(\Theta) = \sum_{i} L(\Theta) + \sum_{k} \Omega(\Theta) \tag{2.3}$$

$$L(\Theta) = l(\hat{y}_i, y_i) \tag{2.4}$$

[3]https://www.kaggle.com/
[4]https://www.kdd.org /kdd-cup

$$\Omega(\Theta) = \gamma\tau + \frac{1}{2}\lambda \|\omega\|^2 \tag{2.5}$$

Onde,

- Θ refere-se aos vários parâmetros presentes na fórmula;
- $L(\Theta)$ é uma função de custo para o treinamento das árvores;
- $\Omega(\Theta)$ é o termo de regularização.

A função de custo $L(\Theta)$ calcula a diferença entre a previsão ($\hat{y}_i$) e a saída desejada (y_i) (Chen *et al.*, 2018). O resultado dessa função permite ajustar os dados para o modelo. O termo de regularização é usado no controle da complexidade dos modelos penalizando aqueles que são mais complexos. A penalização evita o problema de *overfitting* e tende a selecionar modelos preditivos genéricos (Chen & Guestrin, 2016).

Por fim, apesar das similaridades, o RF e XGBoost apresentam características diferentes. Por exemplo, o RF é um algoritmo que usa a técnica de *bootstrapping* para o cultivo de novas árvores. Ou seja, o RF coleta uma reamostra do conjunto de dados original e gera uma nova árvore. O XGBoost utiliza preditores simples de baixa precisão para modelar um preditor mais complexo. Além disso, cada novo preditor do XGBoost é baseado nos preditores anteriores, enquanto no RF, cada preditor é treinado independentemente.

2.4 CONSIDERAÇÕES FINAIS

Este capítulo apresentou os conceitos e fundamentos teóricos usados como base para esta pesquisa. Na área de sistemas adaptativos foram discutidos tópicos como *feedback loop*, microsserviços, VMs e contêineres e as ferramentas de *clustering* para contêineres. Em relação aos tópicos de Aprendizagem de Máquina foram detalhados conceitos sobre séries temporais, algoritmos para previsão de séries temporais e como as previsões são realizadas.

3 ML-ADAPT

Este capítulo apresenta em detalhes o desenvolvimento do ML-Adapt. A Seção 3.1 traz uma visão geral com informações relacionadas aos elementos que compõem o ML-Adapt e o Sistema Gerenciado. A Seção 3.2 apresenta o fluxo de trabalho do ML-Adapt descrevendo as interações dos componentes. As seções 3.3 a 3.6 detalham os componentes do MAPE-K. Por fim, a Seção 3.7 descreve os elementos que integram o Sistema Gerenciado.

3.1 VISÃO GERAL

ML-Adapt (***M**achine **L**earning **Adapt**ation of microservice-based applications*) é um ambiente para adaptação proativa de MBAs. Ele implementa um *feedback loop* baseado no MAPE-K e usa algoritmos de aprendizagem de máquina como elemento central do processo de adaptação. ML-Adapt gerencia o consumo de CPU dos microsserviços mantendo a QoS, ao mesmo tempo que busca a economia de recursos. A métrica de consumo de CPU foi escolhida por ser amplamente adotada em pesquisas que lidam com o gerenciamento de recursos de microsserviços (Wong, 2018; Alipour & Liu, 2017; Zhang *et al.*, 2018; Klinaku *et al.*, 2018).

A solução proposta apresenta um modo de operação proativo que emprega modelos de AM para previsão do consumo de CPU dos microsserviços. Por conseguir prever condições futuras da aplicação, o ambiente adaptativo é capaz de perceber antecipadamente possíveis violações no desempenho. Dessa forma, ele pode preparar um plano adaptativo que contenha ações para solucionar a violação percebida. A Figura 11 apresenta uma visão geral dos elementos envolvidos diretamente no processo adaptativo. Estes elementos compõem dois grupos: *Sistema Gerenciado*, que contempla toda a estrutura das aplicações a serem gerenciadas; e o *Feedback Loop*, que engloba a arquitetura do ML-Adapt.

O Sistema Gerenciado é composto pelos elementos: *Coletor* e as MBAs. O Coletor é um banco de dados, implantado dentro de uma Ferramenta de *Clustering* para Contêineres (FCC), responsável por monitorar métricas das MBAs e armazená-las. Uma MBA é composta por microsserviços (*e.g.,* μX_1, μX_2, μX_3) que são implementados como serviços totalmente independentes, mas que atuam para fornecer um conjunto de serviços complexos. O Feedback Loop é constituído pelos componentes: *Monitor*, *Analisador*, *Planejador* e *Executor*. O Monitor é o primeiro componente do MAPE-K cuja função é comunicar-se com o Coletor para obter, processar e disponibilizar informações das MBAs para etapas posteriores do Feedback Loop, em tempo de execução. O Analisador é constituído de dois elementos: o *Previsor* e o *Analisador de Previsão*. O Previsor prediz o estado futuro dos microsserviços, *e.g.,* se o microsserviço μX_1 irá consumir 70% de CPU no próximo minuto. O Analisador de Previsão é responsável por avaliar e determinar as condições futuras dos microsserviços. O Planejador é encarregado de planejar e propor um conjunto de ações capazes de reconfigurar os microsserviços. O Executor comunica-se com a *API de Gerenciamento* para executar comandos que modificam os elementos que compõem a FCC.

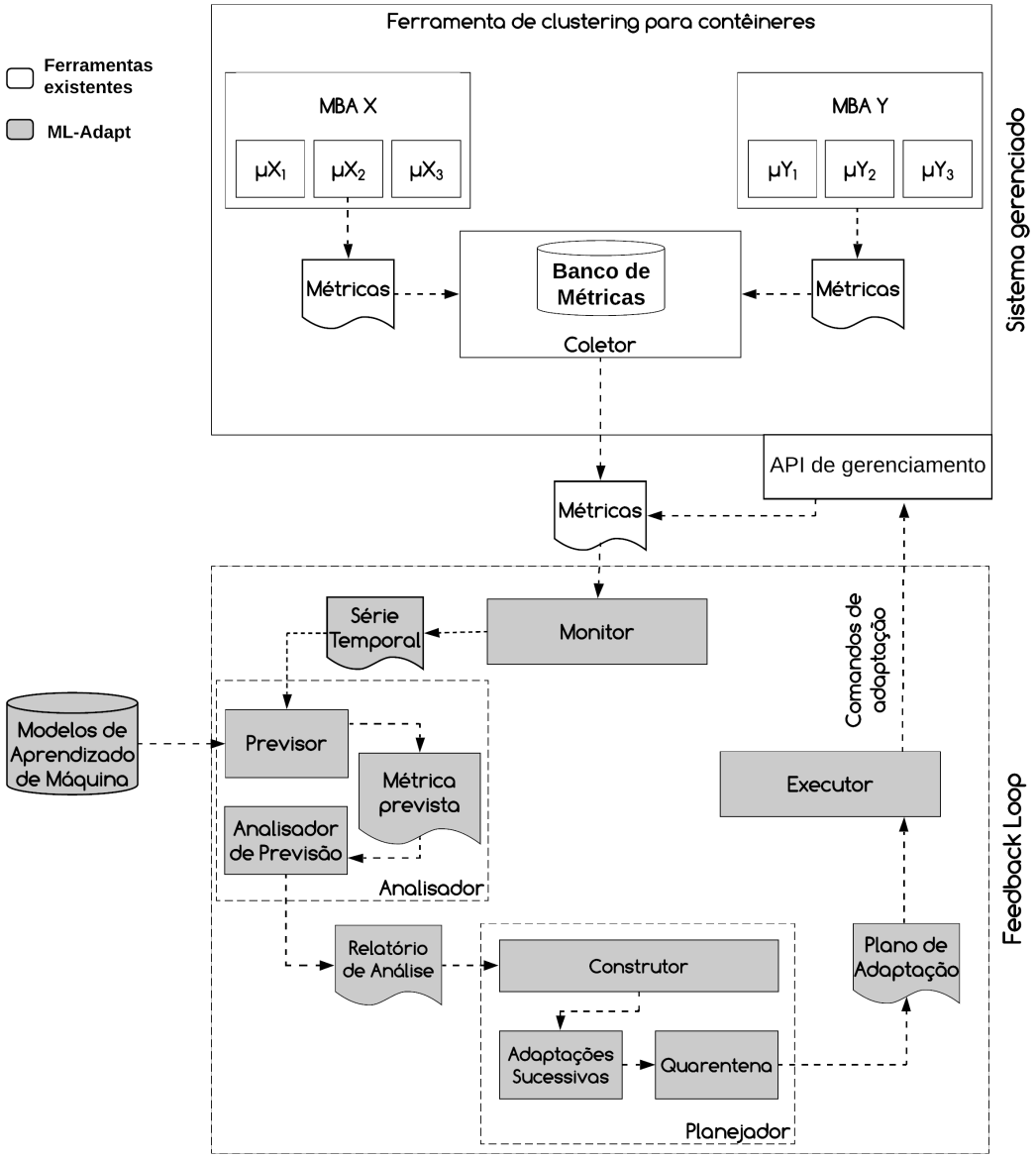


Figura 11: Visão geral.

3.2 FLUXO DE ADAPTAÇÃO

A Figura 12 apresenta o fluxo de trabalho do ML-Adapt. Esse fluxo inicia com a atividade *Coletar o Consumo de CPU* dos microsserviços implantados na FCC. Todas as ações efetuadas nesse processo são de responsabilidade do Monitor. Inicialmente, o Monitor solicita o histórico de consumo de CPU de cada microsserviço. Essa informação está armazenada no Coletor. Em seguida, o Monitor processa esses dados para que cada microsserviço seja representado por uma série temporal. Essas séries temporais são utilizadas na atividade responsável por *Prever o Consumo de CPU*.

A atividade Prever o Consumo de CPU utiliza as séries temporais coletadas em modelos de AM, a fim de determinar as condições futuras dos elementos monitorados. O Analisador de Previsão, responsável por *Analisar o Consumo de CPU Previsto*, inspeciona as previsões obtidas,

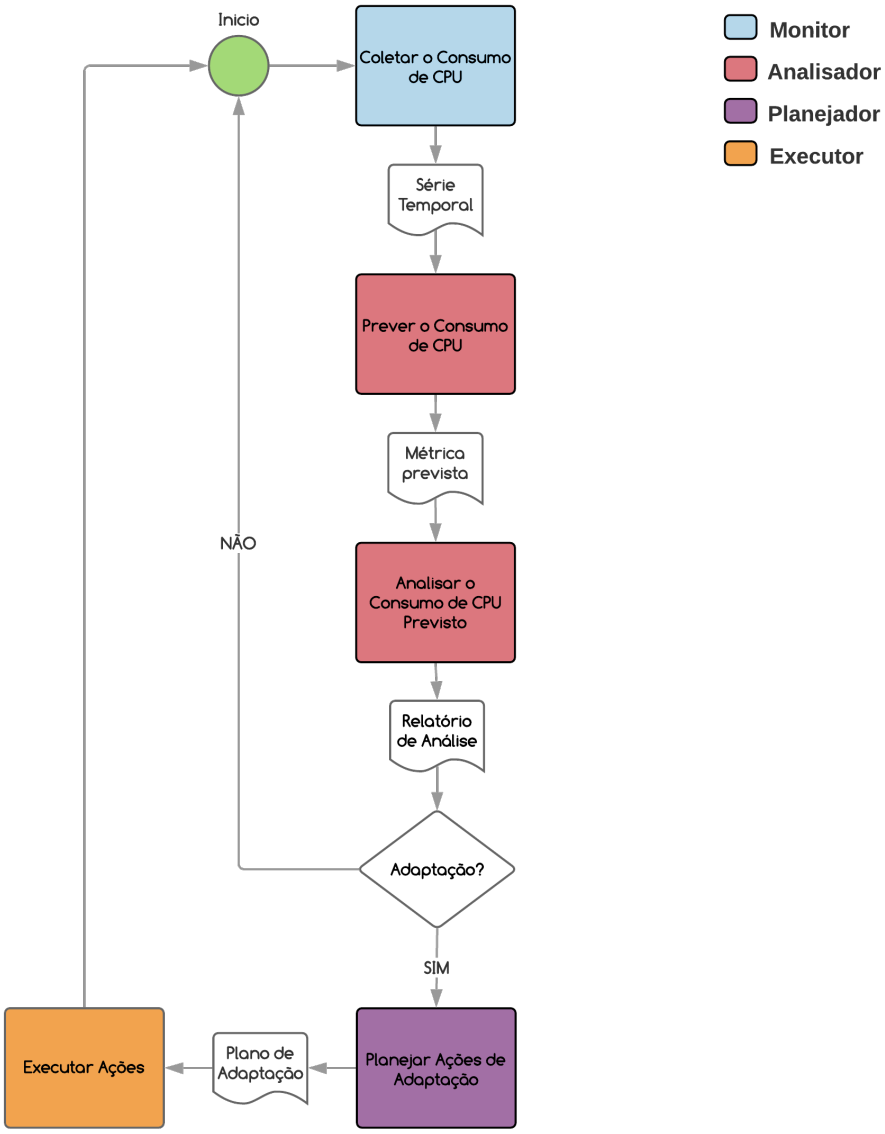


Figura 12: Fluxo de trabalho.

e determina se ocorrerão futuras violações nos limites definidos, referentes ao consumo de CPU.

Se nenhuma violação for encontrada, um novo ciclo se inicia. Por outro lado, se for identificada a necessidade de alguma adaptação, o fluxo será direcionado para a atividade *Planejar Ações de Adaptação*. Nesta atividade, o *Relatório de Análise* é inspecionado. Este relatório contém observações das condições de cada microsserviço (*e.g.,* o microsserviço μX_1 está operando com poucas réplicas), para propor um *Plano de Adaptação*. A inspeção permite calcular a quantidade de réplicas necessárias para um determinado microsserviço, conforme a demanda de consumo prevista. Finalmente, a atividade *Executar Ações* transforma as solicitações definidas no plano de adaptação e as executa modificando a quantidade de réplicas implantadas na FCC, processo que fica no Executor.

3.3 MONITOR

O Monitor, mostrado na Figura 11, é projetado para coletar métricas através do Coletor e da API de Gerenciamento. O Coletor é responsável por obter e armazenar métricas em tempo de execução, como consumo de CPU, memória e uso da rede. Enquanto isso, a API de Gerenciamento produz métricas como o número de *hosts* disponíveis e o número atual de microsserviços executando. O Monitor é flexível e permite a configuração de um conjunto de parâmetros, como o intervalo entre coletas (*e.g.,* a cada 1 minuto), as métricas (*e.g.,* processamento, memória, rede), tamanho da amostra coletada (*e.g.,* 100 amostras), o intervalo entre amostras (*e.g.,* 2 minutos) e os microsserviços monitorados.

O intervalo entre coletas é um aspecto crucial para manter a QoS da aplicação. Se este intervalo for curto, o ambiente adaptativo obtém informações detalhadas sobre o estado do Sistema Gerenciado. Porém, o monitoramento pode sobrecarregar o Coletor. Além disto, a análise de uma amostra de um período curto pode levar o ML-Adapt a identificar situações transientes e classificá-las como violações. Se o intervalo entre coletas for longo, o ambiente adaptativo pode não identificar e corrigir violações, deteriorando o desempenho da aplicação.

O tamanho da amostra coletada e o intervalo entre amostras são parâmetros baseados nas informações utilizadas na construção dos modelos de AM (Seção 3.4.1) e no Planejamento (Seção 3.5). Dessa forma, as métricas monitoradas são o consumo de CPU e o número atual de réplicas de cada microsserviço. Estes parâmetros são baseados no retardo temporal dos modelos preditivos (Seções 2.3.2 e 3.4.3) definido na Tabela 2.

Quando o Monitor precisa coletar estas métricas, ele executa requisições ao Coletor e a API de Gerenciamento, utilizando os parâmetros mencionados. A métrica de utilização de CPU é obtida pelo Coletor, enquanto o número atual de réplicas em execução é obtido através da API de Gerenciamento da FCC. Em seguida, essas informações são enviadas para o Analisador.

3.4 ANALISADOR

A estratégia de dimensionamento automático adotada no ML-Adapt é centrada na previsão das demandas de recursos das MBAs. Como mencionado anteriormente, essa previsão é efetuada no Analisador (Figura 11), o principal componente do ML-Adapt. A *Métrica prevista* gerada pelo Previsor é usada pelo Analisador de Previsões para avaliar possíveis violações ao consumo de recursos, *e.g.,* se o consumo da CPU for superior a 80%.

O resultado dessa avaliação permite estimar a quantidade “ideal” de recursos alocados para atender à demanda dos microsserviços com as violações identificadas pelo Analisador de Previsões. A satisfação da demanda busca reduzir os recursos alocados ou manter a QoS. Por exemplo, diminuir o número de réplicas para reduzir um eventual excesso de recursos alocados ou aumentar o número de réplicas para suportar um aumento repentino na carga de trabalho.

As subseções seguintes apresentam detalhes de funcionamento do Analisador.

3.4.1 Modelos de Aprendizagem de Máquina

A previsão de recursos é realizada pelo Previsor utilizando os modelos de AM. Os modelos são treinados usando um conjunto de dados contendo o consumo de CPU dos microsserviços obtidos conforme a etapa de aquisição dos dados (Seção 2.3.2). Esse conjunto de dados, se necessário (*e.g.,* o dado não está ordenado temporalmente ou não está espaçado em intervalos de tempo uniformes), precisa ser transformado em uma série temporal, estruturado conforme mostrado na Tabela 1.

Tabela 1: Conjunto de dados estruturado como uma série temporal.

Índice	Tempo	Processamento (%)
0	2019-10-02 10:37:00	14.89
1	2019-10-02 10:38:00	27.06
...	...	...
...	...	...
N	2019-10-05 11:36:00	82.04

Como apresentado na Seção 2.3.2, depois da aquisição dos dados, as próximas etapas na construção dos modelos de AM, são o: pré-processamento das informações, o treinamento do algoritmo e a avaliação da sua acurácia. Vale observar que a previsão de recursos está no nível de microsserviço, o que significa que o ML-Adapt percebe cada microsserviço como um sistema único. Ou seja, apesar dos microsserviços atuarem para fornecer um conjunto de serviços complexos, as relações existentes entre estes não são distinguíveis ao ML-Adapt. Logo, é necessário um modelo treinado para cada microsserviço.

Na etapa de pré-processamento, como cada microsserviço precisa de um modelo de AM treinado, cada série temporal é preparada separadamente. Nessa preparação, as séries temporais são organizadas como janelas deslizantes, onde cada janela é delimitada pelo tamanho do retardo temporal (*lags)* conforme definido na Tabela 2. Em seguida, aplica-se a normalização que é um processo que modifica os valores da janela deslizante para um determinado intervalo. No ML-Adapt, essas informações são normalizadas ($\hat{X}_i$) usando a Equação (3.1), referente a *min-max normalization* (Al Shalabi *et al.*, 2006), para o intervalo entre [0, 1].

$$\hat{X}_i = \frac{X_i - X_{min}}{X_{max} - X_{min}} \tag{3.1}$$

Onde,

- X_i é o valor não normalizado;
- X_{min} é o valor mínimo obtido do conjunto de dados;
- X_{max} é o valor máximo obtido do conjunto de dados.

Após realizar o pré-processamento na série, as informações normalizadas são geralmente divididas em três conjuntos: treinamento, validação e teste (Wong, 2018). O conjunto de

treinamento é o único utilizado como entrada para o algoritmo de AM. O conjunto de validação é utilizado para verificar os melhores parâmetros que permitem a um algoritmo criar um modelo de AM que represente o comportamento da série. Por fim, o conjunto de teste é útil para validar o modelo treinado, uma vez que seus dados não foram expostos ao algoritmo durante o treinamento. Além disso, é possível encontrar na literatura pesquisas que utilizam apenas dois conjuntos (treinamento e teste), porém parte do conjunto de treinamento é usado no processo de validação (Nikravesh *et al.*, 2017).

A divisão de dados é comumente adotada no processo de treinamento dos algoritmos de AM. Por exemplo, os percentuais adotados com dois conjuntos (treinamento e teste) podem variar entre 60% e 40% (Kumar & Singh, 2018; Nikravesh *et al.*, 2017) ou 80% e 20% (Alipour & Liu, 2017). Por outro lado, para três conjuntos (treinamento, validação e teste) é possível adotar o percentual 60%, 20% e 20% (Wong, 2018). Uma vantagem da sua adoção é a possibilidade de verificação do problema de *overfitting* comum em modelos de AM (Hawkins, 2004). Este problema acontece quando um modelo de AM apresenta bom desempenho para prever o comportamento de dados apresentados ao algoritmo (conjuntos de treinamento e validação), ao mesmo tempo que demonstra um mal desempenho ao prever novos padrões (conjunto de teste).

Ao concluir o pré-processamento das informações, os dados estão preparados para a etapa de treinamento dos algoritmos. Neste trabalho, são considerados quatro diferentes algoritmos de AM: MLP, RF, SVR e XGBoost (Ver Seção 3.4.1). Vale observar que esses algoritmos foram adotados em pesquisas relacionadas (Alipour & Liu, 2017; Kumar & Singh, 2018; Nikravesh *et al.*, 2017; Wong, 2018; Grohmann *et al.*, 2019; Chen & Guestrin, 2016). No treinamento, o método *Grid Search* (Chang & Lin, 2011) é aplicado para encontrar os melhores parâmetros de configuração para cada algoritmo. A Tabela 2 mostra os parâmetros escolhidos por algoritmo. Os parâmetros adotados neste trabalho são uma adaptação daqueles utilizados em outros trabalhos (Ajila & Bankole, 2016; Gumus & Kiran, 2017; Silva *et al.*, 2018; Grohmann *et al.*, 2019). Os passos do método *Grid Search* são apresentados a seguir:

1. Escolha um conjunto de parâmetros;
2. Use um subconjunto desses parâmetros para treinamento;
3. O conjunto de treinamento é usado para treinar o algoritmo;
4. O conjunto de validação é usado no modelo gerado pelo treinamento;
5. Calcula-se o erro métrico da predição do passo 4;
6. Repita o passo 2 ao 5 até não haver mais parâmetros;
7. O melhor conjunto de parâmetros é aquele com menor erro métrico.

Para determinar o melhor conjunto de parâmetros, é necessário avaliar a precisão dos modelos de AM treinados. Nesta etapa, ML-Adapt adota para cálculo do erro métrico o Root Mean Square Error (RMSE) (Willmott *et al.*, 1985), definido pela Equação (3.2).

Tabela 2: Parâmetros dos algoritmos de Aprendizagem de Máquina.

Algoritmo	Parâmetros	Valores
MLP	*Input (Lags)*	20
	Activation Function	*Identity, Logistic, Tanh, Relu*
	Algorithm	*adam, lbfgs, sgd*
	Learning Rate	*Adaptive, Constant, Invscaling*
	Nodes in Hidden Layer	1, 5, 10, 50, 100
RF	*Input (Lags)*	20
	Bootstrap	*True, False*
	Number of Features	*Auto, Sqrt*
	Number of Trees	200, 1000, 2000
	Maximum Depth	10, 60, 100, None
SVR	*Input (Lags)*	20
	Cost	0.1, 1, 5, 10, 100, 1000
	Epsilon	0.00001, 0.0001, 0.001, 0.1, 1
	Gamma	0.1, 0.5, 10, 50, 100, 1000
	Kernel	*Radial Basis Function, Sigmoid*
XGBoost	*Input (Lags)*	20
	Gamma	0.5, 1, 1.5, 2, 5
	Maximum Depth	3, 4, 5
	Subsample Ratio	0.6, 0.8, 1.0

$$RMSE = \sqrt{\frac{\sum_{n=1}^{N}(y_i - \hat{y}_i)^2}{N}} \tag{3.2}$$

Onde,

- y_i é o valor real do recurso, *e.g.,* 90% de consumo de CPU;
- $\hat{y}_i$ é o valor previsto para o recurso, *e.g.,* 95% de consumo de CPU;
- N é o tamanho do conjunto avaliado.

O modelo que prediz corretamente todos os dados produz um RMSE de 0. RMSE é comum em pesquisas que lidam com a predição de recursos para aplicações na nuvem (Borkowski *et al.*, 2016; Moreno-Vozmediano *et al.*, 2019; Kumar & Singh, 2018). Finalmente, depois de executar essas etapas, o ML-Adapt produz um modelo de AM para cada microsserviço gerenciado. Porém, vale salientar que como há quatro algoritmos diferentes, existem quatro tipos de modelos diferentes por microsserviço.

3.4.2 Previsor

O Previsor é projetado para tratar, em tempo de execução, as séries temporais que serão inseridas nos modelos de AM treinados por meio dos algoritmos MLP, RF, SVR e o XGBoost. Atualmente, ele permite o carregamento de modelos através da parametrização, normalização das

séries temporais, predição de recursos e o retorno da informação prevista a sua escala original (desnormalização), conforme apresentado no Algoritmo 1.

Algoritmo 1: Algoritmo do Previsor.

```
Entrada: nomeMicrosservico, serieTemporal, referenciaModeloAM
Saída: previsaoRecursoCPU
1 início
2     modeloAM = carregarModelo(nomeMicrosservico, referenciaModeloAM);
3     valorNormalizado = normalizar(nomeMicrosservico, serieTemporal);
4     previsao = obterPrevisao(modeloAM, valorNormalizado);
5     previsaoRecursoCPU = desnormalizar(nomeMicrosservico, previsao);
6     retorna previsaoRecursoCPU
7 fim
```

As operações de normalização e desnormalização são efetuadas a cada solicitação do Previsor. Durante a construção dos modelos preditivos (Seção 3.4.1), as séries temporais contendo o consumo de CPU dos microsserviços são normalizadas para treinamento. Por conta disso, os dados processados pelo Previsor, que são diferentes e coletados em tempo de execução pelo Monitor através do Coletor, precisam também serem normalizados antes da previsão dos recursos. Além disso, como o consumo previsto é normalizado e o Analisador de Previsões verifica apenas violações em dados não normalizados, é necessário retornar a previsão à sua escala original. O retorno à escala original (X_i) é realizado através da Equação (3.3).

$$X_i = \hat{X}_i * (X_{max} - X_{min}) + X_{max} \tag{3.3}$$

Onde,

- $\hat{X}_i$ é o valor normalizado;
- X_{max} é o valor máximo obtido do conjunto de dados;
- X_{min} é o valor mínimo obtido do conjunto de dados.

3.4.3 Analisador de Previsão

O Analisador de Previsão é responsável por avaliar as previsões efetuadas pelo Previsor e determinar a existência de violações ao consumo de CPU dos microsserviços. Determinar antecipadamente uma violação de recursos otimiza a QoS dos microsserviços em execução. De modo que, a aplicação mantenha o desempenho acordado, alocando a quantidade de recursos necessários, independente da variação na demanda. Para isso, o Analisador de Previsão determina se o uso de CPU dos microsserviços implantados é aceitável ou não através da proporção (P):

$$P = \frac{Previsão}{LimiteEstabelecido} \tag{3.4}$$

Onde,

- *Previsão* é o valor da métrica no futuro;
- *LimiteEstabelecido* é o valor definido por microsserviço, *e.g.,* 80% de consumo de CPU.

O *LimiteEstabelecido* é um parâmetro específico do contexto de cada aplicação. Na literatura, essa variação é observada em experimentos com microsserviços. Por exemplo, esse parâmetro pode ser atribuído com um valor presente no intervalo de 40% até 50% (Klinaku *et al.*, 2018; Zhang *et al.*, 2018) ou entre 80% até 90% (Alipour & Liu, 2017; Al-Dhuraibi *et al.*, 2017; Abdullah *et al.*, 2019). No geral, o valor do *LimiteEstabelecido* tem como principal objetivo atender aos requisitos especificados pelo desenvolvedor.

O valor de P é usado para classificar o estado do microsserviço considerando um *LimiteInferior* e um *LimiteSuperior*. Se P < *LimiteInferior* isso indica que o microsserviço possui mais réplicas que o necessário. No caso de *LimiteInferior* $\leq$ P $\leq$ *LimiteSuperior*, isto sugere que o microsserviço tem o número ideal de réplicas. Finalmente, P > *LimiteSuperior* indica que o microsserviço contém menos réplicas do que o necessário.

3.5 PLANEJADOR

O Planejador define um conjunto de ações para modificar as MBAs. Ele define estas ações a partir da análise do Relatório de Análise, gerado pelo Analisador de Previsões. O Relatório de Análise contém o estado atual de cada microsserviço implantado na FCC. Além disso, o Planejador precisa verificar a quantidade de réplicas em execução por microsserviço. Essa informação é obtida pelo Monitor através de um cliente da API de Gerenciamento (Seção 3.3).

As ações planejadas dependem do cenário encontrado, seja ele o excesso ou a insuficiência de réplicas por microsserviço. No planejador, essas ações são determinadas pelo elemento Construtor. O objetivo do Construtor, portanto, é propor um plano de adaptação que otimize o uso dos recursos e mantenha a QoS. Na prática, ele determina quais adaptações serão aplicadas na MBA, de maneira que cada microsserviço opere com a quantidade "ideal" de réplicas. O número ideal de réplicas (NIR) é calculado na Equação (3.5). Como resultado, esse elemento gera uma lista de ações contendo novas implantações e/ou remoções.

$$NIR = CEIL(NAR \times P), \tag{3.5}$$

Onde,

- *CEIL* é uma função que retorna o menor inteiro maior ou igual ao número fornecido;
- *NAR* é o número atual de réplicas;
- P é calculado como mostrado na Equação (3.4).

Além do Construtor, o Planejador implementa módulos preventivos que são usados no processo de adaptação: Contenção de Adaptações Sucessivas e Quarentena. O princípio básico destes módulos é verificar a real necessidade de uma adaptação planejada, ou seja, garantir que a violação detectada não é uma consequência de adaptações recentes e nem é um estado transitório de um determinado microsserviço.

Contenção de Adaptações Sucessivas é um módulo que bloqueia, por um intervalo de tempo (*e.g.,* três minutos), a ocorrência de adaptações sucessivas idênticas para o mesmo microsserviço, *e.g.,* o aumento do número de réplicas. Além disso, ele permite que cada operação disponível no Planejador seja bloqueada paralelamente para um microsserviço específico. Por exemplo, o microsserviço A é bloqueado por dois minutos após ser adaptado por meio da ação de remoção de réplicas, enquanto o microsserviço B é bloqueado por três minutos após ser adaptado aplicando a operação de inserção de réplicas. Nesse caso, vale ressaltar que, depois de bloqueados, os microsserviços A e B ficam indisponíveis por dois e três minutos, respectivamente. Esse processo é detalhado no Algoritmo 2.

Algoritmo 2: Algoritmo do módulo de Contenção de Adaptações Sucessivas.

```
Entrada: estadoMicrosservico, ultimaAdaptacao, tempoBloqueioMuitasReplicas
         (TBMR), tempoBloqueioPoucasReplicas (TBPR)
Saída: bloqueadoContecao
1  início
2      /* Microsserviço operando com muitas réplicas?      */
3      se estadoMicrosservico = 1 então
4          se (tempoAtual - ultimaAdaptacao) > TBMR então
5              ultimaAdaptacao = 0;
6              bloqueadoContecao = "O elemento está disponível para ser adaptado."
7      /* Microsserviço operando com poucas réplicas?      */
8      senão se estadoMicrosservico = -1 então
9          se (tempoAtual - ultimaAdaptacao) > TBPR então
10             ultimaAdaptacao = 0;
11             bloqueadoContecao = "O elemento está disponível para ser adaptado."
12     retorna bloqueadoContecao
13 fim
```

A Quarentena é um módulo de segurança aplicável para lidar com cenários que sofrem com violações transientes. Esse mecanismo de contenção avalia uma sequência de violações idênticas para determinar a tomada de decisão. O mecanismo assume que a ocorrência de uma mesma violação repetidamente aumenta a probabilidade de que o problema não seja transiente e, portanto, a adaptação deve ser realizada. O número de avaliações idênticas necessárias para confirmar uma violação é um parâmetro configurável pelo desenvolvedor da MBA, *e.g.,* três percepções da mesma violação consecutivas indicam uma falha real. Assim como a Contenção de Adaptações Sucessivas, esse módulo é usado paralelamente para um microsserviço e uma operação específica. Esse processo é detalhado no Algoritmo 3.

Algoritmo 3: Algoritmo do módulo Quarentena.

Entrada: valorNaQuarentena, NúmeroVerificacoesNecessarias
Saída: bloqueadoQuarentena

```
1 início
2     se valorNaQuarentena < NúmeroVerificacoesNecessarias então
3         bloqueadoQuarentena = "Microsserviço inserido na quarentena."
4     senão
5         bloqueadoQuarentena = "Microsserviço apresentou um problema não
            transitório e está fora da quarentena."
6     retorna bloqueadoQuarentena
7 fim
```

Finalmente, o ML-Adapt pode operar com ambos, somente um ou nenhum módulo preventivo.

3.6 EXECUTOR

O Executor é projetado para aplicar o plano de adaptação, uma sequência de comandos de alto nível, através de requisições efetuadas à API de Gerenciamento. Os comandos de alto nível são ações descritas em linguagem natural, *e.g.,* aumente uma réplica do microsserviço A. O executor, portanto, precisa transformar os comandos de alto nível em comandos de baixo nível compreensíveis a FCC. Dessa forma, o ML-Adapt exige um executor para cada FCC.

Portanto, foi implementado um mecanismo que recebe uma informação de alto nível (*e.g.,* diminua três réplicas do microsserviço B), modifica o dado para o padrão da FCC e executa uma solicitação adaptativa a API de Gerenciamento. A FCC, por sua vez, aplica as alterações atualizando o número de réplicas por microsserviço. Além disso, a API de Gerenciamento informa ao Executor o resultado da ação, *e.g.,* falha, sucesso.

3.7 SISTEMA GERENCIADO

O Coletor é uma aplicação encarregada de monitorar, armazenar e oferecer informações históricas dos componentes que compõem o Sistema Gerenciado ao Monitor. No contexto de MBAs, esse conjunto de recursos são oferecidos por bancos de dados, *e.g.,* Prometheus[1] e InfluxDB[2]. A atividade de monitoramento do Coletor verifica a existência de outras aplicações que disponibilizam métricas da FCC, *e.g.,* Metrics Server[3], Node Exporter[4] e cAdvisor[5]. A FCC é uma aplicação usada para facilitar o gerenciamento dos microsserviços, *e.g.,* Kubernetes[6] ou o

[1] https://github.com/coreos/kube-prometheus
[2] https://github.com/influxdata/influxdb
[3] https://github.com/kubernetes-sigs/metrics-server
[4] https://github.com/prometheus/node_exporter
[5] https://github.com/google/cadvisor
[6] https://github.com/kubernetes/kubernetes

Docker Swarm[7].

O ML-Adapt é um ambiente capaz de se adequar a quaisquer das tecnologias mencionadas. Porém, na avaliação conduzida por esta dissertação o Kubernetes foi selecionado como FCC. O Kubernetes é uma solução de código aberto, utilizada para automatizar a implantação, dimensionamento e gerenciamento de aplicações que são distribuídas em contêineres. Logo, sua função nessa arquitetura, é simplificar a implantação do Coletor e das MBAs. Além disso, possui uma API, que é consumida pelos componentes Monitor e Executor. De modo geral, a escolha da FCC se dá devido a MBA usada nos experimentos (Capítulo 4). A implementação atual da MBA só permite a implantação na FCC Kubernetes. O Coletor, por sua vez, consiste em aplicações padrões de monitoramento do Kubernetes, o Prometheus e o Metrics Server.

O Prometheus é uma aplicação de código aberto para monitoramento e alerta de sistemas. O Metrics Server coleta e agrega métricas relacionadas ao consumo de recursos dos *pods* e nós do Kubernetes. Essas informações são disponibilizadas pelo *Kubelet*, um serviço implantado em cada nó de um *cluster* Kubernetes. As métricas são instantâneas e, portanto, não é possível obter o histórico de recursos utilizados por um determinado contêiner. Para resolver esse problema, o Metrics Server expõe um conjunto de funcionalidades por meio de uma API. Em seguida, os dados são consumidos e armazenados pelo Prometheus.

Finalmente, as aplicações do Sistema Gerenciado não estão intrinsecamente associadas ao ML-Adapt. Ele é um ambiente genérico que pode ser adaptado para atender outras tecnologias já mencionadas. Porém, ao optar por essas tecnologias, é necessário especificar a origem e a função desses componentes.

3.8 CONSIDERAÇÕES FINAIS

Este capítulo apresentou todos os componentes do ML-Adapt. Primeiramente, foram detalhados cada elemento com o auxílio de uma visão geral e um fluxo de trabalho. Em seguida, o ML-Adapt foi detalhado em um conjunto de seções em conformidade com as fases do *feedback loop* MAPE-K. Por fim, foram apresentados os componentes do Sistema Gerenciado.

[7]https://github.com/docker/swarm

4 EXPERIMENTOS E RESULTADOS

Este capítulo apresenta as etapas de avaliação de desempenho do ML-Adapt. O capítulo apresenta inicialmente uma descrição dos experimentos realizados. Em seguida, a Seção 4.2 mostra os resultados encontrados e uma análise dos experimentos.

4.1 EXPERIMENTOS

A metodologia usada para a construção dos experimentos realizados nesse trabalho consiste dos seguintes passos (Jain, 1990):

1. Definição dos objetivos: o objetivo é comparar o ML-Adapt com o Horizontal Pod Autoscaler (HPA), um recurso disponível na API do Kubernetes popularmente usado na indústria. Ambas as soluções gerenciam o consumo de CPU dos microsserviços implantados em uma FCC.

2. Apresentação dos serviços do sistema: o ML-Adapt e o HPA gerenciam a QoS das MBAs através do dimensionamento automático. O dimensionamento automático é um processo que monitora a demanda no consumo de recursos (*e.g.,* memória, rede e CPU) dos microsserviços para intervir, conforme necessário, com ações que aumentam ou diminuem a quantidade de réplicas/recursos alocados;

3. Escolha das métricas de desempenho: as métricas de desempenho consideradas foram o Round Trip Time (RTT)[1] da MBA, o consumo de CPU e memória da FCC. Além disso, a quantidade de intervenções, ações que modificam a MBA por meio do dimensionamento automático executadas pelas soluções também são analisadas.

4. Listar os parâmetros:

    - Os parâmetros do sistema foram a *quantidade de VMs*, *processador* utilizado, *tamanho da memória* e o *sistema operacional*, como apresentado na Tabela 3;

Tabela 3: Parâmetros do sistema.

Parâmetro	Valor
Quantidade de VMs	6
Processador	Intel Xeon 1220 V6 quad-core 3 gigahertz
Tamanho da memória	8 gigahertz (DDR3)
Sistema operacional	Ubuntu Linux 18.04

[1] É a duração em milissegundos necessária para que uma solicitação de um cliente a uma MBA seja processada e devolvida.

- Os parâmetros da carga de trabalho considerados foram o *ambiente adaptativo*, o *modo de operação* e a *carga de trabalho*. O ambiente adaptativo é o software que gerencia o consumo de CPU dos microsserviços, ou seja, o ML-Adapt e o HPA. O ML-Adapt é capaz de ser configurado com quatro diferentes modelos de AM: MLP, RF, SVR e XGBoost. Sendo assim, o ML-Adapt contém quatro versões do mesmo ambiente, onde cada modelo representa uma variação. O modo de operação refere-se aos módulos preventivos, Contenção de Adaptações Sucessivas e Quarentena, disponíveis apenas no ML-Adapt (Seção 3.5). Por fim, a carga de trabalho refere-se a um conjunto de solicitações à MBA.

5. Listar fatores: foram considerados o *ambiente adaptativo*, o *modo de operação* e a *carga de trabalho*. Os valores dos fatores são apresentados na Tabela 3. Porém, vale ressaltar que, o fator *modo de operação* não é considerado nos casos em que o *ambiente adaptativo* é o HPA ou Base, configuração que não efetua adaptações durante o experimento.

Tabela 4: Fatores.

Parâmetro (Fator)	Níveis
Carga de Trabalho	Aleatória 1, Aleatória 2, Alta, Média e Baixa
Ambiente Adaptativo	MLP, RF, SVR, XGBoost, HPA e Base
Modo de Operação	Nenhum, Contenção de Adaptações Sucessivas, Quarentena e Ambos.

6. Escolher a técnica de avaliação: as métricas avaliadas nesta pesquisa são obtidas através da medição;

7. Escolher a carga de trabalho:

   - Tipo da carga: foram utilizados dois diferentes tipos de cargas de trabalho na avaliação: fixas e variáveis. Uma carga de trabalho fixa significa que o número de clientes e o tempo alocado para solicitações não muda em todo o experimento, *e.g.,* 50 clientes fazem solicitações simultâneas a uma MBA por 150 minutos. Por outro lado, uma carga de trabalho variável significa que o número de clientes e o tempo alocado para solicitações mudam enquanto o experimento executa, *e.g.,* 50 clientes simultâneos são executados por 15 minutos, seguidos por 150 clientes simultâneos em execução por 25 minutos. O número de clientes foi definido para caracterizar cargas de trabalho baixas (50), médias (100) e altas (150);
   - Cenários: cinco cenários diferentes foram definidos - três cenários com cargas de trabalho fixas e dois cenários com cargas de trabalho variáveis.

As três cargas de trabalho fixas foram configuradas com 50 (Fixo 1), 100 (Fixo 2) e 150 (Fixo 3) clientes. Nos dois cenários variáveis (Variável 1 e Variável 2), o número de clientes é escolhido aleatoriamente entre 50, 100 e 150 clientes. A quantidade de clientes por cenário foi definida conforme experimentos prévios que analisaram o impacto na MBA. Como resultado, foi constatado que 50, 100 e 150 clientes representavam, respectivamente, um baixo, médio e alto consumo de recursos dos microsserviços da MBA;

- Tempo do experimento: o tempo total alocado para solicitações define a duração do experimento. Em ambos os tipos de cenários (fixos e variáveis), o tempo total alocado para solicitações foi definido como 300 minutos. No entanto, em cenários variáveis, como o número de clientes não é fixo ao longo do experimento, o tempo alocado para solicitações de cada cliente é definido como 10 minutos. Esse tempo foi definido de acordo com valores utilizados em Zhang *et al.* (2018). A Tabela 5 resume os diferentes cenários avaliados;
- Gerador de carga de trabalho: o Locust é um software para geração de carga de trabalho que processa uma sequência de solicitações, descritas na linguagem Python, para uma aplicação de destino. Para configurar o Locust foram utilizadas as informações de número de clientes e o tempo alocado para as solicitações.

Tabela 5: Informações sobre a carga de trabalho.

Cenário	Carga de Trabalho	Clientes Simultâneos	Tempo Alocado (Minutos)
Fixo 1	Baixa	50	300
Fixo 2	Média	100	300
Fixo 3	Alta	150	300
Variável 1	Aleatória 1	100, 100, 150, 100, 100, 50, 50, 150, 150, 150, 50, 50, 50, 50, 50, 50, 150, 100, 150, 150, 100, 50, 100, 100, 150, 150, 50, 50, 100, 150	10
Variável 2	Aleatória 2	150, 150, 150, 150, 50, 150, 150, 100, 150, 50, 150, 150, 100, 100, 50, 150, 100, 50, 50, 100, 50, 150, 150, 100, 100, 50, 100, 100, 50, 50	10

8. Projetar o experimento:

- Ambiente: todos os componentes eram reinicializados a cada experimento. A reinicialização foi uma estratégia para evitar que a sobrecarga de um experimento estivesse presente nos demais.

- Coleta de métricas: foi utilizado um *script* desenvolvido para calcular o RTT. Os valores das métricas memória, CPU e a quantidade de intervenções aplicadas na MBA eram extraídos posteriormente através do Prometheus;
- Tratamento das métricas: foram coletadas 194 amostras da métrica RTT. A quantidade de amostras é resultado da configuração do *script* que solicita a MBA por trinta segundos e aguarda sessenta segundos para uma nova coleta. As métricas analisadas CPU, memória e quantidade de intervenções do ambiente adaptativo produzem 300 amostras, ou seja, uma amostra por minuto. Em seguida, calcula-se a média de cada métrica. A quantidade de intervenções é calculada observando uma série temporal que contém o número de microsserviços em execução ao longo do experimento.

9. Analisar e interpretar os resultados: a análise e interpretações de dados é produzida através de Tabelas e Gráficos que comparam o ML-Adapt com o HPA. Esses elementos possuem informações sobre o desempenho dos ambientes adaptativos e das suas métricas de desempenho como consumo de CPU e memória.

A aplicação utilizada para comparar o desempenho das soluções adaptativas é o Hispter Shop. O Hipster Shop[2] é uma MBA de código aberto composta por 10 microsserviços implementados em diferentes linguagens de programação. A comunicação entre os microsserviços é realizada através do protocolo gRPC[3]. A aplicação simula as vendas de produtos em um site de comércio eletrônico. Além disso, é usada pelo Google para demonstrar tecnologias como Kubernetes, Istio e StackDrivers. Esta aplicação foi escolhida devido ao seu uso como *testbed* em outras pesquisas envolvendo microsserviços (Yu *et al.*, 2019; Li *et al.*, 2019).

Os modelos preditivos do ML-Adapt foram criados através do monitoramento do consumo histórico de CPU dos microsserviços que compõem o Hipster Shop. Os dados obtidos foram normalizados e transformados em séries temporais compostas por 4320 pontos. Essas séries temporais são processadas com o retardo temporal de vinte *lags* conforme definido na Tabela 2. Além disso, a série foi dividida em três subamostras: 60% foi utilizado para treinamento (2592 pontos), 20% para validação (864) e 20% para teste (864). Essas porcentagens foram definidas de acordo com (Wong, 2018).

Dentro dos ambientes adaptativos existem parâmetros a serem configurados para a execução dos experimentos. Esses parâmetros foram definidos de acordo com os valores adotados pelo HPA do Kubernetes. A Tabela 6 resume os valores adotados por parâmetro. Finalmente, considerando a Tabela 4 foram realizados noventa experimentos.

[2]https://github.com/GoogleCloudPlatform/microservices-demo
[3]https://grpc.io/

Tabela 6: Parâmetros dos ambientes adaptativos.

Parâmetros	Fase do MAPE-K	ML-Adapt	HPA	Base
Intervalo entre coletas	Monitor	15 segundos	15 segundos	-
LimiteInferior	Analisador	0,9	0,9	-
LimiteSuperior	Analisador	1,1	1,1	-
LimiteEstabelecido	Analisador	80%	80%	-

4.2 RESULTADOS

As análises dos resultados dos experimentos são apresentadas como segue: na Seção 4.2.1 são comparadas as ferramentas HPA do Kubernetes e o ML-Adapt observando a métrica RTT. Em seguida, fundamentado nos resultados da Seção 4.2.1, são discutidas as métricas de desempenho CPU (Seção 4.2.2) e memória (Seção 4.2.3). Por fim, na Seção 4.2.4, são abordados os módulos preventivos usados no ML-Adapt.

4.2.1 Round Trip Time

Na análise da métrica RTT, há experimentos como no caso da configuração Base, que não usam nenhum software adaptativo, e outros que usam, como nos casos do HPA do Kubernetes, e das variações de configuração do ML-Adapt (MLP, RF, SVR e XGBoost). Além disso, cada conjunto de experimentos é nomeado conforme a sua carga de trabalho, ou seja, Aleatória 1, Aleatória 2, Alta, Média e Baixa.

A Tabela 7 apresenta, em milissegundos, os resultados da métrica RTT dos ambientes adaptativos. Novamente, o ML-Adapt apresenta quatro variações no fator modo de operação (Ver Tabela 4). Por exemplo, ML-Adapt configurado com o SVR é executado com ambos, somente um ou nenhum módulo preventivo na carga variável Aleatória 1. Como consequência, cada configuração do ML-Adapt detém quatro resultados por carga de trabalho. Por causa disso, o resultado da métrica RTT apresentado para o ML-Adapt configurado com qualquer ambiente adaptativo é aquele que obteve o maior desempenho nas variações do modo de operação. Vale salientar que todos os experimentos realizados se encontram ordenados pelo RTT no Apêndice A.

Tabela 7: Melhores resultados do RTT (ms).

Carga de Trabalho	SVR	MLP	HPA	RF	XGBoost	Base
Aleatória 1	185,966	**165,629**	192,086	774,480	837,291	1083,463
Aleatória 2	**138,874**	166,040	144,171	748,371	859,669	817,417
Alta	**220,789**	431,434	546,297	1600,903	1634,794	1723,109
Média	**146,914**	309,417	328,634	728,623	938,257	1164,731
Baixa	**114,874**	192,343	205,114	390,686	426,594	580,629

A Figura 13 mostra que o SVR obtém os melhores resultados com cargas de trabalho fixas. Nas cargas de trabalho Alta e Média, o desempenho do ML-Adapt configurado com

SVR é 59,58% e 55,30% superior ao HPA, respectivamente. Além disso, na carga de trabalho variável Aleatória 1, o ML-Adapt opera com um desempenho superior ao HPA de 13,77% quando comparado ao MLP e 3,19% comparado ao SVR. Na segunda carga de trabalho variável (Aleatória 2), ML-Adapt também tem um desempenho superior ao HPA quando configurado com o SVR de 3,67%. No entanto, na mesma carga de trabalho, o ambiente proposto é inferior ao HPA quando configurado com o MLP em 15,17%.

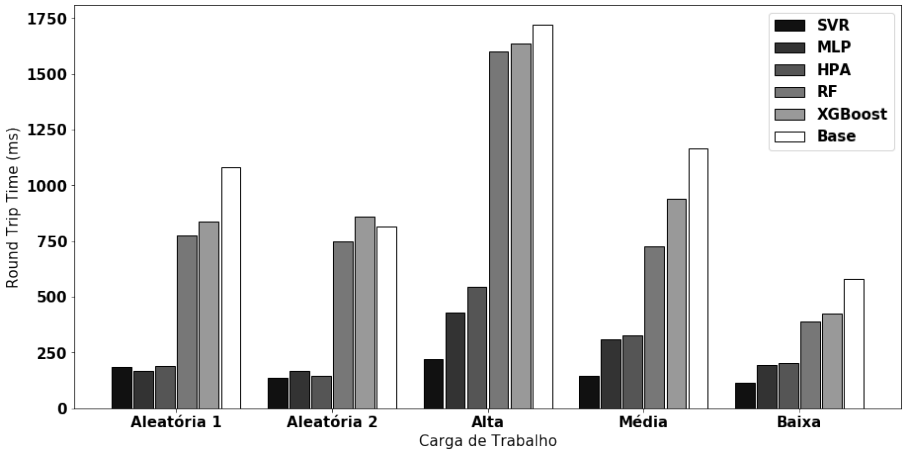


Figura 13: *Round Trip Time* médio.

Ao avaliar o HPA, é possível observar que este não opera bem com cargas de trabalho fixas. Para entender melhor esse resultado, a Tabela 8 apresenta o número de adaptações (*scaling-in/scaling-out*) executadas por cada configuração. Se o comportamento do HPA for observado ao longo das cargas de trabalho, é possível perceber que em cargas de trabalho variáveis (Aleatória 1 (100) e Aleatória 2 (87)), ele executa uma quantidade maior de adaptações quando comparado às fixas (Baixa (5), Média (5) e Alta (5)). No entanto, espera-se que em cargas de trabalho fixas como a Alta, onde a carga de trabalho aplicada tende a consumir muitos recursos, o ambiente adaptativo execute uma quantidade maior de adaptações, mas isso não ocorre nestas cargas de trabalho. Ao avaliar as mesmas informações coletadas do ML-Adapt configurado com o SVR, percebe-se que ele realizou uma quantidade superior de adaptações (102). Esse fato indica que deveria ter ocorrido um número maior de adaptações a serem executadas pelo HPA, mas elas não foram detectadas. Como o SVR conseguiu percebê-las, ele obteve um melhor RTT, como já mencionado.

Tabela 8: Número de adaptações executadas por cada configuração.

Carga de Trabalho	HPA	MLP	SVR
Aleatória 1	87	221	337
Aleatória 2	100	254	232
Alta	5	49	102
Média	5	21	111
Baixa	5	18	56

As configurações RF e XGBoost não apresentaram um bom desempenho nas cargas de

trabalho avaliadas. Por exemplo, na carga de trabalho variável Aleatória 2, o desempenho do XGBoost é 5,17% inferior a configuração Base. Além disso, o ML-Adapt configurado com o RF é superior ao XGBoost e ao caso da configuração Base em todas as cargas de trabalho. Quando o ML-Adapt configurado com o RF é comparado com a configuração Base, ele apresenta um desempenho superior de 28,52% na Aleatória 1, 8,45% na Aleatória 2, 7,09% na Alta, 37,44% na Média e 32,71% na Baixa.

Por outro lado, ainda que os resultados alcançados pelo RF e o XGBoost comparados com a configuração Base pareçam promissores, quando comparados ao HPA e o ML-Adapt configurado com o SVR, a diferença de desempenho entre essas configurações é evidente. Por exemplo, na Aleatória 2, o HPA tem um desempenho 419,09% superior ao RF e 496,28% superior ao XGBoost. Ao mesmo tempo, o SVR tem um desempenho de 438,88% superior ao RF e 519,03% superior ao XGBoost.

Para resumir, quando ML-Adapt usa SVR, ele tem um desempenho superior ao HPA nas cargas de trabalho fixas: 59,58% (Alta), 55,30% (Média) e 44,00% (Baixa). Assim como, o MLP apresenta melhores resultados do que o HPA para as mesmas cargas de trabalho (21,03%, 5,85%, 6,23%). Em cargas de trabalho variáveis, apesar da aproximação nos resultados, o HPA não é superior ao SVR em nenhum deles. Porém, o HPA aciona menos adaptações na MBA. Em relação ao RF e o XGBoost, apesar de serem adotados em muitos problemas na aplicação de séries temporais (Chen & Guestrin, 2016; Grohmann *et al.*, 2019), eles não se sobressaíram na previsão do consumo de CPU dos microsserviços do Hipster Shop. Ao produzir previsões de baixa qualidade que não expressam o estado futuro da MBA, conduziram o ML-Adapt a executar adaptações incorretas/desnecessárias. Portanto, essa análise destaca a necessidade de realizar uma boa avaliação na escolha dos algoritmos de AM.

4.2.2 Processamento

A Figura 14 apresenta o consumo de CPU médio da MBA. Vale salientar que todos os experimentos realizados se encontram ordenados pelo consumo de CPU no Apêndice D. Ao analisar a Figura 14 é possível observar que cargas de trabalho variáveis utilizam mais recursos de infraestrutura do que as fixas. Novamente, a Tabela 8 ajuda a explicar por que esse fato acontece. Quando uma carga de trabalho fixa é usada (Baixa, Média, Alta), o número de adaptações é muito menor quando comparado às variáveis (Aleatória 1 e 2). Assim, a quantidade de recursos alocados e desalocados pela configuração é menor e o consumo de CPU, consequentemente, tende a diminuir.

A obtenção de melhores resultados para a métrica RTT gera uma utilização maior de CPU. Por exemplo, nas cargas de trabalho Aleatória 1 (MLP), Alta (SVR) e Média (SVR), os melhores resultados por carga de trabalho para a métrica RTT consumiram mais recursos. Por outro lado, um resultado inusitado ocorre na Aleatória 1. Apesar do ML-Adapt configurado com o MLP não obter um bom desempenho para a métrica RTT, quando comparado ao HPA e ML-

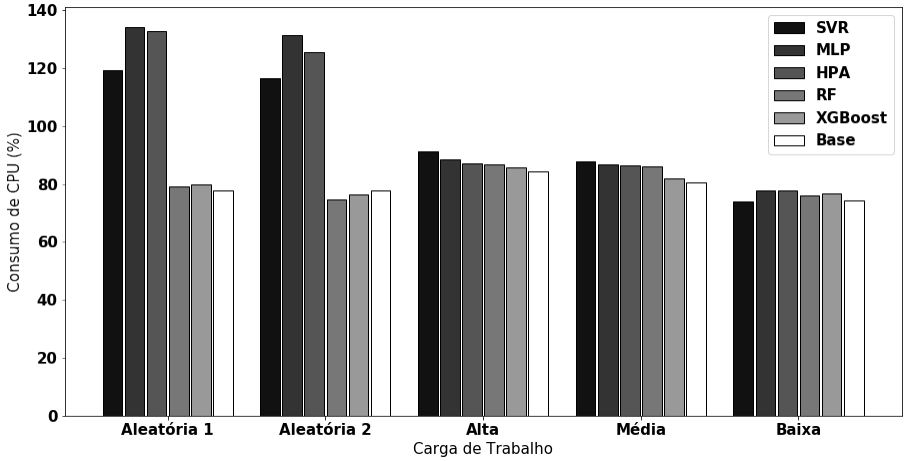


Figura 14: Consumo médio de CPU.

Adapt configurado com o SVR. Nesse grupo de experimentos, ele é ainda a configuração com o maior consumo de CPU. Novamente, esse fato é explicado pela quantidade de intervenções efetuadas pelo ambiente adaptativo. Como observado na Tabela 8, quando o MLP (254) é comparado ao HPA (100), é perceptível uma quantidade maior de adaptações do MLP.

Uma análise apenas da quantidade de intervenções pode mostrar que o HPA é mais eficiente no processo de alocação de recursos. A eficiência de uma configuração significa a capacidade de alocar a "quantidade necessária de recursos", de acordo com a variação da demanda, executando o menor número de intervenções possíveis. No entanto, essa afirmação não é verdadeira porque na mesma carga de trabalho (Aleatória 2), o SVR consumiu 10,17% a menos de CPU e obteve um desempenho de 4,84% superior para a métrica RTT do que o HPA. Portanto, o número de intervenções somente gera um problema de desempenho quando a configuração, seja ela reativa ou proativa, não consegue perceber corretamente as demandas. Dessa forma, a percepção incorreta leva à geração de planos adaptativos que não são consistentes com as condições futuras da aplicação, deteriorando o desempenho da MBA.

Como não é possível alocar recursos adicionais para atender a demanda na configuração Base, o consumo de CPU tende a permanecer próximo do mínimo alocado. Quando ML-Adapt é configurado com RF e XGBoost, o comportamento é semelhante. Como essas configurações não funcionam bem nas cargas de trabalho consideradas, como mencionado na Seção 4.2.1, elas não usam completamente os recursos disponíveis e, consequentemente, levam a um maior RTT.

Para resumir, assim como esperado, os dados mostram que para uma configuração obter melhores resultados para a métrica RTT foi necessário um maior consumo de CPU como evidenciado nas cargas de trabalho Aleatória 1 (MLP), Alta (SVR) e Média (SVR). Além disso, a carga variável Aleatória 2 demonstrou que o consumo de CPU não está diretamente relacionado à quantidade de adaptações efetuadas pela configuração. Por fim, mesmo que haja recursos disponíveis, as configurações RF e o XGBoost não são capazes de usufruir dos recursos adicionais.

4.2.3 Memória

A Figura 15 apresenta a quantidade de memória média consumida pela MBA. Além disso, todos os experimentos efetuados encontram-se ordenados pela métrica de Consumo de Memória no Apêndice F. Ao analisar a Figura 15 observa-se que, diferentemente do consumo de CPU, cargas de trabalho fixas e variáveis consomem memória sem um padrão específico. Por exemplo, na carga de trabalho fixa Alta, o HPA teve um consumo de memória superior ao ML-Adapt configurado com o SVR de 1,64%, enquanto seu desempenho (RTT) foi muito inferior (59,58%). Em contrapartida, na carga de trabalho variável Aleatória 2, o consumo de memória do HPA é 17,97% inferior ao SVR, enquanto seu desempenho (RTT) foi apenas 3,67% inferior.

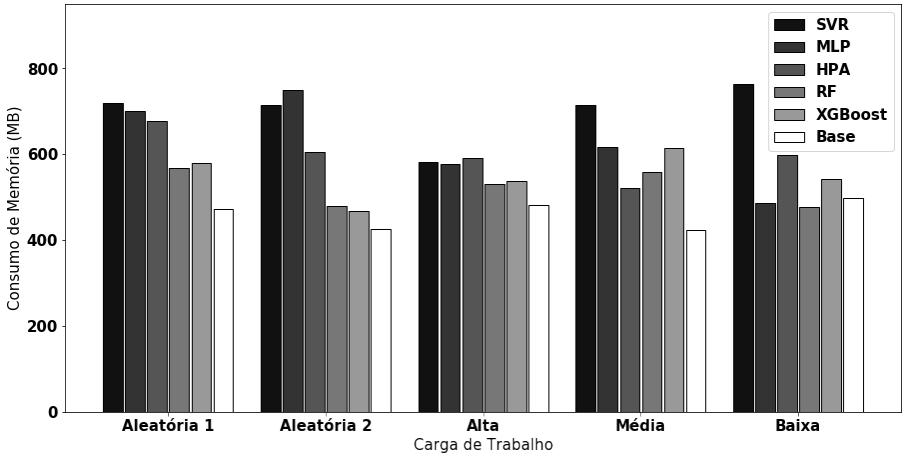


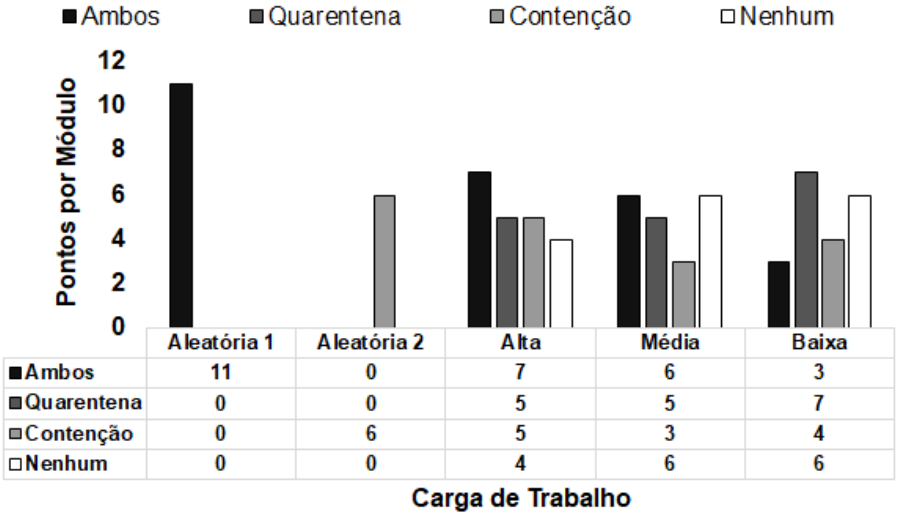
Figura 15: Consumo médio de memória.

A métrica memória está menos relacionada à métrica RTT do que o consumo de CPU, quando se trata das configurações RF e o XGBoost. Assim como no consumo de CPU, as cargas de trabalho com a configuração Base tendem a consumir apenas a quantidade de recursos alocados. No entanto, RF e XGBoost variam entre as cargas de trabalho devido às suas habilidades para adaptar a MBA. Por exemplo, na carga de trabalho variável Aleatória 1, o XGBoost tem um consumo de memória 22,52% superior à configuração Base, enquanto atinge desempenho (RTT) 22,72% superior. Em contrapartida, na Aleatória 2, ela consome uma quantidade 9,89% superior à configuração Base e ainda possui uma redução no RTT da MBA de 5,17%.

4.2.4 Módulos Preventivos do ML-Adapt

O objetivo deste experimento é comparar o impacto dos módulos preventivos - Contenção de Adaptações Sucessivas e a Quarentena - presentes no ML-Adapt. A Figura 16, apresenta a pontuação de um determinado módulo nas cargas de trabalho fixas e variáveis. Inicialmente, são selecionados os experimentos em que o ML-Adapt utilizando qualquer configuração foi superior ao HPA. Em seguida, os módulos preventivos são pontuados conforme a posição em que se encontram. As informações deste processo estão disponíveis no Apêndice G.

A Figura 16 demonstra que o ML-Adapt em cargas de trabalho variáveis obtém um melhor desempenho usando módulos preventivos específicos. Por outro lado, em cargas de trabalho fixas todos os modos de operação se destacam. A razão para isso é que o ML-Adapt apresenta um desempenho superior ao HPA em cargas de trabalho fixas (Seção 4.2.1). Logo, é esperado um número maior de modos de operação nestas cargas. Em cargas variáveis, por outro lado, apenas os experimentos com ambos os módulos e a Contenção de Adaptações Sucessivas pontuaram.



	Aleatória 1	Aleatória 2	Alta	Média	Baixa
Ambos	11	0	7	6	3
Quarentena	0	0	5	5	7
Contenção	0	6	5	3	4
Nenhum	0	0	4	6	6

Figura 16: Pontuação dos módulos preventivos.

Nas cargas de trabalho fixas, o uso de nenhum módulo preventivo ou ambos os módulos são as melhores estratégias na carga de trabalho fixa Média. Além disso, o uso de nenhum módulo preventivo apresenta também uma boa pontuação nas cargas fixas Alta e Baixa. Esses resultados não são inesperados porque a Contenção de Adaptações Sucessivas e a Quarentena são módulos aplicados para conter episódios transientes, algo incomum neste tipo de carga de trabalho. No geral, mesmo que a quantidade de cargas de trabalho fixas seja superior as variáveis, o uso de ambos os módulos preventivos no ML-Adapt é a melhor configuração presente em 34,62% dos experimentos superiores ao HPA. A Contenção de Adaptações Sucessivas é usada em 23,08%, enquanto a Quarentena é a configuração de 21,79% dos experimentos. Por fim, o uso de nenhum módulo preventivo é a pior modo de operação com 20,51%. Logo, a aplicação de módulos preventivos no ML-Adapt está presente em 79,49% dos experimentos que foram superiores ao HPA.

4.3 CONSIDERAÇÕES FINAIS

Nesse capítulo foi apresentada a avaliação de desempenho do ML-Adapt. Inicialmente, foi seguida uma metodologia para elaboração dos experimentos. Em seguida, os experimentos foram realizados variando os níveis expostos. Os resultados demonstraram que o desempenho do ML-Adapt configurado com SVR ou MLP é superior ao HPA na maior parte dos experimentos. E que as configurações RF e XGBoost não alcançam bons resultados no Hipster Shop. Por outro

lado, o HPA é uma configuração menos intervencionista. A métrica de CPU é mais consumida para alcançar um melhor desempenho, enquanto a memória não demonstra um padrão específico de uso. Por fim, a configuração de ambos os módulos preventivos ou apenas a Contenção de Adaptações Sucessivas no ML-Adapt demonstram serem as melhores alternativas.

5 TRABALHOS RELACIONADOS

Este capítulo apresenta os trabalhos encontrados na literatura relacionados ao contexto desta dissertação. A Seção 5.1 descreve soluções adaptativas que lidam com o gerenciamento de recursos dos microsserviços. A Seção 5.2 traz problemas genéricos presentes nos microsserviços como localização, comunicação e falhas. A Seção 5.3 apresenta o uso de modelos de AM aplicados na previsão de recursos para auxiliar o processo de tomada de decisão. A Seção 5.4 mostra uma avaliação comparativa entre os trabalhos relacionados e o trabalho proposto.

5.1 GERENCIAMENTO DE RECURSOS PARA MICROSSERVIÇOS

Florio & Nitto (2016) adicionam recursos autonômicos aos microsserviços sem alterar a maneira como eles são implementados. Para esse fim, os autores propõem uma solução nomeada de Gru. O Gru implanta um Gru-Agent em cada nó do *cluster*. Este agente é um gerenciador de nó independente, consistindo em um *feedback loop* MAPE-K descentralizado para evitar um único ponto de falha. Cada Gru-Agent é capaz de adaptar MBAs, replicando ou migrando microsserviços, levando em consideração as variações de carga de trabalho e as informações fornecidas por outros Gru-Agents.

CAUS (Klinaku *et al.*, 2018) é um processo de adaptação heurística que permite a elasticidade de um microsserviço em um contêiner particular. Essa abordagem lida com alterações na carga de trabalho. Além disso, contém um mecanismo que mantém um *buffer* de contêineres adicionais para lidar com mudanças imprevisíveis na carga de trabalho. A solução adaptativa está estruturada como um *feedback loop* MAPE-K centralizado. O CAUS monitora periodicamente a carga de trabalho para encontrar e corrigir violações, aumentando ou diminuindo o número de réplicas dos microsserviços, conforme os limites estabelecidos.

Zhang *et al.* (2018) exploram a necessidade de novos requisitos para soluções autônomas que gerenciam MBAs e propõem um modelo de referência. Esse modelo é a base para uma estrutura auto adaptativa chamada MSSAF. A arquitetura do MSSAF consiste em três componentes: uma solução adaptativa, um conjunto de ferramentas de tradução e um sistema gerenciado. A solução adaptativa opera sobre um *feedback loop* MAPE-K centralizado. As ferramentas de tradução interpretam estratégias de adaptação. Por fim, o sistema gerenciado é qualquer MBA.

As pesquisas mencionadas nesta seção, monitoram constantemente a QoS provida através de métricas de desempenho como o consumo de CPU. Todas as soluções adaptativas propostas são construídas sobre o *feedback loop* MAPE-K. Florio & Nitto (2016) e Klinaku *et al.* (2018) implementam todas as fases desse modelo, porém Zhang *et al.* (2018) não utilizam o elemento *Knowledge*. Além disso, Florio & Nitto (2016) propõem uma implementação distribuída do MAPE-K. Nesse sentido, o ML-Adapt se destaca por tratar-se de um ambiente proativo, enquanto as soluções propostas por Florio & Nitto (2016), Klinaku *et al.* (2018) e Zhang *et al.* (2018) são reativas. Soluções reativas observam somente o consumo de CPU atual dos microsserviços. Dessa forma, é necessário que um determinado problema ocorra para iniciar o planejamento de

adaptações corretivas. O ML-Adapt, por outro lado, coleta, processa e prediz o consumo futuro de CPU e usa essa informação para identificar e lidar com antecedência a possíveis violações.

5.2 ADAPTAÇÕES EM MICROSSERVIÇOS

App-bisect (Rajagopalan & Jamjoom, 2015) é uma ferramenta autônoma para identificar e reparar problemas de tempo de execução em MBAs. Por exemplo, uma falha de um microsserviço após uma atualização. O App-bisect monitora a QoS fornecida pela MBA analisando problemas de desempenho. Por exemplo, o tempo de resposta e a taxa de transferência dos microsserviços que podem levar à falha. Se um problema de desempenho for encontrado no microsserviço atualizado, ele será revertido para uma versão anterior. Da mesma forma, outros microsserviços dependentes do microsserviço com falha, também são revertidos para a versão mais recente, mantendo a compatibilidade geral.

Kookarinrat & Temtanapat (2016) propõem um barramento de mensagens descentralizado que lida com a comunicação entre os microsserviços. MBAs são sistemas compostos por múltiplos serviços isolados que se comunicam entre si, por meio de chamadas remotas. Essa comunicação, portanto, cria uma dependência entre os elementos, *e.g.,* o microsserviço A descarta uma solicitação do cliente porque essa solicitação deveria ser processada pelo microsserviços B que estava indisponível. A solução permite comunicação assíncrona e síncrona sobre o protocolo TCP. O balanceador de carga do seu barramento aplica o algoritmo *round-robin* ao receber as solicitações. Para descobrir o serviço solicitado, a solução provê uma identificação para cada microsserviço.

REMaP (Sampaio *et al.*, 2019) é uma solução adaptativa autônoma que gerencia a localização dos microsserviços dentro do *cluster*. O REMaP é construído sobre um *feedback loop* MAPE-K centralizado e aplica conceitos de *Models@run.time*. O processo adaptativo é baseado na afinidade dos microsserviços e no uso de recursos. A afinidade é uma métrica proposta pelo autor e definida como a relação entre dois microsserviços dado o número e tamanho das mensagens trocadas ao longo do tempo. A solução é capaz de identificar microsserviços com alta afinidade e agrupá-los no mesmo servidor físico, por exemplo.

Os trabalhos apresentados nesta seção lidam com questões inerentes aos sistemas distribuídos como a atualização, remoção e adição de microsserviços em tempo de execução e a complexidade de comunicação descentralizada. O ML-Adapt também lida com a adaptação de MBAs. No entanto, difere dessas soluções devido ao tipo de problema abordado. ML-Adapt observa a variação na demanda futura de recursos para dimensionar o número de réplicas em execução. Rajagopalan & Jamjoom (2015) analisam se a atualização de um microsserviço não acarretou a alteração do comportamento da MBA. Kookarinrat & Temtanapat (2016) lidam com problemas de comunicação entre microsserviços. Sampaio *et al.* (2019) investigam a otimização dos recursos disponíveis através da realocação de microsserviços.

5.3 PREDIÇÃO DE RECURSOS

Nikravesh *et al.* (2017) investigam o uso de modelos preditivos de séries temporais para minimizar as violações do acordo de nível de serviço das aplicações na nuvem. A previsão de séries temporais mostra ser uma alternativa à adaptação reativa cuja deficiência é negligenciar o tempo de inicialização das VMs. Nesta pesquisa, os algoritmos SVR e o MLP são usados para criação de modelos preditores da carga de trabalho futura da aplicação. A solução proposta monitora o padrão de carga de trabalho e seleciona o modelo preditivo mais apropriado.

Ajila & Bankole (2016) propõem modelos de AM para prever a locação futura de recursos de uma VM. Diferentemente de Nikravesh *et al.* (2017) que preveem a carga de trabalho da aplicação, Ajila & Bankole (2016) usam uma ampla variedade de métricas, como consumo de CPU, tempo de resposta e taxa de transferência. Neste trabalho, cada métrica contém um modelo preditivo associado. Os algoritmos de AM empregados para criação dos modelos são: Rede Neural Artificial (RNA), Linear Regression (LR) e SVR.

Alipour & Liu (2017) apresentam uma arquitetura de microsserviços que encapsula funções de monitoramento de métricas e a aprendizagem de padrões usando modelos de AM. A arquitetura proposta é uma solução adaptativa que implementa cada componente de software como um microsserviço. Nesta arquitetura, o consumo de CPU dos microsserviços é coletado em tempo de execução, e essas informações são processadas para se tornarem séries temporais. Os autores propõem a construção de modelos de AM usando dois algoritmos: o Multinomial Logistic Regression (MLR) e o LR. A previsão de recursos permite ao gerente autônomo antecipar violações aos requisitos estabelecidos e reagir a mudanças na demanda de recursos usando o dimensionamento automático.

HyScale (Wong, 2018) é uma solução adaptativa que lida com o gerenciamento de recursos das MBAs através do dimensionamento automático. O objetivo desta pesquisa é investigar o impacto no uso de técnicas de escalonamento híbridas, a escala vertical e horizontal de recursos. A adoção de ambos os métodos permite o controle refinado (escala vertical) e a alta disponibilidade (escala horizontal). Além disso, o HyScale usa a análise de séries temporais em um banco de dados de terceiros para construir um modelo de AM capaz de prever o consumo futuro de CPU. O modelo é construído pelo algoritmo Long Short-Term Memory (LSTM). No entanto, as etapas de AM estão fora do processo adaptativo do HyScale. Fora do processo, significa que, embora seja uma solução adaptável que pode lidar com a variação no consumo de recursos usando a escala horizontal e vertical, não é uma solução proativa. Isso acontece porque as etapas que lidam com o algoritmo LSTM não são integradas ao processo adaptativo.

Podolskiy *et al.* (2018) apresentam um estudo comparativo entre um grupo de algoritmos de AM para avaliar a qualidade de suas previsões. Os algoritmos investigados são: Exponential Smoothing (ES), Seasonal Autoregressive Integrated Moving Average (SARIMA), Autoregressive Fractionally Integrated Moving Average (ARFIMA), LR, Singular Spectrum Analysis (SSA) e SVR. Além disso, os algoritmos baseados em Autoregressive Integrated Moving Average

(ARIMA) também são estendidos com Generalized Autoregressive Conditional Heteroskedasticity (GARCH). A construção dos modelos preditivos utiliza dados históricos coletados da carga de trabalho de uma MBA. Esta pesquisa não se concentra na adaptação de MBAs e apresenta apenas uma série de algoritmos existentes aplicáveis. Assim, estes algoritmos são úteis para aplicação e avaliação de seus benefícios em sistemas adaptativos proativos.

O ML-Adapt, assim como Alipour & Liu (2017) e Wong (2018), explora a adaptação em MBAs. O ML-Adapt difere das outras, por ser o único ambiente implementado sobre um *feedback loop* MAPE-K. Além disso, ele aplica algoritmos utilizados por Ajila & Bankole (2016) e Nikravesh *et al.* (2017), o MLP e o SVR. Por outro lado, investiga dois outros algoritmos de AM, o RF e o XGBoost, que não estão presentes em nenhum desses trabalhos. Por fim, apesar da técnica de aprendizagem online e do algoritmo LR não estarem presentes no ML-Adapt, estes podem ser posteriormente integrados.

5.4 AVALIAÇÃO COMPARATIVA ENTRE OS TRABALHOS RELACIONADOS E O TRABALHO PROPOSTO

A Tabela 9 apresenta uma comparação entre os trabalhos relacionados mencionados. Essa tabela é uma versão adaptada de um grupo de questões proposto por Salehie & Tahvildari (2009) para a elicitação dos requisitos de um software adaptativo (Seção 2.1). Essa versão adaptada é composta por cinco questões: onde adaptar, quando adaptar, o que adaptar, por que adaptar e como adaptar. Além disso, a tabela apresenta duas colunas extras para determinar se a pesquisa usa AM ou o MAPE-K.

A Tabela 9 simplifica a visualização de características comuns disponíveis nesses trabalhos relacionados. Por exemplo, é perceptível que cinco soluções (Florio & Nitto, 2016; Alipour & Liu, 2017; Klinaku *et al.*, 2018; Zhang *et al.*, 2018; ML-Adapt) lidam com o gerenciamento de recursos, modificando a quantidade de microsserviços por meio do dimensionamento automático de recursos. Além disso, outras três soluções (Ajila & Bankole, 2016; Nikravesh *et al.*, 2017; Wong, 2018) enfrentam o gerenciamento de recursos, porém não implementam soluções adaptativas, focando apenas na construção de modelos preditivos de AM.

Dessas oito pesquisas, cinco (Ajila & Bankole, 2016; Nikravesh *et al.*, 2017; Alipour & Liu, 2017; Wong, 2018; ML-Adapt) abordam o uso de algoritmos de AM para predição de recursos, enquanto as demais são soluções reativas. Ao avaliar as cinco pesquisas, percebe-se que duas abordam a adaptação para VMs (Ajila & Bankole, 2016; Nikravesh *et al.*, 2017) e três para MBAs (Alipour & Liu, 2017; Wong, 2018; ML-Adapt). Nas três pesquisas com MBAs, somente duas propõem uma solução adaptativa (Alipour & Liu, 2017; ML-Adapt). Porém, apenas o ML-Adapt é um ambiente adaptativo implementado sobre um *feedback loop* MAPE-K.

Neste contexto, o ML-Adapt se destaca por ser um novo ambiente adaptativo proativo e genérico que lida com o gerenciamento do desempenho das MBAs. É o único ambiente proativo construído sobre um *feedback loop* comumente utilizado em soluções adaptativas e que

Tabela 9: Sumário dos trabalhos relacionados.

Autor	Onde	Quando	O que	Porque	Como	AM	MAPE-K
Rajagopalan & Jamjoom (2015)	Atualização	Variação no Desempenho	Microsserviço	Falhas	Reversão da versão	Não	Não
Ajila & Bankole (2016)	-	-	-	-	-	Sim (SVR e MLP)	Não
Florio & Nitto (2016)	Recursos	Limites Atingidos	Microsserviço	Desempenho	*Autoscaling*	Não	Sim
Kookarinrat & Temtanapat (2016)	Chamadas remotas	Troca de Mensagens	Protocolo	Comunicação	Barramento externo	Não	Não
Nikravesh *et al.* (2017)	-	-	-	-	-	Sim (SVR e MLP)	Não
Alipour & Liu (2017)	Recursos	Limites Atingidos	Microsserviço	Desempenho	*Autoscaling*	Sim (MLR e LR)	Não
Podolskiy *et al.* (2018)	-	-	-	-	-	Sim (ES, SSA, SARIMA, ARFIMA, SVR e LR)	-
Wong (2018)	-	-	-	Desempenho	*Autoscaling*	Sim (LSTM)	Não
Klinaku *et al.* (2018)	Recursos	Limites Atingidos	Microsserviço	Desempenho	*Autoscaling*	Não	Sim
Zhang *et al.* (2018)	Recursos	Limites Atingidos	Microsserviço	Desempenho	*Autoscaling*	Não	Sim
Sampaio *et al.* (2019)	Localização	Otimização	MBA	Desempenho	*Models Runtime*	Não	Sim
ML-Adapt	Recursos	Limites Atingidos	Microsserviço	Desempenho	*Autoscaling*	Sim (SVR, MLP, RF e XGBoost)	Sim

avaliou efetivamente os benefícios da proatividade em MBAs. Finalmente, é ainda uma proposta que aplica algoritmos de AM comuns, MLP e SVR, para previsão de recursos de aplicações implantadas na nuvem.

5.5 CONSIDERAÇÕES FINAIS

Este capítulo apresentou os trabalhos relacionados a soluções que lidam com a adaptação de MBA e/ou previsão de recursos utilizando algoritmos de AM. Os trabalhos relacionados e o trabalho proposto foram comparados por meio de um conjunto de questões que é capaz de capturar os requisitos do software adaptativo através de perguntas como onde adaptar, por que adaptar, quando adaptar, como adaptar e o que adaptar. Além disso, foi avaliado se a pesquisa utilizava algoritmo de AM ou é desenvolvida sobre o *feedback loop* MAPE-K.

6 CONCLUSÃO E TRABALHOS FUTUROS

Este capítulo apresenta as contribuições do trabalho proposto, as suas limitações e possibilidades de próximos passos.

6.1 CONCLUSÕES

A construção de MBAs está começando a ser uma nova tendência no desenvolvimento de aplicações em nuvem. Por esse motivo, soluções adaptativas foram propostas na literatura para lidar com o gerenciamento de recursos dos microsserviços em tempo de execução. No entanto, dentro dessas soluções, um pequeno número avalia os benefícios da proatividade no processo adaptativo. Como proposta para a adaptação proativa de MBAs, foi apresentada um ambiente independente de plataforma para gerenciar microsserviços com base na demanda futura do consumo de recursos em tempo de execução.

Para isso, foi desenvolvido o ML-Adapt (***M**achine **L**earning **Adapt**ation of microservice-based applications*), um ambiente que prevê o consumo de CPU futuro dos microsserviços e, baseado nessa informação, planeja um conjunto de ações corretivas, *e.g.,* aumento e diminuição de réplicas. No ML-Adapt, a predição de recursos é efetuada por meio de modelos de AM (MLP, RF, SVR e XGBoost) treinados em tempo de projeto. Ele é um ambiente genérico que permite a adição de qualquer componente em sua arquitetura, *e.g.,* FCC, MBAs, coletores de recursos, modelos de AM. Além disso, ele é implementado sobre um *feedback loop* MAPE-K.

O uso do ML-Adapt como ambiente adaptativo tem potencial para:

- Melhorar o desempenho do sistema gerenciado. Conhecer a demanda futura permite modificar com antecedência o número de microsserviços em execução para suprir a demanda solicitada, melhorando a QoS geral provida pelas MBAs;
- Reduzir o consumo de recursos. Outra vantagem possível através da previsão é a desalocação de recursos desnecessários liberando recursos para outros microsserviços;
- Ambiente genérico. O ML-Adapt não especifica a utilização de nenhum software. Logo, este pode ser adaptado para gerenciar FCC como Kubernetes ou Docker Swarm, coletar informações usando banco de dados como Prometheus ou InfluxDB, prever o consumo futuro de recursos usando quaisquer algoritmos de AM como Long Short-Term Memory (LSTM) ou Linear Regression (LR).

6.2 CONTRIBUIÇÕES

A principal contribuição desta dissertação é um ambiente holístico genérico para adaptação proativa de MBAs usando modelos de AM. Além da contribuição principal, ML-Adapt produziu algumas contribuições secundárias:

1. Um conjunto abrangente de etapas e tecnologias para o uso de algoritmos de AM no contexto de MBAs;

2. Monitoramento não-invasivo através de um cliente desenvolvido para o banco de dados Prometheus apto para coletar, minerar e processar métricas de desempenho acessíveis em tempo de execução;

3. Um conector produzido para estabelecer comunicação com a API do Kubernetes capaz de modificar a quantidade de réplicas dos microsserviços através do dimensionamento automático.

6.3 LIMITAÇÕES

O ML-Adapt tem algumas limitações:

- A quantidade de modelos necessários para compor o ambiente é a principal limitação de AM. O Hipster Shop, por exemplo, demandou quarenta modelos treinados. Apesar dessa quantidade, a instrumentação e a construção dos modelos no ML-Adapt não foram atividades com alto custo para este trabalho. No entanto, MBAs, geralmente não se limitam a essa pequena quantidade de microsserviços. O Train Ticket[1], por exemplo, é uma MBA implementada com 41 microsserviços. Se ele fosse usado como *testbed* nesta dissertação, exigiria o total de 164 modelos de AM.

- Avaliação de apenas quatro algoritmos de AM, desconsiderando algoritmos preditivos como Redes Neurais Recorrentes, LR ou LSTM;

- A não aplicação da técnica de Aprendizagem Online, que é importante para lidar com a mudança de padrões ao longo do tempo e está presente em alguns trabalhos de microsserviços (Alipour & Liu, 2017; Wong, 2018).

- O foco apenas na métrica de desempenho CPU, não considerando a predição de outras métricas como consumo de memória, rede, vazão, tempo de resposta;

- A não reversão de adaptações que degradam o sistema. O ML-Adapt ainda não monitora o impacto de uma adaptação para a MBA após a sua ocorrência. Para lidar com degradação de desempenho gerada por adaptações incorretas, o ML-Adapt precisa realizar outra adaptação no próximo monitoramento.

[1] https://github.com/FudanSELab/train-ticket/

6.4 TRABALHOS FUTUROS

Após a realização desta dissertação, novos trabalhos podem dar continuidade para tornar o ML-Adapt um ambiente robusto para adaptação de MBAs. A seguir, serão listados os principais trabalhos futuros:

- Para lidar com a quantidade excessiva de modelos treinados é possível aplicar a técnica de seleção dinâmica de preditores. Essa técnica consiste em treinar um conjunto de modelos preditivos (*ensemble*) usando diferentes algoritmos preditores, *e.g.,* MLP, SVR, ARIMA e o LSTM. Em seguida, se o consumo de um determinado microsserviço precisa ser predito, escolhe-se apenas "o melhor modelo" dentro do conjunto. Essa escolha é baseada na relação da série temporal do microsserviço a ser predito, informação coletada em tempo de execução, com o padrão da série usada no momento do treinamento do modelo. Essa escolha pode ser efetuada por métodos de discrepância como o Akaike's Information Criteria (AIC) (Akaike, 1998) ou Bayesian Information Criteria (BIC) (Schwarz *et al.*, 1978). Dessa forma, dois microsserviços, que possuem séries com comportamentos parecidos, podem ser preditos por um único modelo;

- Os modelos treinados pelos algoritmos RF e XGBoost não obtiveram um bom desempenho em previsão. Porém, apresentam bons resultados em outros problemas, como o de classificação de séries temporais (Chen & Guestrin, 2016; Grohmann *et al.*, 2019). Logo, usar algoritmos de classificação para determinar se a demanda futura de um microsserviço está violando ou não um limite é uma possível alternativa aos modelos de predição de recursos. Dessa forma, o ML-Adapt classificaria como violação ou não, ao invés de tentar predizer o consumo futuro;

- Para melhorar a fase de análise do ML-Adapt, é interessante investigar outras métricas de desempenho, *e.g.,* tempo de resposta, vazão, consumo de memória. Essas métricas podem ser utilizadas para previsão de demanda futura, como já ocorre no ML-Adapt, ou apenas serem consideradas no momento da tomada de decisão. Dessa forma, o segundo caso, avalia o valor da métrica atual (age reativamente) para obter informações extras sobre as condições do sistema.

REFERÊNCIAS

Abdullah, M., Iqbal, W., & Erradi, A. (2019). Unsupervised learning approach for web application auto-decomposition into microservices. *Journal of Systems and Software*, 151:243 – 257.

Ahmed, N. K., Atiya, A. F., Gayar, N. E., & El-shishiny, H. (2010). An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews*, 29(5-6):594–621.

Ajila, S. & Bankole, A. (2016). Using machine learning algorithms for cloud client prediction models in a web vm resource provisioning environment. *Transactions on Machine Learning and Artificial Intelligence*, 4.

Akaike, H. (1998). *Information Theory and an Extension of the Maximum Likelihood Principle*, 199–213. Springer New York, New York, NY.

Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., & Merle, P. (2017). Autonomic vertical elasticity of docker containers with elasticdocker. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, 472–479.

Al Shalabi, L., Shaaban, Z., & Kasasbeh, B. (2006). Data mining: A preprocessing engine. *Journal of Computer Science*, 2(9):735–739.

Alipour, H. & Liu, Y. (2017). Online machine learning for cloud resource provisioning of microservice backend systems. In *2017 IEEE International Conference on Big Data (Big Data)*, 2433–2441.

Baldan, F. J., Ramirez-Gallego, S., Bergmeir, C., Herrera, F., & Benitez, J. M. (2018). A forecasting methodology for workload forecasting in cloud systems. *IEEE Transactions on Cloud Computing*, 6(4):929–941.

Bergman, S. & Schiffer, M. (1951). Kernel functions and conformal mapping. *Compositio Mathematica*, 8:205–249.

Bernstein, D. (2014). Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1(3):81–84.

Borkowski, M., Schulte, S., & Hochreiner, C. (2016). Predicting cloud resource utilization. In *Proceedings of the 9th International Conference on Utility and Cloud Computing*, 37–42.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Brockwell, P. J., Davis, R. A., & Fienberg, S. E. (1991). *Time Series: Theory and Methods: Theory and Methods*. Springer Science & Business Media.

Chang, C.-C. & Lin, C.-J. (2011). Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3).

Chen, T. & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794.

Chen, Z., Jiang, F., Cheng, Y., Gu, X., Liu, W., & Peng, J. (2018). Xgboost classifier for ddos attack detection and analysis in sdn-based cloud. In *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 251–256.

Computing, A. *et al.* (2006). An architectural blueprint for autonomic computing. *IBM White Paper*, 31(2006):1–6.

Cutler, A., Cutler, D. R., & Stevens, J. R. (2012). *Random Forests*, 157–175. Springer US, Boston, MA.

Da, K., Dalmau, M., & Roose, P. (2011). A Survey of adaptation systems. *International Journal on Internet and Distributed Computing Systems*, 2(1):1–18.

de Mattos Neto, P. S., Lima Junior, A. R., & Ferreira, T. A. (2010). Time series forecasting using a perturbative intelligent system. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, 1477–1478.

Desai, A., Oza, R., Sharma, P., & Patel, B. (2013). Hypervisor: A survey on concepts and taxonomy. *International Journal of Innovative Technology and Exploring Engineering*, 2(3):222–225.

Drucker, H., Burges, C. J. C., Kaufman, L., Smola, A. J., & Vapnik, V. (1997). Support vector regression machines. In Mozer, M. C., Jordan, M. I., & Petsche, T., editors, *Advances in Neural Information Processing Systems 9*, 155–161. MIT Press, Denver, CO, USA.

Ehlers, R. S. (2007). Análise de séries temporais. *Universidade Federal do Paraná*.

El Naqa, I. & Murphy, M. J. (2015). What is machine learning? In *Machine Learning in Radiation Oncology*, 3–11. Springer.

Ferreira, M. d. F. M. (1999). *Árvores de regressão e generalizações: Aplicações*. PhD thesis, Faculdade de Ciências da Universidade do Porto.

Florio, L. & Nitto, E. D. (2016). Gru: An approach to introduce decentralized autonomic behavior in microservices architectures. In *2016 IEEE International Conference on Autonomic Computing (ICAC)*, 357–362.

Friedman, J., Hastie, T., Tibshirani, R., *et al.* (2000). Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407.

Gabbrielli, M., Giallorenzo, S., Guidi, C., Mauro, J., & Montesi, F. (2016). Self-reconfiguring microservices. In *Essays Dedicated to Frank De Boer on Theory and Practice of Formal Methods - Volume 9660*, 194–210.

Gardner, M. & Dorling, S. (1998). Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric Environment*, 32(14):2627 – 2636.

Graves, A., Fernández, S., Gomez, F., & Schmidhuber, J. (2006). Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, 369–376.

Grohmann, J., Nicholson, P. K., Iglesias, J. O., Kounev, S., & Lugones, D. (2019). Monitorless: Predicting performance degradation in cloud applications with machine learning. In *Proceedings of the 20th International Middleware Conference*, 149–162.

Gumus, M. & Kiran, M. S. (2017). Crude oil price forecasting using xgboost. In *2017 International Conference on Computer Science and Engineering (UBMK)*, 1100–1103.

Hassan, S., Ali, N., & Bahsoon, R. (2017). Microservice ambients: An architectural meta-modelling approach for microservice granularity. In *2017 IEEE International Conference on Software Architecture (ICSA)*, 1–10.

Hassan, S. & Bahsoon, R. (2016). Microservices and their design trade-offs: A self-adaptive roadmap. In *2016 IEEE International Conference on Services Computing (SCC)*, 813–818.

Hawkins, D. M. (2004). The problem of overfitting. *Journal of Chemical Information and Computer Sciences*, 44(1):1–12. PMID: 14741005.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation*. International edition. Prentice Hall.

Herbst, N. R., Kounev, S., & Reussner, R. (2013). Elasticity in cloud computing: What it is, and what it is not. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*, 23–27.

Hightower, K., Beda, J., & Burns, B. (2017). *Kubernetes: Up and Running: Dive Into the Future of Infrastructure*. O'Reilly Media.

Huebscher, M. C. & McCann, J. A. (2008). A survey of autonomic computing—degrees, models, and applications. *ACM Comput. Surv.*, 40(3).

Hussain, M., Wajid, S. K., Elzaart, A., & Berbar, M. (2011). A comparison of svm kernel functions for breast cancer detection. In *2011 Eighth International Conference Computer Graphics, Imaging and Visualization*, 145–150.

Jain, A. K., Jianchang Mao, & Mohiuddin, K. M. (1996). Artificial neural networks: a tutorial. *Computer*, 29(3):31–44.

Jain, R. (1990). *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley.

Kephart, J. O. & Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 36(1):41–50.

Kisilevich, S., Mansmann, F., Nanni, M., & Rinzivillo, S. (2010). *Spatio-temporal clustering*, 855–874. Springer US, Boston, MA.

Klinaku, F., Frank, M., & Becker, S. (2018). Caus: An elasticity controller for a containerized microservice. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, 93–98.

Kookarinrat, P. & Temtanapat, Y. (2016). Design and implementation of a decentralized message bus for microservices. In *2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 1–6.

Kumar, J. & Singh, A. K. (2018). Workload prediction in cloud using artificial neural network and adaptive differential evolution. *Future Generation Computer Systems*, 81:41–52.

Langley, P. & Simon, H. A. (1995). Applications of machine learning and rule induction. *Commun. ACM*, 38(11):54–64.

Laureno, M. (2006). *Máquinas Virtuais e Emuladores: Conceitos, Técnicas e Aplicaçoes*. Novatec.

Lewis, J. & Fowler, M. (2014). Microservices: A definition of this new architectural term.

Li, X., Chen, Y., & Lin, Z. (2019). Towards automated inter-service authorization for microservice applications. In *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*, 3–5.

Liaw, A., Wiener, M., *et al.* (2002). Classification and regression by randomforest. *R news*, 2(3):18–22.

Lorido-Botran, T., Miguel-Alonso, J., & Lozano, J. A. (2014). A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing*, 12(4):559–592.

Mahalakshmi, G., Sridevi, S., & Rajaram, S. (2016). A survey on forecasting of time series data. In *2016 International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE'16)*, 1–8.

Moreno-Vozmediano, R., Montero, R. S., Huedo, E., & Llorente, I. M. (2019). Efficient resource provisioning for elastic cloud services based on machine learning techniques. *Journal of Cloud Computing*, 8(1):5.

Newman, S. (2015). *Building Microservices*. O'Reilly Media, Inc., 1st edition.

Nikravesh, A. Y., Ajila, S. A., & Lung, C.-H. (2017). An autonomic prediction suite for cloud resource provisioning. *Journal of Cloud Computing*, 6(1):3.

Nowicka-Zagrajek, J. & Weron, R. (2002). Modeling electricity loads in california: Arma models with hyperbolic noise. *Signal Processing*, 82(12):1903 – 1915.

Pahl, C. (2015). Containerization and the paas cloud. *IEEE Cloud Computing*, 2(3):24–31.

Pahl, C., Brogi, A., Soldani, J., & Jamshidi, P. (2019). Cloud container technologies: A state-of-the-art review. *IEEE Transactions on Cloud Computing*, 7(3):677–692.

Palmer, D. S., O'Boyle, N. M., Glen, R. C., & Mitchell, J. B. O. (2007). Random forest models to predict aqueous solubility. *Journal of Chemical Information and Modeling*, 47(1):150–158.

Podolskiy, V., Jindal, A., Gerndt, M., & Oleynik, Y. (2018). Forecasting models for self-adaptive cloud applications: A comparative study. In *2018 IEEE 12th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, 40–49.

Rajagopalan, S. & Jamjoom, H. (2015). App-bisect: Autonomous healing for microservice-based apps. In *Proceedings of the 7th USENIX Conference on Hot Topics in Cloud Computing*, 16–16.

Ren, X., Guo, H., Li, S., Wang, S., & Li, J. (2017). A novel image classification method with cnn-xgboost model. In *Digital Forensics and Watermarking*, 378–390.

Ribeiro, G. H. T., d. M. Neto, P. S. G., Cavalcanti, G. D. C., & Tsang, I. R. (2011). Lag selection for time series forecasting using particle swarm optimization. In *The 2011 International Joint Conference on Neural Networks*, 2437–2444.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Salehie, M. & Tahvildari, L. (2009). Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2):14:1–14:42.

Sampaio, A. R., Rubin, J., Beschastnikh, I., & Rosa, N. S. (2019). Improving microservice-based applications with runtime placement adaptation. *Journal of Internet Services and Applications*, 10(1):4.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229.

Sapankevych, N. I. & Sankar, R. (2009). Time series prediction using support vector machines: A survey. *IEEE Computational Intelligence Magazine*, 4(2):24–38.

Scargle, J. D. (1989). Studies in astronomical time series analysis. iii. fourier transforms, auto-correlation functions, and cross-correlation functions of unevenly spaced data. *The Astrophysical Journal*, 343:874.

Schwarz, G. *et al.* (1978). Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464.

Silva, E. G., de O. Júunior, D. S., Cavalcanti, G. D. C., & de Mattos Neto, P. S. G. (2018). Improving the accuracy of intelligent forecasting models using the perturbation theory. In *2018 International Joint Conference on Neural Networks (IJCNN)*, 1–7.

Smola, A. J. & Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222.

Sompolinsky, H. & Kanter, I. (1986). Temporal association in asymmetric neural networks. *Phys. Rev. Lett.*, 57:2861–2864.

Soppelsa, F. & Kaewkasi, C. (2017). *Native Docker Clustering with Swarm*. Packt Publishing.

Taibi, D., Lenarduzzi, V., & Pahl, C. (2017). Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. *IEEE Cloud Computing*, 4(5):22–32.

Taibi, D., Lenarduzzi, V., Pahl, C., & Janes, A. (2017). Microservices in agile software development: A workshop-based study into issues, advantages, and disadvantages. In *Proceedings of the XP2017 Scientific Workshops*, 23:1–23:5.

Toffetti, G., Brunner, S., Blöchlinger, M., Dudouet, F., & Edmonds, A. (2015). An architecture for self-managing microservices. In *Proceedings of the 1st International Workshop on Automated Incident Management in Cloud*, 19–24.

Toffetti, G., Brunner, S., Blöchlinger, M., Spillner, J., & Bohnert, T. M. (2017). Self-managing cloud-native applications: Design, implementation, and experience. *Future Generation Computer Systems*, 72:165 – 179.

Vapnik, V. (1963). Pattern recognition using generalized portrait method. *Automation and remote control*, 24:774–780.

Wang, L., Zhou, X., Zhu, X., Dong, Z., & Guo, W. (2016). Estimation of biomass in wheat using random forest regression algorithm and remote sensing data. *The Crop Journal*, 4(3):212 – 219.

Werbos, P. J. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University.

Weyns, D., Schmerl, B., Grassi, V., Malek, S., Mirandola, R., Prehofer, C., Wuttke, J., Andersson, J., Giese, H., & Göschka, K. M. (2013). *On Patterns for Decentralized Control in Self-Adaptive Systems*, 76–107. Springer Berlin Heidelberg, Berlin, Heidelberg.

Willmott, C. J., Ackleson, S. G., Davis, R. E., Feddema, J. J., Klink, K. M., Legates, D. R., O'Donnell, J., & Rowe, C. M. (1985). Statistics for the evaluation and comparison of models. *Journal of Geophysical Research: Oceans*, 90(C5):8995–9005.

Wong, J. P. (2018). *HyScale: Hybrid Scaling of Dockerized Microservices Architectures*. PhD thesis, University of Toronto.

Xu, L., Lin, W., & Kuo, C. (2015). *Visual Quality Assessment by Machine Learning*. SpringerBriefs in Electrical and Computer Engineering. Springer Singapore.

Yu, G., Chen, P., & Zheng, Z. (2019). Microscaler: Automatic scaling for microservices with an online learning approach. In *2019 IEEE International Conference on Web Services (ICWS)*, 68–75.

Zhang, L. & Zhan, C. (2017). *Machine Learning in Rock Facies Classification: An Application of XGBoost*, 1371–1374. SEG Global Meeting Abstracts. Society of Exploration Geophysicists and Chinese Petroleum Society.

Zhang, S., Mao, X., Liu, P., & Hou, F. (2018). A self-adaptation framework of microservice systems (s). In *SEKE*, 282–325.

APÊNDICE A – ROUND TRIP TIME

Tabela 10: RTT na carga variável Aleatória 1.

	Módulos Preventivos			
Configuração	Nenhum	Quarentena	Contenção de Adaptações Sucessivas	Ambos
SVR	192.686	229.623	200.189	**185.966**
MLP	402.223	366.771	216.360	**165.629**
HPA	**192.086**	-	-	-
RF	926.069	1014.709	**774.480**	939.583
XGBoost	1051.651	872.686	**837.291**	938.114
Base	**1083.463**	-	-	-

Tabela 11: RTT na carga variável Aleatória 2.

	Módulos Preventivos			
Configuração	Nenhum	Quarentena	Contenção de Adaptações Sucessivas	Ambos
SVR	151.154	228.080	**138.874**	169.000
MLP	301.131	295.149	188.611	**166.040**
HPA	**144.171**	-	-	-
RF	**748.371**	1059.154	942.589	946.406
XGBoost	871.423	**859.669**	868.691	1160.680
Base	**817.417**	-	-	-

Tabela 12: RTT na carga fixa Alta.

	Módulos Preventivos			
Configuração	Nenhum	Quarentena	Contenção de Adaptações Sucessivas	Ambos
SVR	258.577	235.040	268.069	**220.789**
MLP	766.794	774.874	**431.434**	483.291
HPA	**546.297**	-	-	-
RF	1625.583	1879.857	1711.811	**1600.903**
XGBoost	1719.011	**1634.794**	1758.263	1993.417
Base	**1723.109**	-	-	-

Tabela 13: RTT na carga fixa Média.

	Módulos Preventivos			
Configuração	Nenhum	Quarentena	Contenção de Adaptações Sucessivas	Ambos
SVR	**146.914**	198.703	201.451	199.840
MLP	493.640	337.291	379.206	**309.417**
HPA	**328.634**	-	-	-
RF	869.057	909.789	**728.623**	952.509
XGBoost	1282.320	1230.063	992.349	**938.257**
Base	**1164.731**	-	-	-

Tabela 14: RTT na carga fixa Baixa.

	Módulos Preventivos			
Configuração	Nenhum	Quarentena	Contenção de Adaptações Sucessivas	Ambos
SVR	**114.874**	131.937	135.354	145.783
MLP	268.251	**192.343**	221.554	211.006
HPA	**205.114**	-	-	-
RF	**390.686**	445.811	458.880	417.349
XGBoost	**426.594**	466.051	520.526	539.177
Base	**580.629**	-	-	-

Tabela 15: Melhores RTT por carga de trabalho.

	Configurações					
Carga de Trabalho	SVR	MLP	HPA	RF	XGBoost	Base
Aleatória 1	185.966	**165.629**	192.086	774.480	837.291	1083.463
Aleatória 2	**138.874**	166.040	144.171	748.371	859.669	817.417
Alta	**220.789**	431.434	546.297	1600.903	1634.794	1723.109
Média	**146.914**	309.417	328.634	728.623	938.257	1164.731
Baixa	**114.874**	192.343	205.114	390.686	426.594	580.629

APÊNDICE B – MÉTRICA ROUND TRIP TIME ORDENADA

Tabela 16: Experimentos da carga variável Aleatória 1 ordenados pelo RTT.

Posição	Configuração	RTT (ms)	Módulo Preventivo
1ª	MLP	165.629	Ambos
2ª	SVR	185.966	Ambos
3ª	HPA	192.086	-
4ª	SVR	192.686	Nenhum
5ª	SVR	200.189	Contenção de Adaptações Sucessivas
6ª	MLP	216.360	Contenção de Adaptações Sucessivas
7ª	SVR	229.623	Quarentena
8ª	MLP	366.771	Quarentena
9ª	MLP	402.223	Nenhum
10ª	RF	774.480	Contenção de Adaptações Sucessivas
11ª	XGBoost	837.291	Contenção de Adaptações Sucessivas
12ª	XGBoost	872.686	Quarentena
13ª	RF	926.069	Nenhum
14ª	XGBoost	938.114	Ambos
15ª	RF	939.583	Ambos
16ª	RF	1014.709	Quarentena
17ª	XGBoost	1051.651	Nenhum
18ª	Base	1083.463	-

Tabela 17: Experimentos da carga variável Aleatória 2 ordenados pelo RTT.

Posição	Configuração	RTT (ms)	Módulo Preventivo
1ª	SVR	138.874	Contenção de Adaptações Sucessivas
2ª	HPA	144.171	-
3ª	SVR	151.154	Nenhum
4ª	MLP	166.040	Ambos
5ª	SVR	169.000	Ambos
6ª	MLP	188.611	Contenção de Adaptações Sucessivas
7ª	SVR	228.080	Quarentena
8ª	MLP	295.149	Quarentena
9ª	MLP	301.131	Nenhum
10ª	RF	748.371	Nenhum
11ª	Base	817.417	-
12ª	XGBoost	859.669	Quarentena
13ª	XGBoost	868.691	Contenção de Adaptações Sucessivas
14ª	XGBoost	871.423	Nenhum
15ª	RF	942.589	Contenção de Adaptações Sucessivas
16ª	RF	946.406	Ambos
17ª	RF	1059.154	Quarentena
18ª	XGBoost	1160.680	Ambos

Tabela 18: Experimentos da carga fixa Alta ordenados pelo RTT.

Posição	Configuração	RTT (ms)	Módulo Preventivo
1ª	SVR	220.789	Ambos
2ª	SVR	235.040	Quarentena
3ª	SVR	258.577	Nenhum
4ª	SVR	268.069	Contenção de Adaptações Sucessivas
5ª	MLP	431.434	Contenção de Adaptações Sucessivas
6ª	MLP	483.291	Ambos
7ª	HPA	546.297	-
8ª	MLP	766.794	Nenhum
9ª	MLP	774.874	Quarentena
10ª	RF	1600.903	Ambos
11ª	RF	1625.583	Nenhum
12ª	XGBoost	1634.794	Quarentena
13ª	RF	1711.811	Contenção de Adaptações Sucessivas
14ª	XGBoost	1719.011	Nenhum
15ª	Base	1723.109	-
16ª	XGBoost	1758.263	Contenção de Adaptações Sucessivas
17ª	RF	1879.857	Quarentena
18ª	XGBoost	1993.417	Ambos

Tabela 19: Experimentos da carga fixa Média ordenados pelo RTT.

Posição	Configuração	RTT (ms)	Módulo Preventivo
1ª	SVR	114.874	Nenhum
2ª	SVR	131.937	Quarentena
3ª	SVR	135.354	Contenção de Adaptações Sucessivas
4ª	SVR	145.783	Ambos
5ª	MLP	192.343	Quarentena
6ª	HPA	205.114	-
7ª	MLP	211.006	Ambos
8ª	MLP	221.554	Contenção de Adaptações Sucessivas
9ª	MLP	268.251	Nenhum
10ª	RF	390.686	Nenhum
11ª	RF	417.349	Ambos
12ª	XGBoost	426.594	Nenhum
13ª	RF	445.811	Quarentena
14ª	RF	458.880	Contenção de Adaptações Sucessivas
15ª	XGBoost	466.051	Quarentena
16ª	XGBoost	520.526	Contenção de Adaptações Sucessivas
17ª	XGBoost	539.177	Ambos
18ª	Base	580.629	-

Tabela 20: Experimentos da carga fixa Baixa ordenados pelo RTT.

Posição	Configuração	RTT (ms)	Módulo Preventivo
1ª	SVR	114.874	Nenhum
2ª	SVR	131.937	Quarentena
3ª	SVR	135.354	Contenção de Adaptações Sucessivas
4ª	SVR	145.783	Ambos
5ª	MLP	192.343	Quarentena
6ª	HPA	205.114	-
7ª	MLP	211.006	Ambos
8ª	MLP	221.554	Contenção de Adaptações Sucessivas
9ª	MLP	268.251	Nenhum
10ª	RF	390.686	Nenhum
11ª	RF	417.349	Ambos
12ª	XGBoost	426.594	Nenhum
13ª	RF	445.811	Quarentena
14ª	RF	458.880	Contenção de Adaptações Sucessivas
15ª	XGBoost	466.051	Quarentena
16ª	XGBoost	520.526	Contenção de Adaptações Sucessivas
17ª	XGBoost	539.177	Ambos
18ª	Base	580.629	-

APÊNDICE C – CPU

Tabela 21: Consumo de CPU na carga variável Aleatória 1.

	Módulos Preventivos			
Configuração	Nenhum	Quarentena	Contenção de Adaptações Sucessivas	Ambos
SVR	101.02%	114.37%	117.63%	**119.43%**
MLP	82.18%	93.06%	126.67%	**134.33%**
HPA	**132.95%**	-	-	-
RF	76.41%	76.54%	**79.09%**	79.35%
XGBoost	76.05%	78.95%	**80.01%**	78.70%
Base	**77.84%**	-	-	-

Tabela 22: Consumo de CPU na carga variável Aleatória 2

	Módulos Preventivos			
Configuração	Nenhum	Quarentena	Contenção de Adaptações Sucessivas	Ambos
SVR	104.10%	112.47%	**116.51%**	118.38%
MLP	83.78%	97.82%	121.93%	**131.32%**
HPA	**125.47%**	-	-	-
RF	**74.77%**	78.73%	77.89%	78.47%
XGBoost	76.20%	**76.32%**	75.76%	74.62%
Base	**77.89%**	-	-	-

Tabela 23: Consumo de CPU na carga fixa Alta

	Módulos Preventivos			
Configuração	Nenhum	Quarentena	Contenção de Adaptações Sucessivas	Ambos
SVR	88.99%	90.82%	89.94%	**91.36%**
MLP	83.29%	84.30%	**88.66%**	85.19%
HPA	**87.10%**	-	-	-
RF	82.24%	84.89%	85.38%	**86.62%**
XGBoost	87.74%	**85.80%**	84.95%	83.65%
Base	**84.21%**	-	-	-

Tabela 24: Consumo de CPU na carga fixa Média.

	Módulos Preventivos			
Configuração	Nenhum	Quarentena	Contenção de Adaptações Sucessivas	Ambos
SVR	**87.70%**	85.98%	92.28%	90.47%
MLP	80.61%	90.08%	81.27%	**86.81%**
HPA	**86.59%**	-	-	-
RF	83.17%	84.35%	**85.95%**	81.86%
XGBoost	76.79%	79.85%	83.84%	**81.83%**
Base	**80.68%**	-	-	-

Tabela 25: Consumo de CPU na carga fixa Baixa.

	Módulos Preventivos			
Configuração	Nenhum	Quarentena	Contenção de Adaptações Sucessivas	Ambos
SVR	**73.85%**	77.90%	77.45%	76.18%
MLP	72.49%	**77.70%**	77.75%	77.07%
HPA	**77.82%**	-	-	-
RF	**75.93%**	76.69%	76.65%	77.62%
XGBoost	**76.61%**	76.68%	74.56%	73.29%
Nenhum	**74.45%**	-	-	-

Tabela 26: Melhores resultados do consumo de CPU por carga de trabalho.

	Configuração					
Carga de Trabalho	SVR	MLP	HPA	RF	XGBoost	Base
Aleatória 1	119.43%	**119.43%**	119.43%	119.43%	119.43%	119.43%
Aleatória 2	**131.32%**	131.32%	131.32%	131.32%	131.32%	131.32%
Alta	**87.10%**	87.10%	87.10%	87.10%	87.10%	87.10%
Média	**85.95%**	85.95%	85.95%	85.95%	85.95%	85.95%
Baixa	**76.61%**	76.61%	76.61%	76.61%	76.61%	76.61%

APÊNDICE D – MÉTRICA CPU ORDENADA

Tabela 27: Experimentos da carga variável Aleatória 1 ordenados pela CPU.

Posição	Configuração	Processamento (%)	Módulo Preventivo
1ª	XGBoost	76.05%	Nenhum
2ª	RF	76.41%	Nenhum
3ª	RF	76.54%	Quarentena
4ª	Base	77.84%	-
5ª	XGBoost	78.70%	Ambas
6ª	XGBoost	78.95%	Quarentena
7ª	RF	79.09%	Contenção de Adaptações Sucessivas
8ª	RF	79.35%	Ambas
9ª	XGBoost	80.01%	Contenção de Adaptações Sucessivas
10ª	MLP	82.18%	Nenhum
11ª	MLP	93.06%	Quarentena
12ª	SVR	101.02%	Nenhum
13ª	SVR	114.37%	Quarentena
14ª	SVR	117.63%	Contenção de Adaptações Sucessivas
15ª	SVR	119.43%	Ambas
16ª	MLP	126.67%	Contenção de Adaptações Sucessivas
17ª	HPA	132.95%	-
18ª	MLP	134.33%	Ambas

Tabela 28: Experimentos da carga variável Aleatória 2 ordenados pela CPU.

Posição	Configuração	Processamento (%)	Módulo Preventivo
1ª	XGBoost	74.62%	Ambas
2ª	RF	74.77%	Nenhum
3ª	XGBoost	75.76%	Contenção de Adaptações Sucessivas
4ª	XGBoost	76.20%	Nenhum
5ª	XGBoost	76.32%	Quarentena
6ª	RF	77.89%	Contenção de Adaptações Sucessivas
7ª	Base	77.89%	-
8ª	RF	78.47%	Ambas
9ª	RF	78.73%	Quarentena
10ª	MLP	83.78%	Nenhum
11ª	MLP	97.82%	Quarentena
12ª	SVR	104.10%	Nenhum
13ª	SVR	112.47%	Quarentena
14ª	SVR	116.51%	Contenção de Adaptações Sucessivas
15ª	SVR	118.38%	Ambas
16ª	MLP	121.93%	Contenção de Adaptações Sucessivas
17ª	HPA	125.47%	-
18ª	MLP	131.32%	Ambas

Tabela 29: Experimentos da carga fixa Alta ordenados pela CPU.

Posição	Configuração	Processamento (%)	Módulo Preventivo
1ª	RF	82.24%	Nenhum
2ª	MLP	83.29%	Nenhum
3ª	XGBoost	83.65%	Ambas
4ª	Base	84.21%	-
5ª	MLP	84.30%	Quarentena
6ª	RF	84.89%	Quarentena
7ª	XGBoost	84.95%	Contenção de Adaptações Sucessivas
8ª	MLP	85.19%	Ambas
9ª	RF	85.38%	Contenção de Adaptações Sucessivas
10ª	XGBoost	85.80%	Quarentena
11ª	RF	86.62%	Ambas
12ª	HPA	87.10%	-
13ª	XGBoost	87.64%	Nenhum
14ª	MLP	88.66%	Contenção de Adaptações Sucessivas
15ª	SVR	88.99%	Nenhum
16ª	SVR	89.94%	Contenção de Adaptações Sucessivas
17ª	SVR	90.82%	Quarentena
18ª	SVR	91.36%	Ambas

Tabela 30: Experimentos da carga fixa Média ordenados pela CPU.

Posição	Configuração	Processamento (%)	Módulo Preventivo
1ª	XGBoost	76.79%	Nenhum
2ª	XGBoost	79.85%	Quarentena
3ª	MLP	80.61%	Nenhum
4ª	Base	80.68%	-
5ª	MLP	81.27%	Contenção de Adaptações Sucessivas
6ª	XGBoost	81.83%	Ambas
7ª	RF	81.86%	Ambas
8ª	RF	83.17%	Nenhum
9ª	XGBoost	83.84%	Contenção de Adaptações Sucessivas
10ª	RF	84.35%	Quarentena
11ª	RF	85.95%	Contenção de Adaptações Sucessivas
12ª	SVR	85.98%	Quarentena
13ª	HPA	86.59%	-
14ª	MLP	86.81%	Ambas
15ª	SVR	87.70%	Nenhum
16ª	MLP	90.08%	Quarentena
17ª	SVR	90.47%	Ambas
18ª	SVR	92.28%	Contenção de Adaptações Sucessivas

Tabela 31: Experimentos da carga fixa Baixa ordenados pela CPU.

Posição	Configuração	Processamento (%)	Módulo Preventivo
1ª	MLP	72.49%	Nenhum
2ª	XGBoost	73.29%	Ambas
3ª	SVR	73.85%	Nenhum
4ª	Base	74.45%	-
5ª	XGBoost	74.56%	Contenção de Adaptações Sucessivas
6ª	RF	75.93%	Nenhum
7ª	SVR	76.18%	Ambas
8ª	XGBoost	76.61%	Nenhum
9ª	RF	76.65%	Contenção de Adaptações Sucessivas
10ª	XGBoost	76.68%	Quarentena
11ª	RF	76.69%	Quarentena
12ª	MLP	77.07%	Ambas
13ª	SVR	77.45%	Contenção de Adaptações Sucessivas
14ª	RF	77.62%	Ambas
15ª	MLP	77.70%	Quarentena
16ª	MLP	77.75%	Contenção de Adaptações Sucessivas
17ª	HPA	77.82%	-
18ª	SVR	77.90%	Quarentena

APÊNDICE E – MEMÓRIA

Tabela 32: Consumo de memória na carga variável Aleatória 1.

	Módulos Preventivos			
Configuração	Nenhum	Quarentena	Contenção de Adaptações Sucessivas	Ambos
SVR	877.050	720.570	812.798	**720.529**
MLP	621.720	545.829	722.079	**700.797**
HPA	**678.259**	-	-	-
RF	511.431	474.878	**567.822**	478.704
XGBoost	553.516	513.468	**579.383**	503.914
Base	**472.878**	-	-	-

Tabela 33: Consumo de memória na carga variável Aleatória 2.

	Módulos Preventivos			
Configuração	Nenhum	Quarentena	Contenção de Adaptações Sucessivas	Ambos
SVR	1009.308	651.681	**714.300**	599.252
MLP	715.571	623.351	660.352	**750.106**
HPA	**605.471**	-	-	-
RF	**479.204**	483.859	463.376	459.286
XGBoost	455.435	**467.690**	462.387	459.953
Base	**425.589**	-	-	-

Tabela 34: Consumo de memória na carga fixa Alta.

	Módulos Preventivos			
Configuração	Nenhum	Quarentena	Contenção de Adaptações Sucessivas	Ambos
SVR	740.346	607.984	645.104	**581.750**
MLP	552.606	535.517	**577.970**	553.217
HPA	**591.430**	-	-	-
RF	639.895	519.684	550.794	**529.972**
XGBoost	563.664	**538.607**	473.545	481.716
Base	**481.317**	-	-	-

Tabela 35: Consumo de memória na carga fixa Média.

	Módulos Preventivos			
Configuração	Nenhum	Quarentena	Contenção de Adaptações Sucessivas	Ambos
SVR	**715.403**	649.750	789.084	754.942
MLP	532.763	541.555	534.777	**617.266**
HPA	**522.151**	-	-	-
RF	547.706	503.177	**557.787**	596.056
XGBoost	471.594	444.981	555.028	**615.010**
Base	**424.006**	-	-	-

Tabela 36: Consumo de memória na carga fixa Baixa.

	Módulos Preventivos			
Configuração	Nenhum	Quarentena	Contenção de Adaptações Sucessivas	Ambos
SVR	**763.784**	548.390	628.908	491.313
MLP	452.796	**486.537**	474.684	484.037
HPA	**598.077**	-	-	-
RF	**478.224**	462.532	535.455	516.959
XGBoost	**541.751**	469.150	434.088	436.002
Base	**499.105**	-	-	-

Tabela 37: Melhores resultados do consumo de memória por carga de trabalho.

	Configuração					
Carga de Trabalho	SVR	MLP	HPA	RF	XGBoost	Base
Aleatória 1	720.529	**700.797**	678.259	567.822	579.383	472.878
Aleatória 2	**714.300**	750.106	605.471	479.204	467.690	425.589
Alta	**581.750**	577.970	591.430	529.972	538.607	481.317
Média	**715.403**	617.266	522.151	557.787	615.010	424.006
Baixa	**763.784**	486.537	598.077	478.224	541.751	499.105

APÊNDICE F – MÉTRICA MEMÓRIA ORDENADA

Tabela 38: Experimentos da carga variável Aleatória 1 ordenados pela memória.

Posição	Configuração	Memória (MB)	Módulos Preventivos
1ª	Base	472.878	-
2ª	RF	474.878	Quarentena
3ª	RF	478.704	Ambos
4ª	XGBoost	503.914	Ambos
5ª	RF	511.431	Nenhuma
6ª	XGBoost	513.468	Quarentena
7ª	MLP	545.829	Quarentena
8ª	XGBoost	553.516	Nenhuma
9ª	RF	567.822	Contenção de Adaptações Sucessivas
10ª	XGBoost	579.383	Contenção de Adaptações Sucessivas
11ª	MLP	621.720	Nenhuma
12ª	HPA	678.259	-
13ª	MLP	700.797	Ambos
14ª	SVR	720.529	Ambos
15ª	SVR	720.570	Quarentena
16ª	MLP	722.079	Contenção de Adaptações Sucessivas
17ª	SVR	812.798	Contenção de Adaptações Sucessivas
18ª	SVR	877.050	Nenhuma

Tabela 39: Experimentos da carga variável Aleatória 2 ordenados pela memória.

Posição	Configuração	Memória (MB)	Módulos Preventivos
1ª	Base	425.589	-
2ª	XGBoost	455.435	Nenhuma
3ª	RF	459.286	Ambos
4ª	XGBoost	459.953	Ambos
5ª	XGBoost	462.387	Contenção de Adaptações Sucessivas
6ª	RF	463.376	Contenção de Adaptações Sucessivas
7ª	XGBoost	467.690	Quarentena
8ª	RF	479.204	Nenhuma
9ª	RF	483.859	Quarentena
10ª	SVR	599.252	Ambos
11ª	HPA	605.471	-
12ª	MLP	623.351	Quarentena
13ª	SVR	651.681	Quarentena
14ª	MLP	660.352	Contenção de Adaptações Sucessivas
15ª	SVR	714.300	Contenção de Adaptações Sucessivas
16ª	MLP	715.571	Nenhuma
17ª	MLP	750.106	Ambos
18ª	SVR	1009.308	Nenhuma

Tabela 40: Experimentos da carga fixa Alta ordenados pela memória.

Posição	Configuração	Memória (MB)	Módulos Preventivos
1ª	XGBoost	473.545	Contenção de Adaptações Sucessivas
2ª	Base	481.317	-
3ª	XGBoost	481.716	Ambos
4ª	XGBoost	519.684	Quarentena
5ª	RF	529.972	Ambos
6ª	RF	535.517	Quarentena
7ª	SVR	538.607	Contenção de Adaptações Sucessivas
8ª	RF	552.606	Nenhuma
9ª	MLP	553.217	Ambos
10ª	SVR	559.290	Quarentena
11ª	RF	577.970	Contenção de Adaptações Sucessivas
12ª	SVR	587.244	Nenhuma
13ª	HPA	591.430	-
14ª	MLP	607.984	Quarentena
15ª	SVR	611.311	Ambos
16ª	XGBoost	639.895	Nenhuma
17ª	MLP	645.104	Contenção de Adaptações Sucessivas
18ª	MLP	740.346	Nenhuma

Tabela 41: Experimentos da carga fixa Média ordenados pela memória.

Posição	Configuração	Memória (MB)	Módulos Preventivos
1ª	Base	424.006	-
2ª	XGBoost	444.981	Quarentena
3ª	XGBoost	471.594	Nenhuma
4ª	RF	503.177	Quarentena
5ª	HPA	522.151	-
6ª	MLP	532.763	Nenhuma
7ª	MLP	534.777	Contenção de Adaptações Sucessivas
8ª	MLP	541.555	Quarentena
9ª	RF	547.706	Nenhuma
10ª	XGBoost	555.028	Contenção de Adaptações Sucessivas
11ª	RF	557.787	Contenção de Adaptações Sucessivas
12ª	RF	596.056	Ambos
13ª	XGBoost	615.010	Ambos
14ª	MLP	617.266	Ambos
15ª	SVR	649.750	Quarentena
16ª	SVR	715.403	Nenhuma
17ª	SVR	754.942	Ambos
18ª	SVR	789.084	Contenção de Adaptações Sucessivas

Tabela 42: Experimentos da carga fixa Baixa ordenados pela memória.

Posição	Configuração	Memória (MB)	Módulos Preventivos
1ª	XGBoost	434.088	Contenção de Adaptações Sucessivas
2ª	XGBoost	436.002	Ambos
3ª	MLP	452.796	Nenhuma
4ª	RF	462.532	Quarentena
5ª	XGBoost	469.150	Quarentena
6ª	MLP	474.684	Contenção de Adaptações Sucessivas
7ª	RF	478.224	Nenhuma
8ª	MLP	484.037	Ambos
9ª	MLP	486.537	Quarentena
10ª	SVR	491.313	Ambos
11ª	Base	499.105	-
12ª	RF	516.959	Ambos
13ª	RF	535.455	Contenção de Adaptações Sucessivas
14ª	XGBoost	541.751	Nenhuma
15ª	SVR	548.390	Quarentena
16ª	HPA	598.077	-
17ª	SVR	628.908	Contenção de Adaptações Sucessivas
18ª	SVR	763.784	Nenhuma

APÊNDICE G – MÓDULOS PREVENTIVOS

Tabela 43: Módulos preventivos superiores ao HPA na carga variável Aleatória 1.

Posição	Configuração	RTT (ms)	Módulo Preventivo	Pontuação
1ª	MLP	165.629	Ambos	6
2ª	SVR	185.966	Ambos	5

Tabela 44: Módulos preventivos superiores ao HPA na carga variável Aleatória 2.

Posição	Configuração	RTT (ms)	Módulo Preventivo	Pontuação
1ª	SVR	138.874	Contenção de Adaptações Sucessivas	6

Tabela 45: Módulos preventivos superiores ao HPA na carga fixa Alta.

Posição	Configuração	RTT (ms)	Módulo Preventivo	Pontuação
1ª	SVR	220.789	Ambos	6
2ª	SVR	235.040	Quarentena	5
3ª	SVR	258.577	Nenhum	4
4ª	SVR	268.069	Contenção de Adaptações Sucessivas	3
5ª	MLP	431.434	Contenção de Adaptações Sucessivas	2
6ª	MLP	483.291	Ambos	1

Tabela 46: Módulos preventivos superiores ao HPA na carga fixa Média.

Posição	Configuração	RTT (ms)	Módulo Preventivo	Pontuação
1ª	SVR	146.914	Nenhum	6
2ª	SVR	198.703	Quarentena	5
3ª	SVR	199.840	Ambos	4
4ª	SVR	201.451	Contenção de Adaptações Sucessivas	3
5ª	MLP	309.417	Ambos	2

Tabela 47: Módulos preventivos superiores ao HPA na carga fixa Baixa.

Posição	Configuração	RTT (ms)	Módulo Preventivo	Pontuação
1ª	SVR	114.874	Nenhum	6
2ª	SVR	131.937	Quarentena	5
3ª	SVR	135.354	Contenção de Adaptações Sucessivas	4
4ª	SVR	145.783	Ambos	3
5ª	MLP	192.343	Quarentena	2

Tabela 48: Pontuação dos módulos preventivos por carga de trabalho.

	Módulos Preventivos			
Configuração	Nenhum	Quarentena	Contenção de Adaptações Sucessivas	Ambos
Aleatória 1	0	0	0	11
Aleatória 2	0	0	6	0
Alta	4	5	5	7
Média	6	5	3	6
Baixa	6	7	4	3