UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

# Aplicações de Sistemas de Transporte Inteligente em um Cenário de Computação de Borda Móvel

Thiago Correia Medeiros

**Orientador**

Dsc. Carlos Alberto Vieira Campos

RIO DE JANEIRO, RJ - BRASIL
DEZEMBRO DE 2020

Aplicações ITS em um Ambiente de Computação em Neblina e um Cenário de Computação de Borda Móvel

THIAGO CORREIA MEDEIROS

DISSERTAÇÃO APRESENTADA COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE MESTRE PELO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA DA UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO (UNIRIO). APROVADA PELA COMISSÃO EXAMINADORA ABAIXO ASSINADA.

Aprovada por:

---

Dsc. Carlos Alberto Vieira Campos — UNIRIO

---

Dsc. Sidney Cunha de Lucena — UNIRIO

---

Dsc. Kleber Vieira Cardoso — UFG

---

Dsc. Sand Luz Correa — UFG

RIO DE JANEIRO, RJ - BRASIL
DEZEMBRO DE 2020.

Medeiros, Thiago Correia
M929 Aplicações ITS em um Ambiente de Computação em Neblina e um Cenário de Computação de Borda Móvel / Thiago Correia Medeiros, 2020.
xiii, 187f.

Orientador: Carlos Alberto Vieira Campos
Dissertação (Mestrado em Informática) - Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2018.

1. Sistemas de Informação - Requisitos - Reconlhimento de Padrões I. Carlos Alberto Vieira Campos. II. Universidade Federal do Estado do Rio de Janeiro (2003-). Centro de Ciências Exatas e Tecnologia. Curso de Mestrado em Informática. Título.

CDD - 004.678

*Dedico essa dissertação à minha esposa Jaqueline, que me deu apoio e incentivo nas horas difíceis de desânimo e cansaço.*

# Agradecimentos

Agradeço a minha esposa Jaqueline e minha enteada Letícia que abriram mão junto comigo de vários momentos de lazer, que exercitaram a paciencia em diversos momentos difíceis, pois aturaram meu mau humor e cansaço.

Aos meus amigos Bruno Marques e Cristiano Samel, pelas alegrias, tristezas e dores compartilhadas.

Aos meus amigos Estevan Barbará, Adriano Mounthé e Marc Chevallier que me ajudaram nessa caminhada durante as aulas.

Aos meus amigos Elton Soares e Tiago Saraiva que me ajudaram na avaliação contínua do meu trabalho, revisões e várias ideias de implementações.

Agradeço também ao professor Carlos Alberto Vieira Campos, que me recebeu de braços abertos, mesmo sabendo que eu não tinha experiência alguma na área acadêmica, abraçou as minhas ideias de projeto de pesquisa, sempre atento nas suas orientações e, com sua vasta experiência acadêmica, permitiu a produção deste trabalho científico.

Medeiros, Thiago Correia, **Aplicações de Sistemas de Transporte Inteligente em um Cenário de Computação de Borda Móvel**. UNIRIO, 2020. 81 páginas. Dissertação de Mestrado. Departamento de Informática Aplicada, UNIRIO.

## RESUMO

O uso de abordagens de Computação em Borda é uma forma de aprimorar os aplicativos de Sistemas de Transporte Inteligente (ITS) em cidades inteligentes. Aumentar a eficiência em elasticidade de aplicações é uma forma de melhorar a entrega de serviços para os consumidores finais. Este trabalho propõe uma arquitetura chamada ARTSIA (Architecture for Response Time Sensitive ITS Applications), que possui uma arquitetura de Computação em Borda Móvel (MEC) para apoiar aplicações para ITS com requisitos mais rígidos de latência e perdas de requisições. Visando garantir o mínimo de perdas possíveis em requisições oriundas da camada terminal, ARTSIA conta com a proposta de um algoritmo baseado nas redes neurais autorregressivas e em limiares adaptativos de elasticidade, para analisar os recursos dos servidores em uma solução de computação em neblina e indicar o melhor momento para aumentar ou diminuir a capacidade da infraestrutura que suporta a aplicação de acordo com a demanda gerada em função das variações em sua carga de trabalho. Além disso, fazendo uso de técnicas e políticas de elasticidade, este trabalho apresenta uma avaliação experimental de diferentes cenários para gerenciar a elasticidade. Este trabalho também propõe um estudo de caso de um sistema ITS com arquitetura MEC usando o TSITS (Time-Sensitive Intelligent Transportation System), que é uma variante do ARTSIA, distribuindo as responsabilidades de computação com a camada terminal. A arquitetura tem como objetivo oferecer suporte a aplicativos ITS com requisitos rígidos de latência e perda de solicitações. Avaliamos um aplicativo ITS em três cenários diferentes: na nuvem usando uma conexão 4G, na nuvem usando banda larga e no Computação de Borda Móvel (MEC). Os resultados experimentais indicam que o MEC tem menor perda de solicitações e pode trabalhar com um maior número de solicitações por segundo, o que o torna uma boa solução para aplicações ITS.

**Palavras-chave:** sistemas de transportes inteligente, computação de borda, computação em neblina, computação de borda móvel, elasticidade de aplicações, aplicações sensíveis ao tempo

## ABSTRACT

Using Edge Computing approaches is a way to enhance Intelligent Transport Systems (ITS) applications in smart cities. Increasing the efficiency of application elasticity is a way to improve service delivery to end consumers. This work proposes a architecture named ARTSIA (Architecture for Response Time Sensitive ITS Applications), which has a Mobile Edge Computing (MEC) architecture to support applications for ITS with stricter requirements for latency and loss of requests. In order to guarantee the minimum possible losses in requests coming from the terminal layer, ARTSIA has the proposal of an algorithm based on autoregressive neural networks and adaptive elasticity thresholds, to analyze the server resources in a fog computing solution and indicate the best time to increase or decrease the capacity of the infrastructure that supports the application according to the demand generated due to variations in your workload. In addition, using elasticity techniques and policies, this paper presents an experimental assessment of different scenarios to manage elasticity. This work also proposes a case study of an ITS system with MEC architecture using TSITS (Time-Sensitive Intelligent Transportation System), which is a variant of ARTSIA, distributing computing responsibilities with the terminal layer. The architecture aims to support ITS applications with strict latency requirements and loss of requests. We evaluated an ITS application in three different scenarios: in the cloud using a 4G connection, in the cloud using broadband and in the MEC. The experimental results indicate that the MEC has less loss of requests and can work with a greater number of requests per second, which makes it a good solution for ITS applications.



**Palavras-chave:** intelligent transport systems, edge computing, fog computing, mobile edge computing, application elasticity, time-sensitive applications

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Lista de Nomenclaturas

| | |
|---|---|
| 4G | 4ª Generation |
| 5G | 5ª Generation |
| AR | Auto-regressive |
| AP | Access Point |
| ARM | Acorn RISC Machine |
| ARNN | Auto-regressive Neural Networks |
| ARMA | Auto-regressive Moving Average |
| ARTSIA | Architecture for Response Time-Sensitive ITS Applications |
| CC | Cloud Computing |
| CoAP | Constrained Application Protocol |
| CSV | Comma Separeted Values |
| CTMC | Continuous Time Markov Chain |
| DL | Deep Learning |
| DTMC | Discrete Time Markov Chain |
| EC | Edge Computing |
| FAC | Função de Autocorrelação |
| FC | Fog Computing |
| GHz | Giga Hertz |
| GP | Gigabyte |
| GPS | Geographic Position System |
| HTTP | Hypertext Transfer Protocol |
| i5 | Intel 5ª Generation |
| IoT | Internet of Things |
| ITS | Intelligent Transportation System |
| LAN | Local Area Network |
| LTS | Long Term Support |

| | |
|---|---|
| M2M | Machine-to-Machine |
| MA | Moving Average |
| MDP | Markov Decision Process |
| ML | Machine Learning |
| MEC | Mobile Edge Computing |
| OBU | On-Board Unit |
| POI | Point-of-Interest |
| TCP | Transmission Control Protocol |
| TSITS | Time-Sensitive Intelligent Transportation System |
| RAM | Random Access Memory |
| REST | Representational State Transfer |
| RL | Reinforcement Learning |
| SABANN | Scalability Algorithm Based on Neural Networks |
| SO | System Operational |
| SSH | Security Shell |
| VCC | Vehicular Cloud Computing |
| VM | Virtual Machine |
| VNF | Virtual Network Function |

# Lista de Símbolos e Unidades

| SIGLA | UN. | SIGNIFICADO |
|---|---|---|
| Tad | s | Intervalo de execução do Atualizador de Domínios do ARTSIA |
| Tesa | s | Tempo de inicialização de uma instância do ARTSIA |
| Tcf | s | Intervalo de execução no Coletor Fog do ARTSIA |
| Tcc | s | Intervalo de execução do Transmissor do ARTSIA |
| To | s | Intervalo de execução do Orquestrador Fog do ARTSIA |
| Ta | s | Intervalo de execução do Ativador do ARTSIA |
| Pa | un | Pontos de base do algoritmo do SABANN |
| Pp | un | Ponto de previsão do algoritmo do SABANN |
| Qtd | un | Quantidade de medidas utilizadas pelo SABANN |
| Tmd | s | Tempo mínimo de disponibilidade de uma instância |
| Tal | s | Intervalo de execução do Atualizador de Domínios do TSITS |
| Tabe | s | Intervalo de execução do Atualizador de Instruções Edge do TSITS |
| Tce | s | Intervalo de execução do Coletor Edge TSITS |
| Tt | s | Intervalo de execução do Transmissor do TSITS |
| Tae | s | Intervalo de execução do Ativador do TSITS |
| Toe | s | Intervalo de execução do Orquestrador Edge do TSITS |

# 1. Introdução

Estima-se que existam mais de 7,7 bilhões de pessoas no mundo, e este número tende a crescer [43]. Com esse aumento populacional, há o crescimento de cidades e da quantidade de pessoas que nelas vivem. Algumas cidades são planejadas e é possível suportar o aumento demográfico, porém, em sua maioria, as cidades não foram planejadas para conter todo o aumento que vem sendo observado. Uma das consequências do elevado número de pessoas nas cidades, é um volume maior de trânsito. Com o acúmulo de veículos transitando pelas cidades, despende-se de mais tempo para que uma pessoa chegue ao seu destino. Por exemplo, pessoas que moram em áreas mais afastadas dos grandes centros urbanos gastam uma grande proporção das horas do seu dia no trânsito para chegarem ao seu local de trabalho, quando em comparação às horas trabalhadas. Com relação ao funcionamento de alguns serviços públicos, o trânsito congestionado pode prejudicar o tempo de chegada de policiais em uma cena de crime, bombeiros que precisam de urgência para salvarem pessoas e, até mesmo, ambulâncias que transportam pessoas em situação de vida ou morte. As pessoas também são prejudicadas financeiramente, uma vez que, táxis e veículos de aplicativos aumentam suas taxas por conta do aumento do tempo das corridas, fazendo com o que o consumidor pague mais.

Se tratando de dispositivos móveis, existem 3,5 bilhões de smartphones no mundo e em 2021 esse número chegará a 3,8 bilhões [51]. Se considerarmos todos os dispositivos móveis, esse número também aumenta significativamente. Esses dispositivos conectam pessoas e coisas em todo o planeta, e têm o potencial de melhorar as rotas de pedestres e veículos nas cidades. Os pedestres podem utilizar celulares, smartphones e tablets como instrumentos, tanto de prestação de informações, quanto de consumo de serviços disponíveis na cidade. Os veículos podem ter dispositivos de sistemas de posicionamento global (Global Positioning System - GPS), bem como unidades de bordo (On-board Unit - OBU) para comunicarem com os Sistemas de Transporte Inteligentes (Intelligent Transportation Systems - ITS).

Cidades Inteligentes (Smart Cities) é uma grande área de estudo e desenvolvimento, na qual, pesquisadores utilizam o método científico para oferecer propostas de melhorias baseadas em dados coletados de vários pontos da cidade e integrações de serviços de forma que os cidadãos tenham melhoria na sua qualidade de vida. Especificamente com relação a movimentação das pessoas na cidade, ITS, é uma área de estudo onde se visa coletar dados das cidades, analisá-los, integrá-los com outros sistemas para poder refinar as informações para possibilitar uma maior qualidade em relação ao transporte para as pessoas [40]. Existem algumas soluções ITS que utilizam Computação na Nuvem (Cloud Computing - CC) para permitir a interação entre todos os dispositivos dentro da cidade. O trabalho [10] investiga como diminuir o tempo de serviços computacionais em Computação em Nuvem Veicular (Vehicular Cloud Computing - VCC). Já no artigo [22], é introduzido um paradigma de VCC que propõe um modelo de coleta de dados que beneficiam ITS. Os autores em [35] apresentam uma arquitetura de CC para suportar o gerenciamento de veículos em grandes cidades.

Oferecer alguns tipos de serviços para ITS através de CC, por vezes, podem se tornar ineficientes em algumas cidades, por conta do longo tempo de resposta aos serviços. Computação de Borda (Edge Computing - EC) é um paradigma de computação que trabalha como uma camada intermediária entre a nuvem e os consumidores. Duas características que contribuem para adoção de EC são: alta largura de banda e baixa latência. EC trabalha na borda da rede, podendo acessar a nuvem ou não. Nem todas as aplicações precisam enviar dados à nuvem para realizar seus objetivos. Uma tomada de decisão na própria rede e uma resposta mais rápida podem ser a solução para algumas aplicações que visam melhorar o transporte. Os autores em [23] classificam EC em três paradigmas diferentes de acordo com seu objetivo, Computação em Neblina (Fog Computing - FC), Computação de Borda Móvel (Mobile Edge Computing - MEC) e Cloudlets.

Na literatura, podemos encontrar vários trabalhos que relacionam FC e ITS. O artigo [14] propõe um sistema de coleta de dados através de sensores em ônibus urbanos. O trabalho [64] apresenta o CFC-IoV, uma arquitetura cooperativa baseada em FC para redes de veículos inteligentes. Já os autores em [36], apresentam a plataforma chamada FogFly, baseada em FC para resolver problemas de otimização de semáforos.

Também encontramos trabalhos anteriores que relacionam MEC e ITS como o VTracer que apresentaram métodos para compressão de trajetórias online e rastreamento de veículos [11]. Os autores em [66], desenvolveram um sensor de multidão móvel robusto utilizando Aprendizado Profundo (Deep Learning - DL) e MEC. Por fim, o trabalho [16] apresenta uma proposta de compartilhamento de recursos de computação acessíveis usando redes multiacesso.

Antes de desenvolver qualquer solução com EC é preciso considerar suas limitações. Dentre elas podemos citar a capacidade restrita de processamento e armazenamento. Sistemas baseados em borda possuem recursos limitados, logo, precisam ser planejados. Além de planejados seus recursos precisam ser muito bem gerenciados. Uma infraestrutura de borda pode suportar várias aplicações, algumas aplicações podem ter mais utilização que outras em determinados momentos do dia. Sendo assim torna-se necessário alocar mais recursos para uma aplicação quando aumenta sua demanda de uso, assim como é preciso liberar recursos dessa aplicação quando sua demanda diminui, para que outras aplicações possam fazer uso dos recursos. A literatura denomina esse aumento ou liberação de recursos de elasticidade.

## 1.1 Motivação

ITS são uma área de estudos onde são buscadas soluções para melhorar a qualidade de vida das pessoas nas cidades. EC é uma tecnologia que pode apoiar ITS uma vez que ela oferece maior largura de banda e menor latência, com melhorias relevantes na qualidade de resposta aos consumidores, sejam eles veículos ou pedestres através de dispositivos móveis.

Devido a várias características diferentes de aplicações ITS, EC pode ser utilizado através de suas diferentes variações para solucionar problemas específicos como FC e MEC. Desta forma, torna-se necessário entender essas variações para buscar melhorar algumas de suas deficiências. Pesquisas existentes sobre EC apresentam aplicações, direções e desafios para FC e MEC [6, 23, 27, 29, 45, 59].

O autor em [39] cita que o assunto de elasticidade em FC ainda é um tema a ser trabalhado. A pesquisa [55] utiliza de FC para estender a elasticidade de plataformas IoT/M2M em ambientes industriais. Ainda na área industrial, o artigo [56] apresenta uma plataforma de virtualização integrada, onde também é apresentada uma proposta de elasticidade de aplicações baseada em lógica fuzzy. Com recursos divididos em uma universidade na Itália e outra na Suécia, os autores em [62] realizam um experimento onde exibe uma arquitetura que trabalha com elasticidade utilizando um algoritmo geométrico.

Com relação ao gerenciamento de recursos no MEC, a necessidade de estratégias para virtualizar e gerenciar a alocação de recursos em servidores de borda tem sido destacada na literatura, e embora alguns trabalhos proponham arquiteturas genéricas sobre alocação de recursos em servidores de borda, alguns autores afirmam que faltam propostas arquitetônicas mais detalhadas [6, 59]. Focando em veículos, o trabalho [27] argumenta

que a pesquisa sobre Computação de Borda Veicular (Vehicular Edge Computing - VEC) ainda está em estágio embrionário e cita o compartilhamento de recursos em servidores de borda como um desafio aberto.

Motivados pelos trabalhos existentes, entendemos ser necessário buscar soluções que utilizem elasticidade de aplicações para EC, de forma, a melhorar a qualidade de respostas ao consumidor final. Também entendemos ser necessário a apresentação de propostas de arquiteturas mais detalhadas para EC e que visem melhorar o compartilhamento de recursos em servidores de borda. E, por último, como utilizar estes paradigmas para prover soluções para ITS.

## 1.2 Definição do Problema

Aplicações ITS que tenham o valor da informação sensível ao tempo precisam entregar mensagens com um mínimo possível de tempo de resposta. Em uma época em que as cidades têm cada vez mais diferentes meios de transporte e pessoas utilizando dispositivos móveis, a demanda desses dispositivos por recursos computacionais é cada vez maior. Paradigmas de computação de EC auxiliam na solução deste problema por apresentarem melhores desempenhos de largura de banda e latência em relação a CC. Porém, apesar da melhora com esses paradigmas, o problema se constitui em: (i) - *como realizar uma utilização adequada desses paradigmas*, (ii) - *como diminuir o tempo de resposta ao consumidor final, diminuindo as requisições perdidas através de elasticidade de aplicações*.

## 1.3 Justificativa

Com base na revisão da literatura realizada, foi observado que apesar de existir alguns trabalhos na área acadêmica sobre elasticidade de aplicações em EC, até onde sabemos, este é o primeiro trabalho a propor arquiteturas baseadas em EC, que gerenciam suas aplicações fazendo uso de técnicas de elasticidade para atender aplicações de ITS sensíveis ao tempo. Algumas aplicações ITS possuem serviços onde o tempo de resposta influencia diretamente no valor da informação, de forma a satisfazer as requisições de veículos ou usuários de dispositivos móveis para uma melhor tomada de decisão. Ainda assim, apesar das soluções já existem de arquiteturas propostas para ITS no geral, faltam propostas de arquiteturas detalhadas para uma melhor compreensão e replicação.

## 1.4 Objetivo

O objetivo deste trabalho é estudar os paradigmas de EC e como escalar aplicações ITS de forma a reduzir a quantidade de requisições perdidas dos consumidores finais. Baseado em problemas em abertos, também serão apresentadas propostas de arquiteturas detalhadas de MEC, de forma que o leitor tenha total compreensão da atuação deste paradigma e como ele corrobora junto a técnica de elasticidade de aplicações para entregar serviços de qualidade ao consumidor final. Por último, foram utilizados experimentos reais para demonstrar como nossa solução tem atingido o objetivo de elasticidade de aplicações com a maior eficácia do que os trabalhos do estado da arte.

## 1.5 Contribuições

Como pode ser visto na Seção 1.1, existem alguns desafios em abertos relacionados a detalhamento de arquiteturas EC e elasticidade de aplicações de forma que possa alcançar o objetivo de entregar o melhor tempo de resposta possível em aplicações ITS sensíveis ao tempo. Por esta razão correlaciona-se abaixo os desafios em aberto e as contribuições deste trabalho:

1. Propostas de arquiteturas detalhadas de como aplicar os paradigmas de EC com aplicações ITS:
   (a) Foi apresentado uma proposta de arquitetura chamada ARTSIA (Architecture for Response Time-Sensitive ITS Applications) onde tem por objetivo detalhar todos os componentes, suas atuações e suas integrações necessárias para suportar aplicações ITS sensíveis ao tempo utilizando MEC;
   (b) Também foi apresentada uma proposta de arquitetura chamada TSITS (Time-Sensitive Intelligent Transportation System), que é uma variante do ARTSIA, distribuindo as responsabilidades de computação com a camada terminal.
2. Como reduzir a quantidade de requisições perdidas de aplicações ITS sensíveis ao tempo através de elasticidade de aplicações:
   (a) Foi apresentada uma proposta de algoritmo baseada em redes neurais para elasticidade (Scalability Algorithm Based on Neural Networks - SABANN) que trabalha em uma arquitetura baseada em MEC, utiliza como suporte uma técnica de previsão de séries temporais com redes neurais chamada redes neu-

rais auto-regressivas (Autoregressive Neural Networks - ARNN) e faz uso de limiares adaptativos de elasticidade;

(b) Foi realizado um estudo utilizando as principais técnicas de Aprendizado de Máquina (Machine Learning - ML) sobre um conjunto de dados gerado a partir dos experimentos utilizando componentes reais neste trabalho de forma a obter a técnica que melhor atende a elasticidade de aplicações em EC.

3. Realização de experimentos utilizando componentes reais:

   (a) Avaliação experimental da arquitetura ARTSIA comparando a desempenho do algoritmo SABANN em relação as propostas existentes no estado da arte;

   (b) Um estudo de caso de uma aplicação ITS, rodando em cima de uma arquitetura MEC, em que o tempo de resposta influencia diretamente o valor da informação;

   (c) Avaliação experimental da arquitetura TSITS utilizando o algoritmo SABANN, de forma a comparar a desempenho de uma aplicação ITS sensível ao tempo em nossa arquitetura MEC em relação a desempenho dessa aplicação utilizando CC, tanto com conexão através de banda larga, quando com conexão 4G.

As contribuições deste trabalho relacionadas ao ARTSIA foram publicadas em [32] e, após a evolução do trabalho, foi submetido o artigo [33]. Já as contribuições relacionadas ao TSITS foram submetidas em [34].

## 1.6 Estrutura do Trabalho

O Capítulo 2 fornece uma exploração sobre os conceitos preliminares necessários para compreensão do presente trabalho. Dentre tais conceitos são apresentadas as arquiteturas hierárquicas de FC e MEC, assim como, suas camadas, modos de virtualização utilizados por servidores de borda, métodos e políticas de elasticidade de aplicações, modelos auto-regressivos para séries temporais, a técnica de rede neural ARNN e a modelagem por Cadeias de Markov.

O Capítulo 3 apresenta a arquitetura ARTSIA, os trabalhos relacionados a elasticidade de aplicações do estado da arte, a proposta de uma arquitetura especialista para MEC e aplicações ITS sensíveis ao tempo, o algoritmo SABANN e uma avaliação experimental utilizando componentes reais comparando o desempenho de elasticidade do algoritmo SABANN com as técnicas do estado da arte em EC.

O Capítulo 4 detalha uma investigação realizada sobre o conjunto de dados obtido no experimento do Capítulo 3. Nesta investigação, foi realizada uma uma exploração dos dados do conjunto de dados, assim como foram aplicados os principais algoritmos de ML.

O Capítulo 5 apresenta a arquitetura TSITS, assim como, a proposta de uma arquitetura especialista para MEC e aplicações ITS sensíveis ao tempo, um estudo de caso de uma aplicação ITS sensível ao tempo rodando sobre o TSITS e uma avaliação experimental comparando o desempenho de uma aplicação ITS no TSITS em relação a uma atuação em CC, tanto com conexão banda larga, quanto com conexão 4G.

No Capítulo 6 conclui-se o trabalho. Descreve como o ARTSIA e o SABANN foram mais eficientes que o estado da arte. Também é descrito os resultados da investigação realizada com ML. Apresenta as considerações finais sobre o experimento comparando MEC com CC e, por fim, apresenta sugestões para o futuro da pesquisa.

# 2. Fundamentação Teórica

O objetivo deste capítulo é fornecer os conceitos preliminares necessários para compreensão do presente trabalho. Na Seção 2.1 introduz-se o conceito de Computação de Borda, assim como, suas variações que foram utilizadas como base na arquitetura do presente trabalho. A Seção 2.2 expõe métodos e políticas de elasticidade de aplicações que foram utilizadas nos experimentos realizados. Na Seção 2.3 são abordados o conceito de ITS e como as aplicações ITS atuam. Técnicas de Séries Temporais exibidas na Seção 2.4 e de Cadeia de Markov na Seção 2.6 foram utilizadas no algoritmo do experimento do Capítulo 3. Por fim, a Seção 2.5 contém redes neurais que foram utilizadas no algoritmo proposto deste trabalho no Capítulo 3 e na investigação realizada no Capítulo 4.

## 2.1 Computação de Borda

A Computação de Borda (Edge Computing - EC) é uma nova tendência no cenário de computação. EC atua como intermediário entra a Computação na Nuvem (Cloud Computing - CC) e a camada de consumidores finais e tem como objetivos trazer serviços e utilitário da CC para borda da rede. O resultado disso são aplicações que oferecem tempo de resposta mais rápido que se comparado a aplicações CC.

As principais características EC são suporte de mobilidade, reconhecimento de localização, latência ultrabaixa e proximidade com o usuário [23, 27]. Essas características habilitam a EC a se tornar um paradigma promissor para uma série de aplicações como automação industrial, realidade virtual, monitoramento de tráfego em tempo real, casa inteligente, monitoramento marítimo e uma série de aplicações relacionadas com análise de dados veloz para uma melhor tomada de decisão.

EC pode ter suas variações e entre elas estão Computação em Neblina e Computação de Borda Móvel.

### 2.1.1 Computação em Neblina

Computação em Neblina (Fog Computing - FC) é um paradigma de computação que tem por objetivo aproximar os serviços comumente oferecidos na CC para perto da borda da rede. Serviços esses que podem ser de computação, armazenamento, gerenciamento de tarefas, entre outros. Os autores em [39] fazem uma alusão de que neblina é a nuvem mais próxima ao chão. Artigos de pesquisa como [1, 39] informam que soluções de Computação em Neblina devem seguir a arquitetura hierárquica disposta na Figura 2.1.

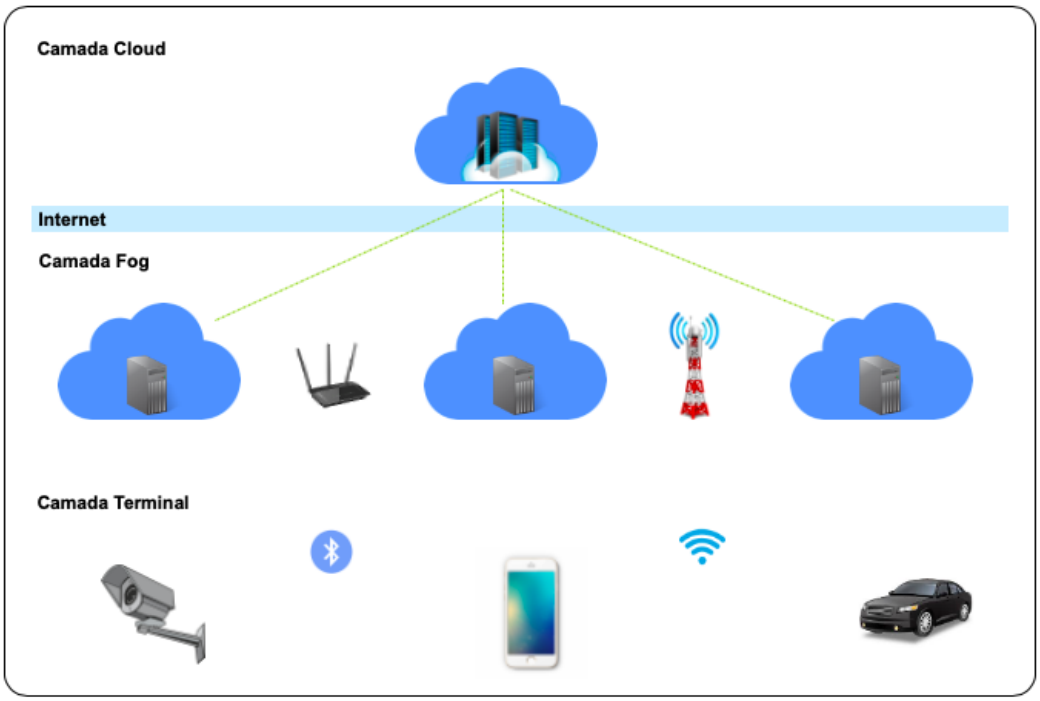


Figura 2.1: Arquitetura hierárquica de computação em neblina composta pelas camadas de nuvem, de neblina e terminal.

É chamado de camada na nuvem (camada cloud) onde se encontra a maior capacidade computacional. Ela é construída por inúmeros datacenters com todo tipo de serviço de computação. Essa camada permite que sistemas de neblina estendam parte de trabalho para a nuvem. Os artigos [4, 14, 19, 54] utilizam a camada na nuvem como local de armazenamento de dados que vão ser utilizados para processamentos futuros, como oferecimento de outros serviços baseados nos dados, aplicação de Aprendizado de Máquina (Machine Learning - ML) ou analítica de dados (data analytics). Apesar da camada da nuvem estender soluções para algumas situações e fazer parte da arquitetura hierárquica, seu uso não é obrigatório em um sistema de neblina.

A camada de neblina (camada fog) é onde se encontram as aplicações do sistema de neblina. Aplicações de neblina podem dar uma resposta mais rápida ao consumidor por

razão de estarem na mesma rede. Uma característica dos sistemas de neblina é que funcionam sobre redes sem fio. Pelo fato de estar na borda da rede, a camada de neblina possui recursos limitados de computação, armazenamento e serviços de rede em relação a camada da nuvem. Porém, consegue oferecer uma taxa de latência melhor. Os trabalhos apresentados em [14,28,56,62] utilizam a camada de neblina para pequenos processamentos, armazenamentos e uma rápida tomada de decisão que podem envolver uma resposta a camada terminal.

A camada terminal é onde estão os consumidores das aplicações de neblina. O artigo [14] nomeia essa camada como camada sensora, já os autores em [55, 62] denominam como camada de Internet das Coisas (Internet Of Things - IoT). De fato, essa camada é composta de sensores, dispotivos IoT, dispositivos móveis como celular, tablets, veículos, etc.

Na solução apresentada no Capítulo 3, a camada de nuvem é utilizada para fazer a orquestração dos servidores de neblina, no entanto, a possível interrupção desta camada não torna o sistema de neblina indisponível. Ainda no Capítulo 3, a camada de neblina é utilizada para hospedar aplicações de neblina com serviços para ITS e na camada terminal os consumidores seriam os veículos.

### 2.1.2 Computação de Borda Móvel

Computação de Borda Móvel (Mobile Edge Computing - MEC) é um paradigma de computação que visa fornecer serviços de computação com foco em dispositivos móveis com alta largura de banda, baixa latência, melhor qualidade de serviço e experiência do usuário. Esses serviços incluem serviços de processamento, armazenamento e gerenciamento de tarefas, entre outros. Os servidores MEC trabalham na borda da rede, podem se interconectar com vários serviços na nuvem e são uma tecnologia chave para o desenvolvimento de 5G no que diz respeito a problemas como atrasos, programação e elasticidade [45, 59].

As soluções MEC típicas seguem a arquitetura mostrada na Figura 2.2, onde a camada de nuvem é a que possui a maior capacidade computacional e pode ser composta por vários centro de dados que suportam todos os tipos de serviços de computação. Além disso, essa camada permite que os sistemas de neblina e sistemas de borda móvel estendam parte da carga de trabalho para a nuvem. Por exemplo, VTracer é um sistema que usa a camada de nuvem como local para armazenar dados compactados gerados a partir da trajetória dos veículos [11]. Os dados armazenados na nuvem podem ser usados para aprendizado de máquina ou análise de dados para extrair informações para futuras tomadas de decisão.

Embora a camada de nuvem estenda soluções para alguns aplicativos e faça parte da arquitetura do modelo, seu uso não é obrigatório em um sistema MEC. No caso de estudo apresentado por nosso trabalho, que pode ser visto na Seção 5.4, a nuvem é utilizada como fonte central de pontos de interesse (POIs) da cidade. De acordo com nossa arquitetura, apresentada na Seção 5.3, a nuvem também é utilizada como orquestradora de servidores de borda, o que significa que a possível interrupção desta camada não torna o sistema MEC indisponível.

A camada de borda é onde os aplicativos do sistema MEC são implantados. Os aplicativos MEC têm como objetivo fornecer respostas mais rápidas ao consumidor por estar na mesma rede, também conhecida como borda da rede. Essa camada tem processamento, armazenamento e serviços de rede limitados em comparação com a nuvem. Porém, consegue oferecer melhores condições de atendimento. A pesquisa [46] apresenta alguns cenários de servidores MEC que atendem a aplicativos de casas inteligentes, agricultura inteligente e cidades inteligentes.

A camada móvel é onde os consumidores e extensores de aplicativos MEC estão localizados. Esta camada é composta por dispositivos móveis como smartphones, tablets, navegadores GPS, veículos com OBUs, etc.

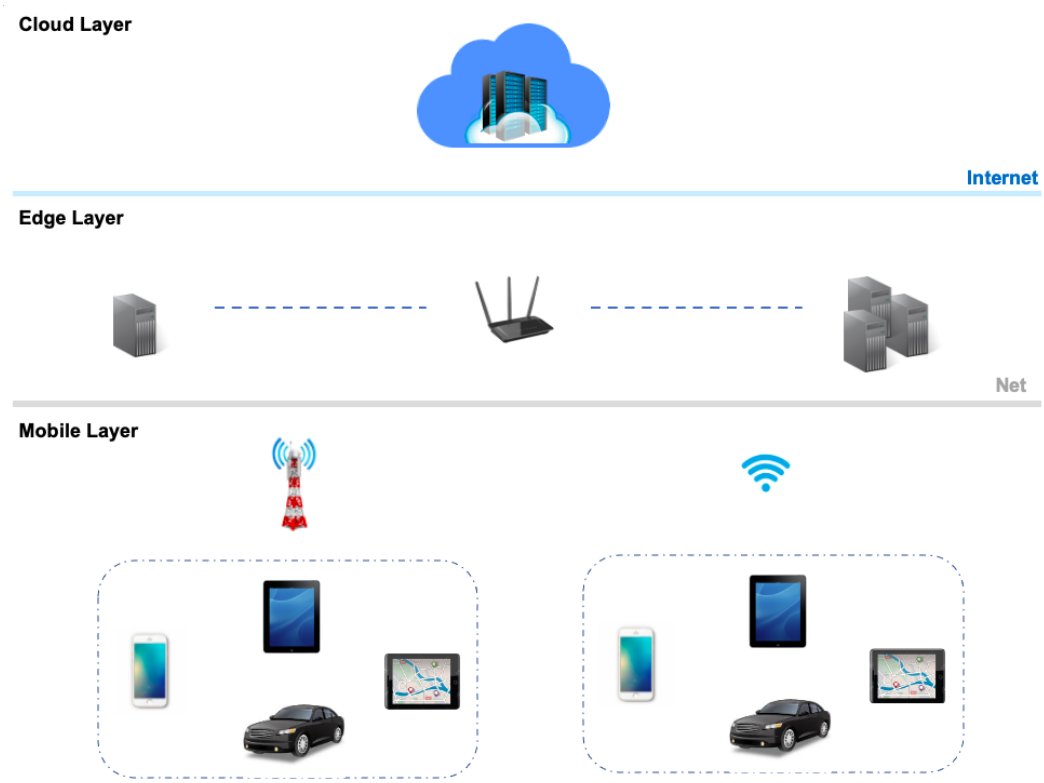


Figura 2.2: Componentes e camadas da arquitetura básica para Computação de Borda Móvel.

### 2.1.3 Computação em Neblina x Computação de Borda Móvel

Ambos paradigmas são especializações de EC, por isso a grande semelhança entre eles. Sua principal diferença está no atendimento a camada final de usuários. O MEC tem foco maior em dispositivos móveis, já o FC é mais abrangente e atende a todo tipo de dispostivos. Tais diferenças implicam em requisitos de aplicações diferentes. No presente trabalho utilizaremos MEC, uma vez que nossa aplicação tem foco em atender dispositivos móveis.

## 2.2 Elasticidade de Aplicações

Escalabilidade e elasticidade são termos utilizados na literatura para representar a capacidade de uma arquitetura redimensionar seus recursos para que as aplicações contidas nelas possam atender uma carga de trabalho oriunda da camada terminal variante ao longo do tempo. Como mostra a Figura 2.3, esse redimensionamento se dá no aumento de recursos para utilização da aplicação quando a carga de trabalho aumenta, o que chamamos de *scale-up*. Da mesma forma quando a carga de trabalho diminui, é preciso liberar recursos para que outras aplicações utilizem, que é o que chamamos de *scale-down*.


Figura 2.3: Representação do funcionamento de elasticidade, onde os servidores com aplicações variam sua quantidade em uso de acordo com a demanda.

Segundo os autores em [2], os métodos de elasticidade de serviços podem ser classificados como vertical ou horizontal. Escalar verticalmente um serviço, seria aumentar ou diminuir sua capacidade sem criar instâncias do mesmo. Por exemplo, o serviço de banco de dados é um serviço que precisa ser escalado verticalmente, para aumentar a capacidade do banco de dados, adiciona-se mais recurso de disco rígido a ele, mas continua-se

tendo somente uma instância de banco de dados. Escalar horizontalmente serviços seria duplicar sua instância e alocá-la para uma mesma aplicação. Por exemplo, escalar horizontalmente um servidor web, seria exatamente criar outra instância do servidor web e associá-la a uma aplicação.

Com relação as políticas que as plataformas utilizam para escalar recursos, o artigo [2] as classifica em: reativas ou proativas. Uma política reativa é quando a plataforma, uma vez já definidos os limiares (ao longo deste trabalho utilizaremos o termo limiares para referenciar o termo computacional *threshold*) para realizar um *scale-up* ou *scale-down*, é esperado que a utilização de medidas chegue a esse limiar para que a plataforma tome uma ação. Por exemplo, quando a utilização de processamento (cpu, mas iremos sempre nos referir como processamento) alcançar 80%, a plataforma toma as ações se aumente uma instância de determinado serviço. Já a política proativa, é quando a plataforma junto a utilização de algum método, tentar prever a aproximação dos limiares e age com antecedência.

Trabalhar com limiares de elasticidade e recursos é uma prática oriunda de CC. Os autores em [25] criaram uma aplicação de código aberto para analisar algumas características de CC, entre elas elasticidade. A infraestrutura utilizada foi a AWS Services[1]. Foi utilizado como limiar para *scale-up* quando o recurso de processamento alcança 80% e como limiar para *scale-down* em quando o processamento chega à 5%. O trabalho [63] propõe um estrutura de elasticidade baseado em contêineres. Ele utiliza os limiares de 70% para *scale-up* e 30% para *scale-down* tanto de utilização de processamento quanto para memória, mas cita que o processamento é um recurso mais relevante.

Porém alguns trabalhos de nuvem também apresentam o uso de uma política de elasticidade proativa. Foi apresentado no artigo [26], um modelo de elasticidade baseado em Cadeia de Markov de Tempo Contínuo utilizando um modelo de filas $M/M/m$. O artigo [17] também apresenta um algoritmo de controle de elasticidade da infraestrutura utilizando contêineres em CC. O algoritmo considera dois agentes: mestre e hospedeiro. O agente mestre coordena os hospedeiros e realiza a tomada de decisão. O hospedeiro monitora e deriva seu estado utilizando Médias Móveis Auto-regressivas.

Uma das características dos servidores de borda é que eles trabalham com virtualização. A literatura apresenta dois tipos de virtualização: baseada em VMs como em [44] e baseada em contêineres como em [56, 62] e nas nossa propostas dos Capítulos 3 e 5. Ambas abordagens permitem que se trabalhe com várias instâncias de serviços virtuais dentro de um mesmo servidor.

[1] https://aws.amazon.com/

### 2.2.1 Virtualização

Quando o assunto é computação existe a possibilidade de se recorrer a virtualização de recursos. A virtualização permite que se compartilhe ou particione recursos físicos em recursos virtuais. Com isso, é possível oferecer uma gama de serviços maior mesmo com uma limitação física de recursos, além de ser uma opção muito mais barata e escalável.

Entre as tecnologias que se destacam tanto na área acadêmica quanto no mercado, podemos citar Máquinas Virtuais (Virtual Machines - VM) e contêineres. A seguir abordaremos as características de cada tecnologia.

#### 2.2.1.1 Máquinas Virtuais

Quando trabalhamos com VMs, um ou mais discos rígidos virtuais podem ser criados dentro de um disco rígido físico. Esta ação permite que se reserve pedaços do disco rígido, de tal forma, que podemos representar cada um desses pedaços como um ambiente independentemente um do outro. Do ponto de vista do usuário uma VM é um sistema computacional como qualquer outro. Já na visão do núcleo a VM é um sistema puramente lógico. Cada uma dessas VMs tem configurado a utilização de uma parte de outros recursos como unidades de processamento e quantidade de memória. Os autores em [37] apresentam técnicas de gerenciamento de memória em sistemas virtualizados. Com relação a rede, pode ser configurado uma ou mais placas de redes virtuais para cada VM de forma a se obter uma rede de computadores virtualizados.

O software responsável pela criação de VMs se chama *hipervisor*, que é executado na máquina física. Ele também é conhecido como Monitor de Máquinas Virtuais (Virtual Machine Monitor - VMM). Como a máquina virtual representa uma visão lógica, ele precisa de um sistema operacional. Dessa forma, as VMs podem ter sistemas operacionais diferentes, desde que não haja conflito com a arquitetura computacional (por exemplo: x86, ARM) da máquina física. A Figura 2.4 mostra a diferença entre uma arquitetura tradicional e uma arquitetura baseada em VMs. Ações que ocorrem dentro de uma VM não afeta a máquina física.

A técnica de virtualização através de VMs é amplamente utilizada para soluções de micro-serviços [42]. Os autores em [61] abordam o assunto de virtualização para IoT. É possível criar e excluir VMs de acordo com a necessidade, diferente do caso de particionamento de disco rígido, onde não existe maleabilidade para modificações. Outra vantagem é o fato de poder fazer backup de VMs e restaurar em outras máquinas físicas.

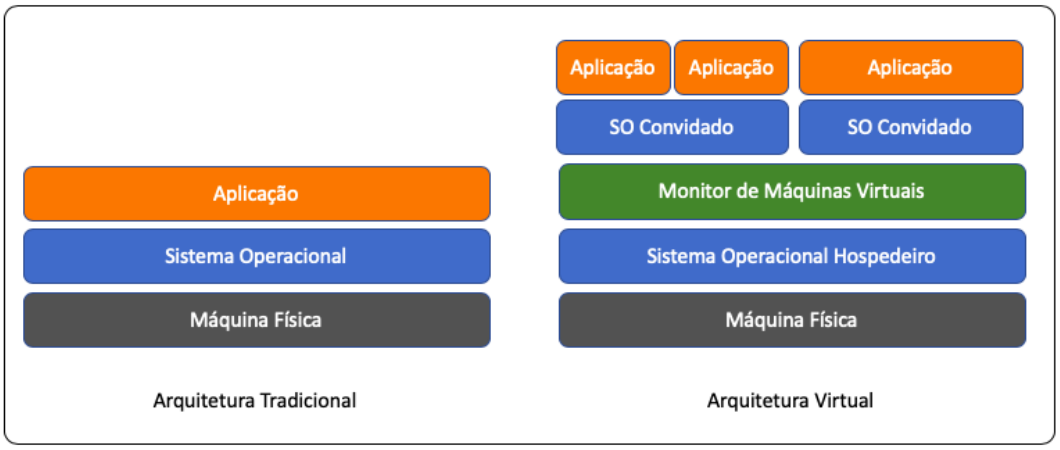


Figura 2.4: As diferenças entre uma arquitetura tradicional e uma arquitetura virtual.

#### 2.2.1.2 Contêineres

Os contêineres trabalham a nível de sistema operacional, e não de hardware. Cada contêiner compartilha o núcleo do sistema operacional em que se encontra hospedado, assim como seus binários e bibliotecas [15]. Do ponto de vista de desenvolvimento de software, cada contêiner funciona como um ambiente imutável para uma aplicação. Recursos computacionais podem ser divididos entre contêineres, porém, por padrão, eles são compartilhados, de forma que se permita que o contêiner mais acionado utilize mais recursos computacionais. Com relação a rede, cada contêiner que representa uma aplicação responde a uma porta pré-configurada no sistema operacional hospedeiro.

Sistemas de contêineres fazem uso de um Motor de Contêiner (Conteiner Engine - CE). Tal motor é um software onde é possível criar e gerenciar contêineres. Esse motor possui o básico necessário para executar qualquer contêiner sobe o sistema operacional hospedeiro. A Figura 2.5 mostra a diferença entre uma arquitetura tradicional e uma arquitetura baseada em contêineres.

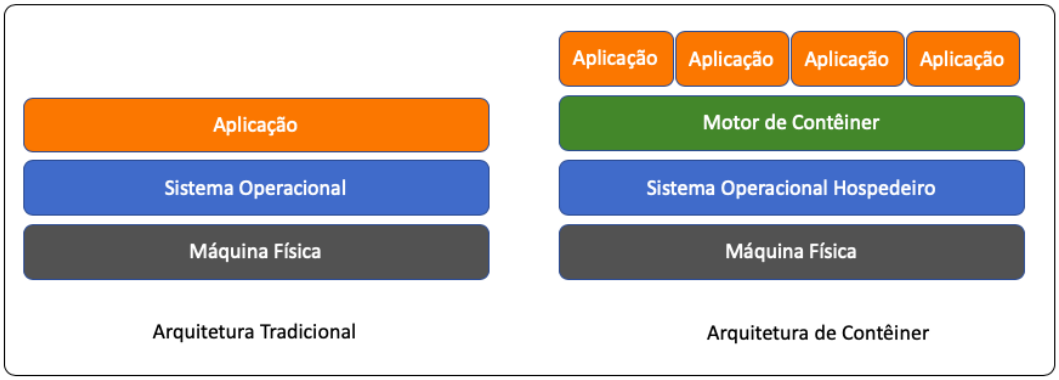


Figura 2.5: As composições físicas e lógicas entre uma arquitetura tradicional e uma arquitetura de contêiner.

Sistemas de contêineres é uma técnica de virtualização amplamente utilizada em soluções na nuvem e na neblina. No artigo [9] é apresentada uma solução de nuvem das coisas baseada em contêineres. Uma das vantagens de se trabalhar com contêineres é o fato que eles iniciam rapidamente e deixam a aplicação disponível em menos tempo que em VMs. Além disso também é possível fazer backups e restaurações de contêineres independente do sistema operacional que é executado no hospedeiro. Contêineres podem trabalhar dentro de VMs [3], o que possibilita uma grande redução de custos.

## 2.3 Aplicações ITS

ITS (Intelligent Transportation System) é uma área de estudo que se objetiva coletar dados das cidades, analisar esses dados, quando necessários integrar com outros sistemas da cidade e transformar em informações de forma que possibilite fornecer a população uma maior qualidade de vida em relação à transportes[39]. ITS aplica tecnologias de sensoriamento, controle, análise e comunicação aos transportes para melhorar a segurança, mobilidade e eficiência.

Existe uma gama de possibilidades onde aplicações ITS podem colaborar como facilitar o congestionamento de trânsito, melhorar a gestão do tráfego, minimizar o impacto ambiental e aumentar os benefícios do transporte tanto para empresas privadas como com objetivo público.

Paradigmas de EC podem ser grandes aliados de ITS por suas arquiteturas possuirem características que diminuam o tempo de resposta das aplicações, conseguindo assim atender uma maior demanda de requisições.

Especificamente de sensoriamento, o artigo [14] propõe um sistema de coleta de dados como o posicionamento geográfico durante o trajeto através de sensores em ônibus urbanos. O trabalho [64] apresenta o CFC-IoV, uma arquitetura cooperativa baseada em FC, que visa trabalhar com uma grande quantidade de dados da cidade para redes de veículos inteligentes. Já os autores em [36], apresentam a plataforma chamada FogFly, baseada em FC para resolver problemas de otimização de semáforos. Com relação a trajetórias, o VTracer apresenta métodos para compressão de trajetórias online e rastreamento de veículos [11]. No trabalho [16] apresenta uma proposta de compartilhamento de recursos de computação acessíveis usando redes multiacesso.

## 2.4 Séries Temporais

Uma série temporal é um conjunto de observações coletadas e ordenadas de acordo com o tempo de coleta [38]. O estudo do histórico de uma determinada variável pode ser utilizado para identificação de tendência de crescimento ou diminuição, sazonalidade e predição. Séries temporais são uma forma de avaliar o comportamento de uma variável ao longo do tempo. Exemplos de séries temporais:

- valores diários de veículos transitado por uma via;
- valores mensais de pessoas contagiadas por uma doença específica;
- índices diários da cotação de uma moeda;
- índices anuais de desmatamento de uma área geográfica.

Os modelos existentes para descrever séries temporais são processos estatísticos. Esses modelos utilizam o histórico de uma variável para projetar futuras observações. A Figura 2.6 mostra o exemplo de uma série temporal que representa idade de morte de 42 reis sucessivos da Inglaterra.

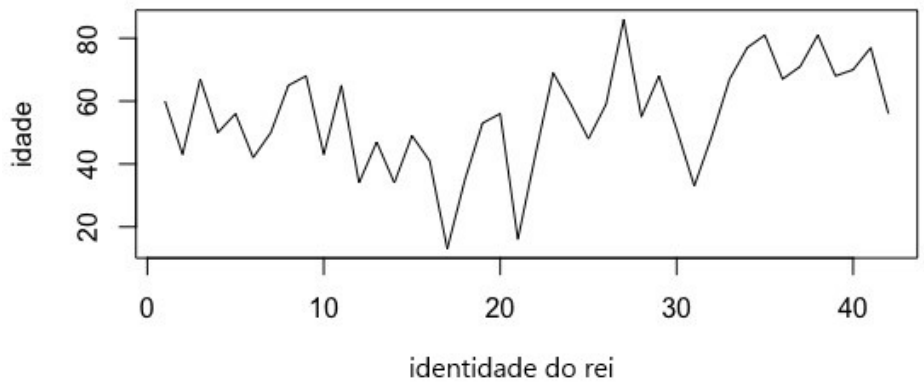


Figura 2.6: Uma série temporal representada graficamente

Para fazer análise de séries temporais deve-se levar algumas das suas principais características em consideração. A seguir, uma descrição das características de autocorrelação, tendência, sazonalidade e estacionariedade.

### 2.4.1 Autocorrelação

Existem modelos estocásticos no qual a modelagem de variáveis considera observações como independentes. Diferentemente, no caso de uma série temporal, é preciso considerar a dependência de suas observações. Levar em consideração a dependência seria partir do princípio que o valor de uma observação depende de uma ou mais observações obtidas anteriormente. Por exemplo, o índice de cotação de uma moeda no mês de Agosto pode estar relacionada com sua cotação no mês de Julho.

A autocorrelação é estabelecida na relação que em uma observação no tempo $t$ está relacionada com suas observações anteriores. É possível classificar a autocorrelação de acordo com sua ordem. A autocorreção de primeira ordem é quando em uma série, uma observação no tempo $t$ está correlacionada com a sua observação anterior, no caso $t-1$. Já uma autocorrelação de segunda ordem, a observação do tempo $t$ estaria correlacionada a observação nos tempos $t-1$ e $t-2$. E assim sucessivamente.

A identificação da autocorrelação é feita através da FAC (Função de Autocorrelação). A Figura 2.7 representa um exemplo de função de autocorrelação aplicada a série temporal apresentada na Figura 2.6.

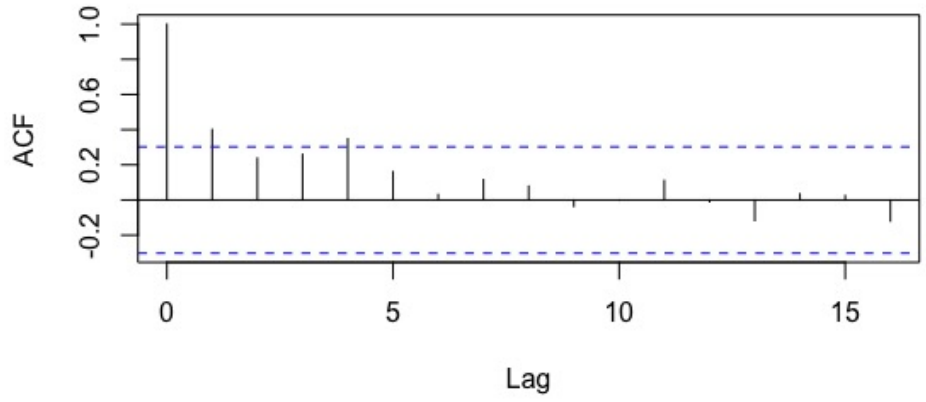


Figura 2.7: A aplicação de uma função de autocorrelação pela linguagem de programação *R*

No exemplo da Figura 2.7 a significância da ordem de autocorrelação é avaliada através dos intervalos de confiança (em azul). Dessa forma, esta série possui autocorrelação de primeira ordem, uma vez que o ponto no lag 1 é significativo. A autocorrelação no lag 0 é sempre igual a 1, por *default*.

A expressão de autocorrelação pode ser representada através da Equação 2.1 onde $r_t$

são os valores da série temporal.

$$p_k = \frac{Cov(r_t, r_{t-k})}{Var(r_t)} \tag{2.1}$$

### 2.4.2 Tendência

A tendência de uma série temporal é apresentada como um padrão de crescimento ou diminuição do valor da variável em uma janela de tempo. Existem testes específicos para a identificação da tendência, como o teste de sequencias (Wald-Wolfowitz), teste de sinal (Cox-Stuart) e o teste baseado no coeficiente de correlação de Spearman [38]. Regressão Linear Simples também pode ser utilizado para a identificação da inclinação da linha de tendência. Segue abaixo uma representação de uma série temporal com tendência de alta e outra com tendência de baixa. Na Figura 2.8, consideramos os dados referentes ao número de nascimentos por mês na cidade de Nova York. Pode-se observar uma clara tendência de alta na série ao longo dos anos.

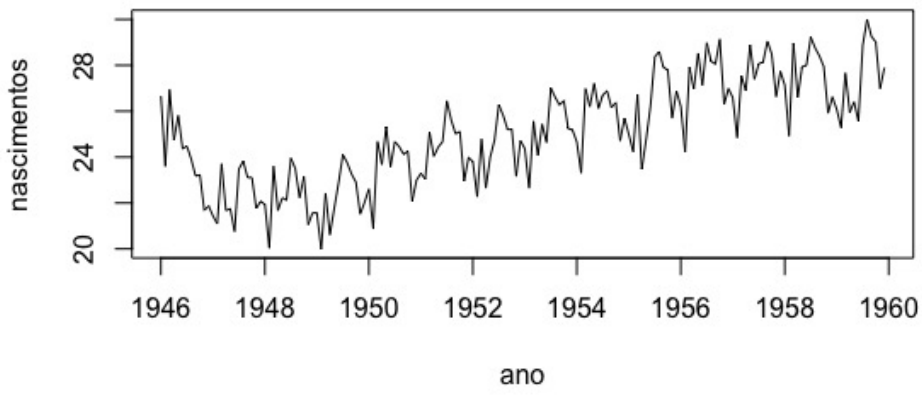


Figura 2.8: Representação de uma série temporal apresentando uma tendência de crescimento

### 2.4.3 Sazonalidade

A sazonalidade representa padrões de comportamento que se repetem em períodos idênticos de tempo, ou seja, diariamente, semanalmente, mensalmente, entre outros. A sazonalidade pode ser determinística onde apresenta um padrão de comportamento sazonal regular no tempo, o que facilita a previsão do comportamento sazonal a partir de dados históricos. A sazonalidade também pode ser estocástica quando o comportamento

sazonal da série varia com o tempo. É possível utilizar de alguns testes para identificação de sazonalidade dentro de uma série temporal, como o Teste de Kruskal-Wallis e o Teste de Friedman [38]. Na Figura 2.9 ilustra as vendas mensais da loja de souvenires em um resort de praia em Queensland, Austrália. Podemos observar nesta figura um comportamento similar que acontece a cada ano com um pico de vendas sempre no final do ano.

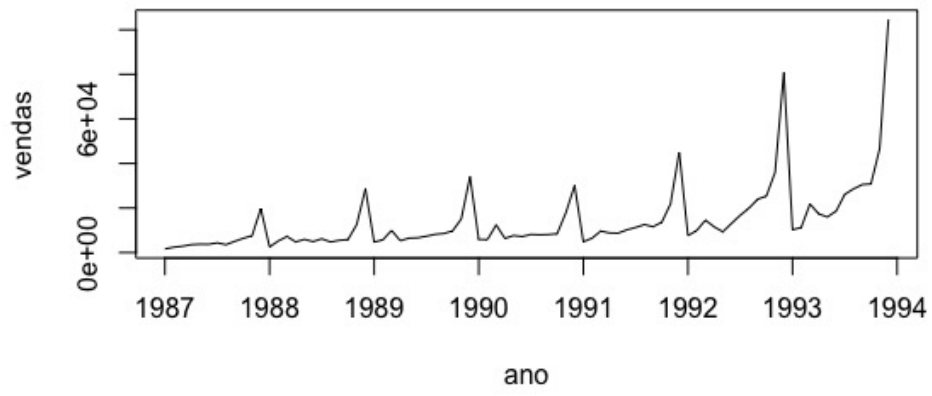


Figura 2.9: Representação de uma série temporal apresentando uma característica de sazonalidade

### 2.4.4 Estacionariedade

A estacionariedade é uma propriedade da série temporal onde a média, a variância e autocorrelação não mudam no decorrer do tempo [38]. A estacionariedade pode ter períodos longos ou curtos, e logo mudar de nível ou inclinação. Séries temporais que possuem tendência ou sazonalidade não são estacionárias e é necessário o uso de técnicas adequadas a tal situação. Na Figura 2.10, mostramos entrada de pessoas em um shopping. Podemos observar com o passar do tempo quantidade de pessoas entrando no shopping varia entre 3 e 7, mantendo sempre a mesma média e variância.

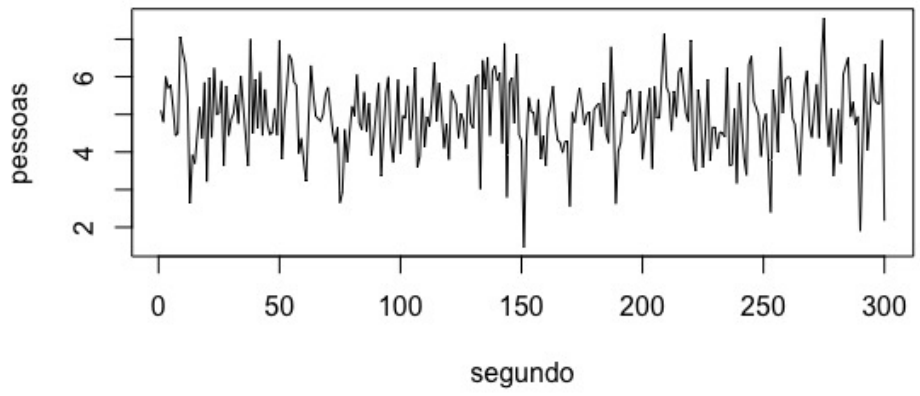


Figura 2.10: Uma série temporal demonstrando uma característica de estacionariedade

### 2.4.5 Modelos Auto-regressivos Estacionários

Modelos estacionários são aqueles em que sua média, a variância e autocorrelação não mudam no decorrer do tempo. São classificados em: Auto-regressivos, Médias Móveis e Auto-regressivo de Médias Móveis. Segue uma explicação dos modelos:

#### 2.4.5.1 Auto-Regressivo

O modelo Auto-regressivo (Auto-regressive - AR) especifica que a variável de saída depende linearmente de seus próprios valores anteriores e de um termo probabilístico. O modelo AR de ordem $p$ é definido como

$$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-i} + \varepsilon_t \tag{2.2}$$

onde $c$ é uma constante, $\varphi_i ... \varphi_p$ são os parâmetros e variável aleatória $\varepsilon_t$ é o ruído branco. Para que o modelo se mantenha estacionário é preciso considerar algumas premissas. Por exemplo, modelos AR(1) com $|\varphi_1| \geq 1$ não são estacionários.

#### 2.4.5.2 Médias Móveis

Médias Móveis (Moving Average - MA) são usadas em séries temporais para suavizar oscilações curtas e destacar tendências de longo prazo. A diferença entre curto e longo prazo depende da aplicação, bem como dos parâmetros da média móvel. O modelo genérico de MA de ordem $q$ é definido como

$$X_t = \mu + \varepsilon_t + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} \tag{2.3}$$

onde $\mu$ é o valor esperado de $X_t$, $\theta_i ... \theta_q$ são os parâmetros e $\varepsilon_t$, $\varepsilon_{t-i}$, ... são os termos de ruído branco.

#### 2.4.5.3 Auto-regressivo de Médias Móveis

O modelo Auto-regressivo de Médias Móveis (Auto-regressive Moving Average - ARMA) é a junção dos termos dos modelos AR e MA. É utilizado pra representar um processo não muito grande de parâmetros e que tenha os termos auto-regressivos e de médias móveis. O modelo ARMA de ordem $p, q$ é definido como

$$X_t = c + \varepsilon_t + \sum_{i=1}^{p} \varphi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} \tag{2.4}$$

Por ser uma junção dos modelos AR e MA a descrição da Equação 2.4 acompanha os termos já descritos nas Seções 2.4.5.1 e 2.4.5.2.

### 2.4.6 Modelos Auto-Regressivos Não-Estacionários

Muitos modelos de séries temporais não são estacionários, logo podem ter alterações em sua média, variância e autocorrelação com o passar do tempo. O principal modelo não-estacionário é o Médias Móveis Auto-regressivas Integradas que é descrito a seguir:

#### 2.4.6.1 Auto-regressivo de Médias Móveis Integradas

O principal modelo não-estacionário é o modelo Auto-regressivo de Médias Móveis Integradas (Auto-regressive Integrated Moving Average - ARIMA). Este modelo é uma generalização do ARMA visto na Seção 2.4.5.3. Ambos modelos são utilizados pra entender a série temporal ou realizar a predição. A parte auto-regressiva (AR) do ARIMA indica que a variável de interesse é regressada em seus próprios valores anteriores. A parte de média móvel (MA) indica que o erro de regressão é na verdade uma combinação linear dos ruídos brancos. A parte integrada (I) mostra que os valores fornecidos foram trocados a diferença entre estes valores e os valores históricos, e que este processo de troca pode ter ocorrido mais de uma vez. O objetivo deste processo de troca é que o modelo se ajuste da melhor forma possível. O modelo ARIMA de ordem $p, d, q$ é definido como

$$(1-\sum_{i=1}^{p}\varphi_i L^i)(1-L)^d X_t = (1+\sum_{i=1}^{q}\theta_i L^i)\varepsilon_t \tag{2.5}$$

Por ser uma generalização do modelo ARMA, a descrição da Equação 2.5 acompanha os termos já descritos na Seções 2.4.5.3. Para que a equação caracterize o ARIMA como não-estacionário é necessário que a variável $d$ seja maior que 0, pois quando $d = 0$, a expressão $(1-L)^d$ passa a ser 1. Logo se $d = 0$, não temos um modelo ARIMA(p, q, d) e sim um modelo ARMA(p, q). A variável $L$ é o operador defasagem. O operador de defasagem representa o número de períodos associados a uma observação anterior. O operador $L$ é linear. Logo para um valor $y_t$, temos $L^i y_t = y_t - i$, onde $L^i$ representa a defasagem de $y_t$ por $i$ períodos.

## 2.5 Redes Neurais

As redes neurais artificiais são métodos de previsão baseados em modelos matemáticos simples baseados nos neurônios. Uma rede neural pode ser considerada uma rede de neurônios organizados em camadas. A camada de entrada é formada pelos preditores e as camadas de saída são formados pelas previsões. Pode haver camadas intermediárias ou ocultas que são formadas pelos neurônios ocultos. A Figura 2.11 fornece um exemplo de uma rede neural com as três camadas citadas.

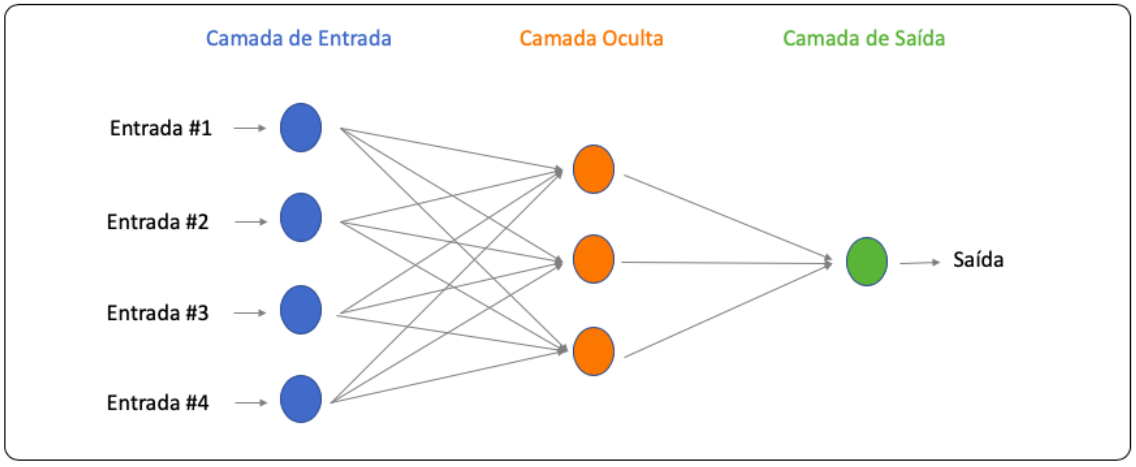


Figura 2.11: Diagrama de cadeia de Markov com 3 estados: Sem Chuva, Chuviscando e Tempestade

Diferentes tipos de redes neurais podem apresentar complexidades diferentes. A seguir uma descrição de alguns tipos de redes neurais.

### 2.5.1 Redes Multilayer Perceptrons

As Redes Multilayer Perceptron (MLP) é um algoritmo com uma ou mais camadas ocultas e um número indeterminado de neurônios [47]. A camada oculta possui esse nome porque não é possível prever a saída desejada nas camadas intermediárias. O MLP utiliza uma técnica de aprendizado supervisionado chamado retropropagação para o treinamento. A retropropagação é dividida em 4 etapas:

- inicialização: atribui valores aleatoriamente para pesos e limites, na qual sua escolha influencia o comportamento da rede;
- ativação: calcula os valores dos neurônios na camada oculta e na camada de saída;
- treino de pesos: calcula os erros dos neurônios nas camadas oculta e de saída e depois calcula correção dos pesos e atualiza os pesos;
- iteração: repete o processo de ativação até satisfazer o critério de erro.

### 2.5.2 Redes Neurais Convolucionais

As Redes Neurais Convolucionais são consideradas redes neurais profundas e podem ser aplicadas para classificar imagens, agrupar imagens por semelhança e reconhecer objetos dentro de imagens. Este tipo de algoritmo pode ser usado para identificar rostos, indivíduos, sinais de fogo em florestas, animais e uma série infindável de objetos visuais. Também podem ser utilizados junto ao OCR (Optical Character Recognition) para digitalizar textos assim como serem aplicados a arquivos de áudio, desde que representados visualmente por um espectrograma.

As redes convolucionais entendem imagens como objetos tridimensionais, ao invés de estruturas planas [41]. Essas redes utilizam a codificação RGB (Red-Green-Blue) para produzir o espectro de cores que os seres humanos percebem e recebem as imagens como três camadas separadas de cores empilhadas.

Redes neurais convolucionais não tem sua utilização restrita a imagens, porém tem se mostrado extremamente eficiente quando é preciso tratar dados extraídos de fontes multimídia.

### 2.5.3 Redes Neurais Recorrentes

As Redes Neurais Recorrentes são compostas por uma série de algoritmos de redes neurais específicos para o processamento de dados sequenciais, como som, dados de sé-

ries temporais ou linguagem natural. As redes recorrentes incluem um *loop de feedback*, pelo qual a saída da etapa $n-1$ é retroalimentada à rede de forma a interferir no resultado do passo $n$, repetindo o processo para cada etapa posterior [18].

As redes recorrentes geram modelos que mudam ao longo do tempo e produzem classificações precisas dependentes do contexto. Um modelo recorrente inclui o estado oculto que determinou a classificação anterior em uma série. Nas etapas posteriores, esse estado oculto é combinado com os dados de entrada da nova etapa e gera um novo estado oculto e uma nova classificação. Cada estado oculto é reciclado para produzir seu sucessor modificado.

### 2.5.4 Redes Neurais Auto-regressiva

Redes Neurais Auto-regressivas (Auto-regressive Neural Network - ARNN) utilizam os dados históricos da série temporal com dados de entrada para a rede neural. Denomina-se um modelo ARNN(p, k) uma rede neural auto-regressiva onde $p$ representa a entrada dos dados históricos e $k$ os nós da camada oculta [20].

O funcionamento de ARNN [31] pode ser matematicamente representado como

$$y_t = G(x_t : \psi) + \varepsilon_t = \alpha' \widetilde{X}_t + \sum_{i=1}^{h} \lambda_i F(\widetilde{\omega}_i' X_t - \beta_i) + \varepsilon_t \tag{2.6}$$

onde G($X_t$:$\psi$) é uma função não linear de variáveis $X_t$ onde os parâmetros $\psi \in \mathbb{R}^{(q+2)h+q+1}$ que são definidos como $\psi = [\alpha', \lambda_1, ..., \lambda_h, \widetilde{\omega}_1', ..., \widetilde{\omega}_h', \beta_i, ..., \beta_h]'$. As variáveis $\widetilde{X}_t \in \mathbb{R}^{q+1}$ e são definidas como $\widetilde{X}_t = [1, \widetilde{X}_t']'$, onde $\widetilde{X}_t \in \mathbb{R}^q$ são as lags de $y_t$ e suas variáveis exógenas. A função $F(\widetilde{\omega}_i' X_t - \beta_i)$ que é utilizada repetidamente pela função principal é a função logística

$$F(\widetilde{\omega}_i' X_t - \beta_i) = (1 + e^{-(\widetilde{\omega}_i' X_t - \beta_i)})^{-1} \tag{2.7}$$

onde $\widetilde{\omega}_i = [\widetilde{\omega}_{1i}, ..., \widetilde{\omega}_{qi}]' \in \mathbb{R}^q$, $\beta_i \in \mathbb{R}$, e a combinação linear dessas funções na Equação 2.6 formam a camada oculta. As Equações 2.6 e 2.7 não contém lags de $\varepsilon_t$ e por isso são chamadas de modelos de redes neurais *feedforward*. Além do mais, $\varepsilon_t$ é uma sequencia de variáveis aleatórias distribuídas de forma independente com média zero e variância $\sigma^2$.

## 2.6 Cadeia de Markov

Um processo estocástico é um modelo de probabilidade que descreve o progresso de um sistema de forma aleatória no decorrer do tempo [58]. Após a observação do sistema em um conjunto de momentos discretos, no final de um dia ou um mês, vamos ter um processo estocástico de tempo discreto. No entanto, se a observação for de forma contínua em todos os momentos, então teremos um processo estocástico de tempo contínuo [5].

Por trabalhar com dados que são obtidos no decorrer do tempo a Cadeia de Markov é uma técnica válida para trabalhar com séries temporais. No caso de séries discretas, usamos Cadeias de Markov de Tempo Discreto e no caso de séries temporais contínua utilizamos Cadeia de Markov de Tempo Contínuo. Neste documento, utilizaremos as siglas em inglês por ser um padrão adotado no meio acadêmico.

### 2.6.1 Cadeias de Markov de Tempo Discreto

Suponha que um sistema seja observado nos momentos $n = 0, 1, 2, 3, ....$ Seja $X_n$ o estado aleatório do sistema no tempo $n$. A sequência de variáveis aleatórias $\{X_0, X_1, X_2, ...\}$ é chamado de processo estocástico de tempo discreto e é escrito como $\{X_n, n \geq 0\}$. Seja $G$ o conjunto de valores que $X_n$ que podem assumir para qualquer n. Então $G$ é chamado de espaço de estados do processo estocástico $\{X_n, n \geq 0\}$.

Considerando Cadeias de Markov de Tempo Discreto (Discrete Time Markov Chain - DTMC) homogêneas com tempos finitos, introduzimos uma notação para a probabilidade de transição em uma etapa:

$$p_{i,j} = P(X_{n+1} = j | X_n = i), i, j = 1, 2, ..., N \tag{2.8}$$

Note que existe a transição de uma etapa $N^2$ probabilidades $p_{i,j}$. Podemos organizar em formas de matriz $NxN$ como na Equação 2.9:

$$P = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & \cdots & p_{1,N} \\ p_{2,1} & p_{2,2} & p_{2,3} & \cdots & p_{2,N} \\ p_{3,1} & p_{3,2} & p_{3,3} & \cdots & p_{3,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{N,1} & p_{N,2} & p_{N,3} & \cdots & p_{N,N} \end{bmatrix} \tag{2.9}$$

A matriz $P$ na Equação 2.9 é chamada de matriz de probabilidade de transição de uma etapa, ou simplesmente matriz de transição. As linhas correspondem ao estado inicial e as colunas correspondem ao estado final de uma transição. Assim, a probabilidade de ir do estado 2 para o estado 3 em uma etapa é armazenada na linha 2 e na coluna 3.

As informações sobre as probabilidades de transição também podem ser representadas de forma gráfica, construindo um diagrama de transição de estados. Um diagrama de transição de estados é um gráfico direcionado com $N$ nós, um nó para cada estado do DTMC. Existe um arco direcionado que vai do nó $i$ ao nó $j$ no gráfico se $p_{i,j}$ for positivo, neste caso, o valor de $p_{i,j}$ é escrito próximo ao arco. Podemos usar o diagrama de transição de estados como uma ferramenta para visualizar a dinâmica do DTMC. A Figura 2.12 ilustra o diagrama de transição. Para efeito de estudos, nesta figura são representadas três estados: Sem Chuva, Chuviscando e Tempestade. As setas represetam a transição de estados e os valores que as setas possuem representam a probabilidade de, dado que se encontra em um determinado estado, de ele passar para o estado seguinte.

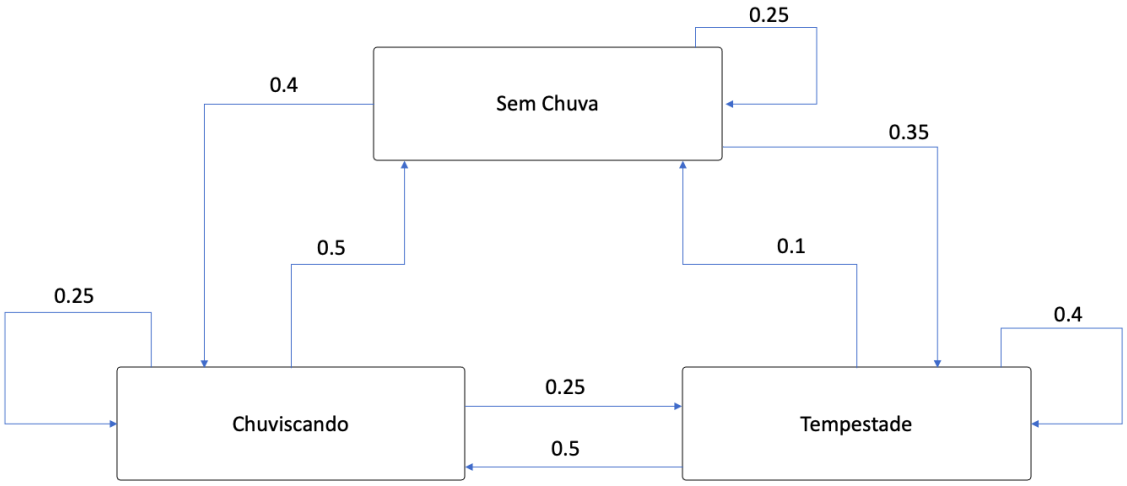


Figura 2.12: Representação de um diagrama de Cadeia de Markov

### 2.6.2 Cadeia de Markov de Tempo Contínuo

No caso de um sistema ser observado continuamente em todos os momentos $t \geq 0$, com $X(t)$ sendo o estado do sistema no tempo $t$. O conjunto de estados que o sistema pode estar em qualquer momento é chamado de espaço de estado e é denotado por $G$. O processo $\{X(t), t \geq 0\}$ é chamado de processo estocástico de tempo contínuo com espaço de estados $G$.

Considerando Cadeia de Markov de Tempo Contínuo (Contínuous Time Markov Chain - CTMC) homogêneas com tempos finitos, apresentamos uma notação para a probabilidade de transição:

$$p_{i,j}(t) = P(X(t) = j | X(0) = i), 1 \leq i, j \leq N \tag{2.10}$$

Os $N^2$ estados $p_{i,j}$ podem ser organizados em forma de matriz como na Equação 2.11:

$$P = \begin{bmatrix} p_{1,1}(t) & p_{1,2}(t) & p_{1,3}(t) & \dots & p_{1,N}(t) \\ p_{2,1}(t) & p_{2,2}(t) & p_{2,3}(t) & \dots & p_{2,N}(t) \\ p_{3,1}(t) & p_{3,2}(t) & p_{3,3}(t) & \dots & p_{3,N}(t) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{N,1}(t) & p_{N,2}(t) & p_{N,3}(t) & \dots & p_{N,N}(t) \end{bmatrix} \tag{2.11}$$

## 2.7 Considerações Finais

A partir daqui admite-se o conhecimento prévio sobre assuntos como EC, elasticidade de aplicações, ITS, séries temporais, redes neurais e cadeia de Markov.

Com esses conhecimentos, este trabalho segue na apresentação de suas propostas como a arquitetura ARTSIA e o algoritmo SABANN no Capítulo 3. Neste capítulo serão utilizados os conceitos de EC e FC para formação da arquitetura e implementação do experimento. Serão utilizadas máquinas virtuais com conteinerês e será aplicado técnicas para elasticidade de aplicações no uso do SABANN. Uma aplicação ITS será utilizada no experimento. Um dos cenários do experimento utilizará o modelo auto-regressivo de médias móveis, outro utilizará cadeia de Markov de tempo discreto, e um terceiro fará uso da redes neurais auto-regressivas.

Uma investigação usando algoritmos de aprendizado de máquina buscando melhorar a função de elasticidade de aplicações no Capítulo 4. Neste capítulo faremos uso de redes multilayer perceptrons.

Já no Capítulo 5 onde apresentamos o TSITS, uma arquitetura que utiliza MEC e seu experimento tem a implementação de sistemas de conteinerês. Realizaremos um estudo de caso com uma aplicação ITS.

Alguns assuntos neste capítulo foram inserido a preceito de conhecimento e posicionamento da literatura, porém não foram utilizados neste trabalho. São eles: virtualização com máquinas virtuais (VM), conceitos de séries temporais como autocorrelação, tendência, sazonalidade, estacionariedade. Também não foram utilizados modelos de séries temporais como o modelo auto-regressivo, médias móveis e auto-regressivo de médias

móveis interadas. Com relação a redes neurais não foram exploradas as redes neurais convolucionais e recorrentes. Por último, não utilizamos cadeias de markov de tempo contínuo.

# 3. ARTSIA: Elasticidade de Aplicações para Sistemas Inteligentes de Transportes em um Ambiente de Computação de Borda Móvel

O presente capítulo tem por objetivo apresentar a arquitetura ARTSIA (Architecture for Response Time-Sensitive ITS Applications) e o algoritmo SABANN (Scalability Algorithm Based on Neural Networks). Também foi realizado um experimento comparando o SABANN na arquitetura ARTSIA com outras técnicas do estado da arte visando diminuir a quantidade de requisições perdidas oriundas da camada terminal.

## 3.1 Visão Geral

Este capítulo aborda o cenário de aplicações ITS sensíveis ao tempo, aplicações que, precisam entregar respostas com o mínimo de tempo possível. Paradigmas de computação como Computação de Borda Móvel (MEC) corroboram para que a diminuição da entrega das mensagens em um menor tempo. Sendo assim o problema abordado será *como fazer a melhor utilização deste paradigma e como diminuir a quantidade de requisições perdidas através de elasticidade de aplicações*.

Os desafios em aberto (DA) e as principais contribuições (C) deste capítulo são:

1. (DA) Propostas de arquiteturas detalhadas de como aplicar os paradigmas de EC com aplicações ITS. (C) Será apresentado a arquitetura ARTSIA detalhando todos os componentes, suas atuações e suas integrações;
2. (DA) Como reduzir a quantidade de requisições perdidas de aplicações ITS sensíveis ao tempo através de elasticidade de aplicações. (C) Será introduzido o algoritmo SABANN que trabalha com previsão de séries temporais utilizando ARNN e limiares adaptativos de elasticidade;

3. (DA) Realização de experimentos utilizando componentes reais. (C) Será mostrado o resultado de um experimento utilizando o ARTSIA e comparando o desempenho do SABANN com o estado da arte.

Na Seção 3.2, é exibido os trabalhos relacionados de Computação de Borda que realizam elasticidade de aplicações. Dentre as contribuições mencionadas na Seção 1.5, o presente capítulo aborda na Seção 3.3, uma arquitetura detalhada para MEC nomeada de ARTSIA, assim como uma explicação de cada um dos seus componentes. Sobre como reduzir a quantidade de requisições perdidas em aplicações ITS sensíveis ao tempo a Seção 3.3 também descreve o funcionamento do algoritmo SABANN. O experimento e os resultados podem ser vistos na Seção 3.4, onde é detalhada a implementação, os cenários e as fases do experimento, assim como seus resultados. Por último, a Seção 3.5 revela uma discussão sobre alguns pontos cruciais do trabalho.

## 3.2 Trabalhos Relacionados

Em nossa pesquisa encontramos alguns trabalhos que tratam a questão de elasticidade em computação de borda. Cada trabalho, de acordo com o objetivo da aplicação trata a elasticidade utilizando métodos e políticas diferentes. Porém, dentro do nosso melhor conhecimento, o presente trabalho é o primeiro a tratar elasticidade em computação de borda para ITS.

O artigo [55] propõe computação de borda como uma alternativa à computação em nuvem para estender a elasticidade de plataformas IoT/M2M. Para atender os dispositivos M2M a plataforma oneM2M [2] possui dois tipos de nós: Infraestructure Node (IN) que é onde reside o servidor oneM2M e fica na nuvem e Middle Node (MN) que fica na borda. oneM2M possui várias interfaces de comunicação características de sua estrutura, porém ele também pode fazer um "binding" para HTTP e CoAP e, assim, permitir comunicação utilizando REST. A arquitetura de borda possui dois tipos de nós: Fog Manager que é responsável pelo processo de escalonamento nos Fog Worker, que são responsáveis por extrair a utilização de processamento e memória de cada Fog Worker e enviar ao Fog Manager. O Fog Worker também atua como o nó de serviço da aplicação recebendo as requisições dos sensores, refinando os dados e enviando para nuvem. Um Fog Worker atua como repositório das imagens de serviço. A virtualização nos MN foi feita utili-

[2] https://www.onem2m.org/

zando Docker[3] e foi utilizado HAProxy[4] como balanceador de carga. Docker Swarm[5] foi utilizado para trabalhar os Fog Workers como uma Fog Cluster. Foi estabelecido duas abordagens para os testes: estática onde cada Fog Worker possuía 3 instâncias de serviço e dinâmica onde as instâncias do Fog Worker começavam com 1 unidade e poderia escalar até 4 unidades, considerando o limite de *scale-up* como 80% de utilização do processamento. Para comparação, foram utilizados a média de consumo de energia em cada Fog Worker, a média de uso de processamento em cada Fog Worker e a média de tempo de resposta por requisição.

FRAS é uma plataforma de borda para IV (Integrated Virtualization) para atender aplicações industriais apresentada em [56]. A plataforma é composta por um orquestrador, pela rede docker, um gerenciamento de agentes e um balanceador de carga. Os componentes são baseados em contêiner como VNFs (Virtual Network Functions). Cada nó borda possui uma Docker Engine[6]. O docker engine recebe instruções do orquestrador sobre como gerenciar os contêineres, monitora os recursos do hospedeiro e envia para o orquestrador e gerencia a rede com todos os contêineres no nó. Com as informações recebidas pelo orquestrador, o mesmo decide como alocar os recursos na plataforma. A disponibilidade da plataforma é garantida utilizando um balanceador de carga que faz interface com o usuário através de uma VPN (Virtual Private Network), um sistema de monitoramento de saúde que realiza um *ping* via TCP ou HTTP para o VNF e um manipulador de eventos que reporta as informações para o Orquestrador. A proposta de elasticidade de recursos no FRAS é baseada em teoria fuzzy e leva em consideração a média de carga de processamento, média de utilização de memória e média de utilização de redes. A solução oferece uma escala de serviço com menor atraso médio e taxa de erro em comparação esquemas da Amazon AWS, DC/OS[7], EWMA e C-Scale.

Em [62] exibe uma proposta de arquitetura de borda para dispositivos IoT. A arquitetura da proposta segue a arquitetura hierárquica de computação de borda com três elementos: camada IoT, onde estão os sensores e acionadores; camada borda, onde estão os nós que realizam os cálculos estatísticos das informações dos sensores e realiza o monitoramento dos seus próprios recursos e; camada nuvem, que coordena e orquestra a distribuição de recursos na camada borda. Os nós de borda são virtualizados em contêineres utilizando docker e cada nó possui ao menos 3 instâncias: uma com Mosquitto

[3]https://www.docker.com/
[4]http://www.haproxy.org/
[5]https://docs.docker.com/engine/swarm/
[6]https://docs.docker.com/engine/
[7]https://dcos.io/

MQTT[8], uma com MongoDB[9] e a outra com a aplicação de negócio. O dispositivo físico dos nós de borda são RaspberryPi[10] e as imagens utilizadas pelo Docker são buscadas no serviço de registro do Docker Hub[11]. A proposta prevê que caso um nó de borda esteja próximo de não atender a demanda de requisições ele deve informar a nuvem para que ela decida se irá atender essa demanda ou irá repassar para outro nó na camada borda. Para isso, o autor propôs um monitoramento geométrico proativo considerando a utilização dos recursos de processamento, memória e disco. Para isso, o monitoramento utiliza uma "zona segura", onde os parâmetros de elasticidade são definidos e quando violados é preciso fazer um rebalanceamento no nó de borda. Essa "zona segura" é criada com base nos dados de monitoramento e os limites de elasticidade podem ser modificados de acordo com conhecimento adquirido pela "zona segura". Todos os nós de borda atuam como "worker node" no docker e o "master node" fica na nuvem.

FScaler é um agente de aprendizado por reforço desenvolvido pelos autores em [48] que trabalha com elasticidade horizontal baseada em contêineres em um ambiente de MEC. Esta solução foi modelada com Processos de Decisão de Markov (Markov Decision Processes - MDP) para resolvida com um algoritmo de Aprendizado por Reforço (Reinforcement Learning - RL). Para o gerenciamento de contêineres, a ferramenta Kubernetes[12] foi utilizada para suportar o algoritmo proposto chamado SARSA. Os recursos considerados para trabalhar elasticidade foram processamento, memória e disco rígido.

## 3.3 Arquitetura do ARTSIA

Para apresentar nossa proposta de elasticidade em computação de borda, especificamente para aplicações ITS criamos uma arquitetura especializada que segue a arquitetura de Computação de Borda Móvel sendo dividida em três camadas: nuvem, borda e terminal, sendo ilustrada na Figura 3.1.

A camada terminal é onde ficam os clientes de aplicações que consomem e fornecem informações as aplicações ITS. Essas interações com as aplicações ITS se dão através da comunicação de duas camadas: camada terminal e camada de borda. Na nossa proposta, consideramos que esses clientes em sua maioria serão os veículos.

A camada de borda é a principal camada da arquitetura. Nessa camada ocorre a maior

[8] https://mosquitto.org/
[9] https://www.mongodb.com/
[10] https://www.raspberrypi.org/
[11] https://hub.docker.com/
[12] https://kubernetes.io/

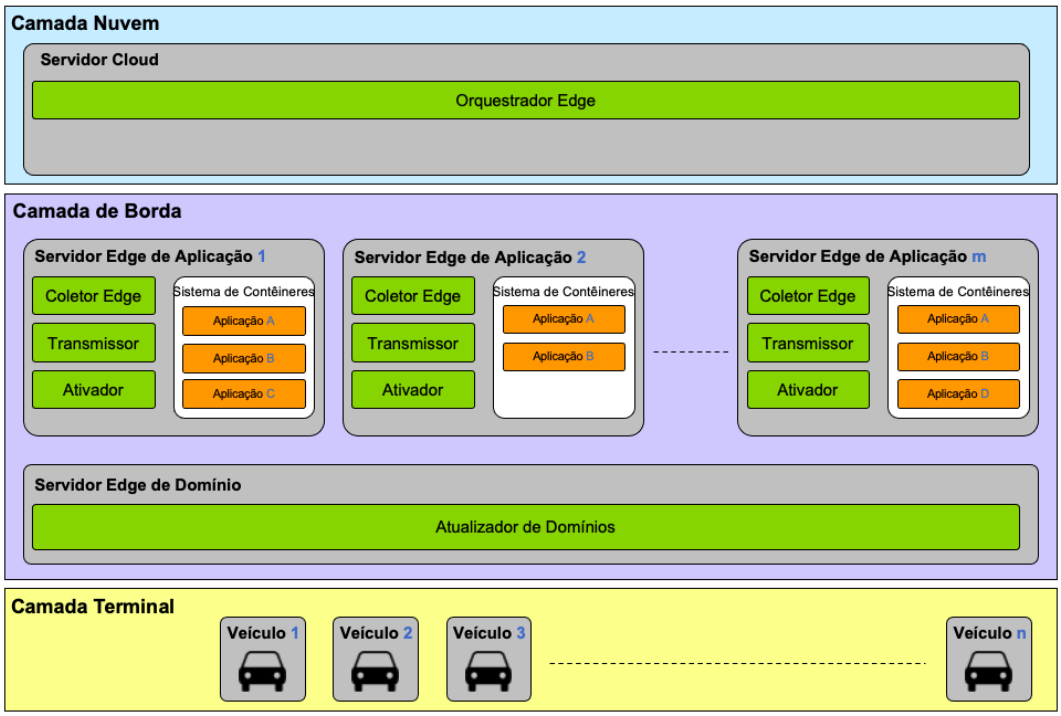


Figura 3.1: Arquitetura do ARTSIA separada em camadas e seus respectivos componentes

parte do processamento. Os veículos (clientes) se comunicam com as aplicações diretamente após serem endereçadas com o auxílio do Atualizador de Domínios. Ainda na camada de borda, existem os servidores de aplicação. Nesses servidores estão publicadas as aplicações ITS (representados na Figura 3.1 por Aplicações A, B, C e D). Cada servidor de borda possui um sistema de contêiner onde é possível fazer virtualização de várias aplicações ITS no mesmo servidor de borda. O sistema de contêiner permite que os recursos do servidor (processamento, memória, armazenamento, etc) sejam compartilhados entre as aplicações. Com esse compartilhamento, o sistema de contêiner automaticamente readéqua a utilização de recursos de cada aplicação de acordo com sua carga de trabalho. Porém, como esses recursos de um servidor de borda são limitados, a arquitetura prevê que haja $m$ servidores de borda, para que uma aplicação possa ser escalada entres os mesmos.

A arquitetura do ARTSIA divide o sistema de borda em 5 módulos: Atualizador de Domínios, Transmissor, Coletor Edge, Orquestrador Edge e Ativador. Cada módulo é independente e toda a interação entre os módulos é feita de forma assíncrona. Desta forma, se algum módulo ou servidor ficar sem funcionar por um tempo, o funcionamento do sistema não será interrompido. Todas as variáveis utilizadas tanto na arquitetura quanto no algoritmo proposto estão descritas na Tabela 3.1.

Tabela 3.1: Variáveis utilizadas na arquitetura e no algoritmo proposto.

| Variável | Descrição |
|---|---|
| *Tad* | intervalo de execução do Atualizador de Domínios |
| *Tesa* | tempo de inicialização de uma instância |
| *Tcf* | intervalo de execução no Coletor Edge |
| *Tcc* | intervalo de execução do Transmissor |
| *To* | intervalo de execução do Orquestrador Edge |
| *Ta* | intervalo de execução do Ativador |
| *Pa* | pontos de base do algoritmo |
| *Pp* | ponto de previsão do algoritmo |
| *Qtd* | quantidade de medidas utilizadas |
| *Tmd* | tempo mínimo de disponibilidade de uma instância |

### 3.3.1 Módulo Atualizador de Domínios

O módulo Atualizador de Domínios tem a responsabilidade de manter atualizado quais são os servidores disponíveis para o consumo de aplicações pela camada terminal. Cada aplicação ativa pode estar em um ou mais servidores. A inclusão ou exclusão de um servidor na lista do Atualizador de Domínios se dá exatamente pelo efeito da elasticidade de aplicações. Como observado na Figura 3.1, é exemplificado a Aplicação B nos Servidores Edge de Aplicação 1 e 2, logo o Atualizador de Domínio deve trabalhar para que haja apontamentos para a Aplicação B, tanto no Servidor Edge de Aplicação 1, quanto no Servidor Edge de Aplicação 2. Existe somente um Servidor Edge de Domínios, no qual está o módulo Atualizador de Domínios. A cada tempo que chamaremos de *Tad* é realizado uma coleta de informações do Servidor Cloud sobre quais servidores e aplicações estão disponíveis para consumo. O Servidor Edge de Domínio possui um servidor de DNS no qual o roteador consulta para o estabelecimento das rotas da rede. Após identificar que deve disponibilizar um domínio de aplicação para consumo, esse módulo aplica um tempo de espera *Tesa* antes de atualizar as rodas do servidor DNS e, consequentemente, disponibilizar a aplicação. Esse tempo se deve ao fato de que uma aplicação leva um tempo de carregamento para estar disponível. Se forem impostas requisições a ela antes desse tempo, as mesmas serão perdidas.

Alguns trabalhos, como em [55, 56], seguem o modelo de nuvem e utilizam balanceadores de carga. Essa solução tem se mostrado eficiente quando trabalhada com recursos como o uso de processamento e/ou memória como insumo para algoritmos de elastici-

dade. Como dito na introdução, o ARTSIA é para aplicações ITS onde o tempo de resposta precisa ser curto devido à importância da informação no momento. Esse requisito torna a vazão (throughput) um recurso principal para os cálculos de elasticidade. Logo, um balanceador de carga que recebe as requisições e as redistribui para os servidores de aplicação, nesse caso sofre um gargalo, tornando a solução ineficiente. Sendo assim, o módulo de Atualizador de Domínios torna-se uma boa opção, porque ele não recebe a carga de trabalho, mas mantém a rede sempre atualizada de quais aplicações estão disponíveis e onde estão.

### 3.3.2 Módulo Coletor Edge

O módulo Coletor Edge tem a responsabilidade de coletar medidas dos recursos físicos do Servidor Edge de Aplicação em que ele se encontra para que seja utilizado como insumo na tomada de decisão do Orquestrador Edge. Exemplos de medidas que podem ser coletadas são percentual de uso de processamento, percentual de uso memória, taxa de uso de disco rígido, etc. O Orquestrador Edge é que decide sobre as ações de elasticidade dentro do sistema de borda. Cada Servidor Edge de Aplicação possui um módulo do Coletor Edge.

Cada recurso é representado por *Ra*, *Rb*, *Rc*, até *Rz*. Em cada evento de coleta *EC*, são coletadas as medidas *MRa*, *MRb*, *MRc*, até *MRz*. A cada tempo *Tcf* ocorre um evento de coleta que é representado por *EC1*, *EC2*, *EC3*, até *ECn*.

### 3.3.3 Módulo Transmissor

O módulo Transmissor tem a responsabilidade de enviar as medidas de recursos coletadas pelo Coletor Edge para o Servidor Cloud. Cada Servidor de Aplicação Edge possui um módulo do Transmissor. Existe uma relação de confiança entre cada Servidor Edge de Aplicação e o Servidor Cloud. A cada tempo *Tcc* é realizado o envio com as medidas coletadas dos recursos para o Servidor Cloud.

### 3.3.4 Módulo Orquestrador Edge

O Orquestrador Edge tem a responsabilidade de tomar a decisão de qual Servidor Edge de Aplicação precisa habilitar/desabilitar a utilização de uma aplicação de negócio (*scale-up*/*scale-down*), baseado nas medidas coletadas nos Servidor Edge de Aplicação e previamente armazenadas no Servidor Cloud. Só existe um Servidor Cloud e dentro dele um módulo Orquestrador Edge. A cada tempo *To* o Orquestrador Edge realiza a leitura das

medidas dos Servidores Edge de Aplicação que já estão armazenadas no Servidor Cloud, pois foram enviadas por cada Servidor Edge de Aplicação através do módulo Transmissor. A seguir, ele aplica um algoritmo de elasticidade que com base nas medidas lidas, fornece instruções de elasticidade para cada Servidor Edge de Aplicação e armazena no próprio Servidor Cloud. Tais instruções são utilizadas por cada módulo Ativador.

Em [55], é apresentado um orquestrador de borda na camada de borda. Já em [62], [7] e [52] colocam esse módulo na camada da nuvem. No ARTSIA este módulo fica na nuvem por uma questão de manutenibilidade. A perda deste serviço no sistema, estando ele na nuvem, pode se dar por um problema no servidor ou uma quebra de enlace de internet. Caso uma dessas situações ocorra, pode ser corrigida rapidamente e de forma remota. Se este serviço estivesse na camada de borda e ocorre-se um problema no ligação de internet, precisaria do deslocamento de um técnico até o ponto de localização do servidor para seu restabelecimento.

### 3.3.5 Módulo Ativador

O módulo Ativador tem a responsabilidade de habilitar/desabilitar o uso de uma aplicação de negócio (*scale-up*/*scale-down*). Cada Servidor Edge de Aplicação possui um módulo Ativador. A cada instante *Ta*, o Ativador busca no Servidor Cloud quais instruções o Orquestrador Edge definiu, instruções quais o Servidor Edge de Aplicação em que o Ativador está presente, precisa fazer no momento com relação a elasticidade de aplicações. Partindo do princípio de que o ativador que consulta a nuvem e não o inverso contribui para independência da camada de borda. Essa independência é importante no contexto de ITS e é um diferencial em relação aos trabalhos [62] e [52].

### 3.3.6 Algoritmo SABANN

Com o objetivo de obter melhores resultados, que as técnicas utilizadas pelo estado da arte nós propomos o SABANN (Scalability Algorithm Based on Neural Networks), um algoritmo baseado em uma política proativa de elasticidade que faz uso da técnica de limiares adaptativos. Nesse algoritmo, utilizamos uma técnica de predição para séries temporais baseada em redes neurais autorregressivas (ARNN). Em [2] é apresentada uma revisão de algumas técnicas de séries temporais para trabalhar com elasticidade de recursos para computação na nuvem, porém não cita as redes neurais. Na pesquisa [53] foi feito uma revisão sistemática sobre metodologias de redes neurais artificiais para séries temporais. Nessa revisão, foi feito um critério de pontuação para as técnicas encontradas na pesquisa. Dentre as selecionadas, a ARNN foi uma das mais bem avaliadas. ARNN

é um modelo híbrido entre um modelo autoregressivo e um modelo de redes neurais com uma única camada oculta. Além disso, ARNN é utilizado para modelos simples e com baixo custo computacional. Para utilizar a ARNN é preciso informar a quantidade de medidas anteriores *Pa* como entrada e o tempo futuro *Pp*, que irá retornar o valor previsto pelo algoritmo como saída. Ainda assim, consideramos o fato de que servidores de borda podem ter configurações de hardware diferentes e estar sofrendo cargas de trabalho diferentes por estarem aptos a trabalhar com mais de uma aplicação. Por essa razão, aplicamos o que chamamos de limiares adaptativos. Limiares adaptativos analisam as últimas medidas *Qtd*, e com base nisso definem novos valores de limiares para *scale-up* e *scale-down*, a cada execução do algoritmo. Também foi considerado o tempo que mudou o estado da aplicação para disponível, o qual chamamos de *Tesa*. Por diversas vezes verificamos que logo após um *scale-down* os algoritmos realizaram um *scale-up*. Isso causava um consumo de processamento desnecessário, por vezes, permitindo perdas de requisições da camada terminal. Por essa razão, utilizamos um tempo mínimo em que uma instância precisa ficar disponível para que não impacte a execução, o que chamamos de *Tmd*.

O Algoritmo 1 representa os passos que foram utilizados pelo SABANN utilizando os procedimentos descrito nesta seção. Na linha 1 nós buscamos os servidores de borda que estão ativos, ou seja, com a aplicação ITS rodando. Na linha 2 nós buscamos os limiares adaptativos anteriormentes armazenados para comparação. Na linha 3 nós buscamos o último movimento realizado, que pode ter sido de *scale-up* ou *scale-down*. Nas linhas de 4 à 10 nós varremos o array de servidores, buscamos as medidas e calculamos os novos limiares de cada servidor. O cálculo de limiares se dá pela aplicação do 80º percentil nas medidas coletadas pelo recurso adotado no caso de *scale-up* e 20º percentil no caso de *scale-down*. Na linha 11 armazenamos os novos limiares para a futura execução do algoritmo. Na linha 12 traduzimos o ultimo movimento para uma *string*. Na linha 13 nós pegamos o tempo em que foi realizado o último movimento. As linhas 15 à 34 só são executadas se a linha 14 for satisfeita. Na linha 14 verificamos se a última ação do orquestrador foi de *scale-down* ou se foi de *scale-up* e o tempo da última ação é maior que a soma do tempo de inicialização da instância com o tempo mínimo de disponibilidade de uma instância, pois esse é o tempo mínima de realização entre dois *scale-up* para que o sistema não seja sobrecarregado. As linhas 15 e 16 admite que por padrão não haverá *scale-up* ou *scale-down*. Na linha 17 nós varremos novamente o array de servidores para executar as linhas 18 à 27 dentro do contexto de cada servidor. Na linha 18 nós buscamos as medidas armazenadas do servidor em contexto, na linha linha 19 nós pegamos o limiar de *scale-up* e na linha 20 o limiar de *scale-down* do mesmo servidor. Na linha 21 nós executamos o algortimo de ARNN e buscamos uma previsão de medida futura que será

usada em comparação com os novos limiares calculados. Nas linhas de 22 à 24 ele marca que deve haver uma ação de *scale-up* se a previsão de medida futura calculada pelo ARNN for maior que o último limiar de *scale-up*. Assim como nas linhas 25 à 27 é marcado para realizar um *scale-down* se a previsão de medida futura calculada pelo ARNN for menor que o último limiar de *scale-down*. Nas linhas 29 à 31 é verificado se o algoritmo marcou a realização de um *scale-up*, e em caso afirmativo, ele pega um servidor de borda que ainda não esteja rodando a aplicação ITS e instancia a aplicação nele. Já nas linhas 32 à 34 é verificado se o algoritmo marcou a realização de um *scale-down*, e em caso afirmativo, ele pega um servidor de borda que já está rodando a aplicação ITS e encerra esta instância.

## 3.4 Experimentos e Resultados

Nesta avaliação experimental buscamos validar as seguintes hipóteses:

H1 Quando aplicadas políticas de elasticidade proativas, a taxa de perda de requisições são menores que quando se utilizam políticas reativas.

H2 Políticas de elasticidade proativas apresentam uma variabilidade dos resultados obtidos durante o experimento menor que políticas reativas.

H3 O algoritmo SABANN deve apresentar a menor taxa de perda de requisição em comparação com as outras propostas.

A avaliação experimental foi dividida em duas fases: identificação de recursos e execução completa. Para emular os veículos da camada terminal foi utilizada uma ferramenta chamada Apache JMetter[13] na versão 5.3. Com essa ferramenta foi possível emular vários veículos emitindo uma grande quantidade de requisições. Para rodar essa ferramenta foi utilizado um computador na plataforma Windows com 1,7 GHz Intel Core i5 com 6 GB de memória RAM.

A Figura 3.2 mostra a carga de requisições gerada nos experimentos. A carga dura 30 minutos na forma de uma "pirâmide" de threads a cada 10 minutos. A geração de carga começa com 300 threads e tem seu pico em 7.300 threads. O objetivo da criação das "pirâmides" é emular uma carga onde pudessem ocorrer necessidades de *scale-up* e *scale-down* para de fato testarmos a elasticidade nos cenários avaliados. A formação deste plano de carga não tem nenhuma influência empírica de um plano de carga específico para

[13]https://jmeter.apache.org/

**Algorithm 1** Algoritmo SABANN

**Require:** $Qtd > 0 \vee Tesa > 0 \vee Tmd > 0 \vee Pa > 0 \vee Pa < Qtd \vee Pp > 0$

```
servidores ← lerServidoresAtivos()
limiares ← lerLimiares()
ultimoMovimento ← lerUltimoMovimento()
for all servidor ∈ servidores do
    medidas ← filtrarMedidas(servidor)
    if size(medidas) > Qtd then
        medidas ← filtrarUltimasMedidas(Qtd)
        limiares ← calcularLimiares(medidas)
    end if
end for
escreverLimiares(limiares)
acao ← pegarAcao(ultimoMovimento)
tempo ← pegarTempo(ultimoMovimento)
if isDown(acao) or (isUp(acao) and tempo > (Tesa + Tmd)) then
    scaleUp ← FALSE
    scaleDown ← FALSE
    for all servidor ∈ servidores do
        medidas ← filtrarMedidas(servidor)
        limUp ← pegarLimiarUp(servidor, limiares)
        limDown ← pegarLimiarDown(servidor, limiares)
        pontoFuturo ← ARNN(medidas, Pa, Pp)
        if pontoFuturo > limUp then
            scaleUp ← TRUE
        end if
        if pontoFuturo < limDown then
            scaleDown ← TRUE;
        end if
    end for
    if scaleUp then
        escalarServidorEdge()
    end if
    if scaleDown then
        retirarServidorEdge()
    end if
end if
```

aplicações ITS. Este plano foi gerado com o intuito de haver movimentos de *scale-up* e *scale-down*.

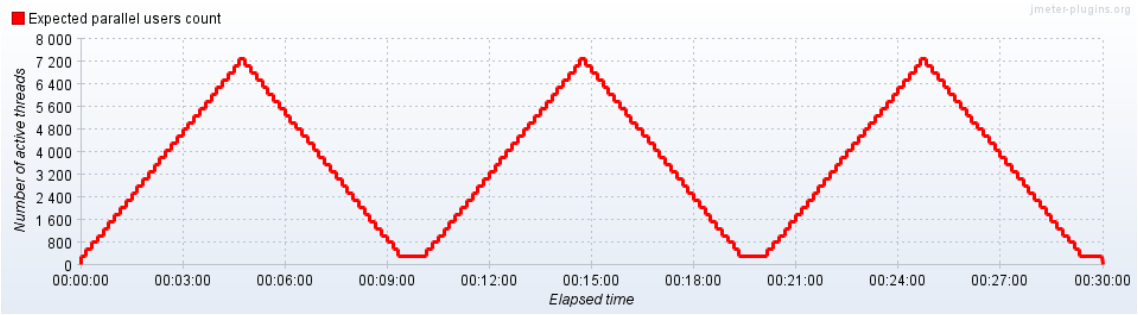


Figura 3.2: Carga de trabalho planejada no JMeter para emular as requisições dos veículos para o sistema de borda

A seguir demonstraremos a implementação da arquitetura ARTSIA que foi utilizada nos experimentos. Na Seção 3.4.2, foram também criados quatro cenários para comparar as técnicas já utilizadas em trabalhos anteriores e o algoritmo SABANN dentro da arquitetura ARTSIA.

### 3.4.1 Implementação

A comunicação entre a camada terminal e camada de borda se dá pelo protocolo HTTP. A aplicação desenvolvida para o experimento fornece serviços através do estilo arquitetural REST. A interação dos consumidores finais com a aplicação ITS se dá de forma síncrona. Já a comunicação entre a camada de borda e a camada da nuvem, que são interações entre componentes arquiteturais, utilizam o protocolo SSH (Secure Shell).

O Coletor Edge realiza a coleta de recursos e armazena em um arquivo CSV (Comma Separated Values). Os recursos utilizados foram vazão *Ra* e processamento *Rb*. Para cada evento de coleta *EC*, foi definido o *Tcf* de 5 segundos e são coletadas a vazão em Mbits/s *MRa* e o processamento em percentual de utilização *MRb*. O valor de 5 segundos foi escolhido após verificarmos que o tempo de coleta levou aproximadamente 5 segundos, sendo assim, em nossa implementação, é o tempo mínimo. O script de execução foi desenvolvido em Shell Script (SS) e ele executa uma aplicação de coleta desenvolvida em Java. Para a coleta de processamento, foi utilizada uma biblioteca chamada Java Sigar como no artigo [50]. Para coletar a vazão, foi utilizada a ferramenta linux Dstat[14].

Foi definido para o módulo Transmissor o *Tcc* de 10 segundos. O tempo de 10 segundos foi escolhido pois após testes, verificamos que esse valor garante que sempre tenha dados para ser enviado à nuvem. A cada *Tcc* o módulo pega os dados obtidos pelo Coletor

[14]https://linux.die.net/man/1/dstat

Edge previamente armazenado em CSV, envia esses dados ao Servidor Cloud, e limpa o arquivo CSV de forma que nunca se tenha dados repetidos. O script de execução foi desenvolvido em SS e o envio é feito utilizando uma ferramenta linux SCP (Secure Copy). A relação de confiança entre os servidores da borda e o servidor da nuvem é feita utilizando uma chave privada na comunicação entre eles.

No Orquestrador Edge foi definido um *To* de 5 segundos. O tempo de 5 segundos foi escolhido pois após testes, verificamos que este é o tempo adequado por garantir que sempre tenha dados novos para processamento. O script de execução foi desenvolvido em SS e o algoritmo foi feito em linguagem de programação R. Foi utilizado um algoritmo para cada cenário da Seção 3.4.2.

Para o Ativador foi definido um *Ta* de 10 segundos. O tempo de 10 segundos foi escolhido pois verificamos que este é o menor tempo que podemos utilizar e ainda assim encontrar mudanças de instrução em relação a execução anterior. Utilizando SS o Ativador busca as instruções da nuvem com SCP e utiliza uma aplicação Java que verifica a necessidade de realizar um *scale-up* ou *scale-down*.

Foi definido para o Atualizador de Domínio o *Tad* de 5 segundos. O atualizador de módulos é reflexo do resultado do Orquestrador Edge, sendo assim não faz sentido ter um tempo diferente do utilizado no Orquestrador Edge. Também foi configurado *Tesa* como 15 segundos. Já esse valor foi escolhido após várias observações da aplicação iniciando. Utilizando SS e uma aplicação Java que desenvolvemos ele busca as instruções do Orquestrador Edge com SCP e atualiza a lista de servidores disponíveis no Servidor DNS.

No Sevidor Edge de Domínio, foi configurado um servidor DNS que é consultado pelo roteador para resolver os domínios das requisições. O sistema de contêiner utilizado foi o Docker. Cada aplicação é um contêiner instanciado dentro do Docker Engine. Os servidores de borda são computadores Raspberry pi 3 com 1,2 GHz de 64-bit quad-core ARMv8, 1 GB de RAM e com sistema operacional Raspbian 10 instalado. O serviço da nuvem utilizado foi da AWS Services. O servidor da nuvem foi uma instância t2.micro com sistema operacional Ubuntu 18.04.4 LTS. A Figura 3.3 apresenta o desenho da infraestrutura utilizada no experimento.

No algoritmo SABANN, definimos *Qtd* como 120. Isso representa que utilizaremos os últimos 10 minutos do experimento como base para renovação de limiares. Definimos *Tesa* com 15 segundos, por que é o tempo médio que a nossa aplicação leva para ficar disponível. Já o tempo mínimo de disponibilidade após iniciar *Tmd*, colocamos 30 segundos.

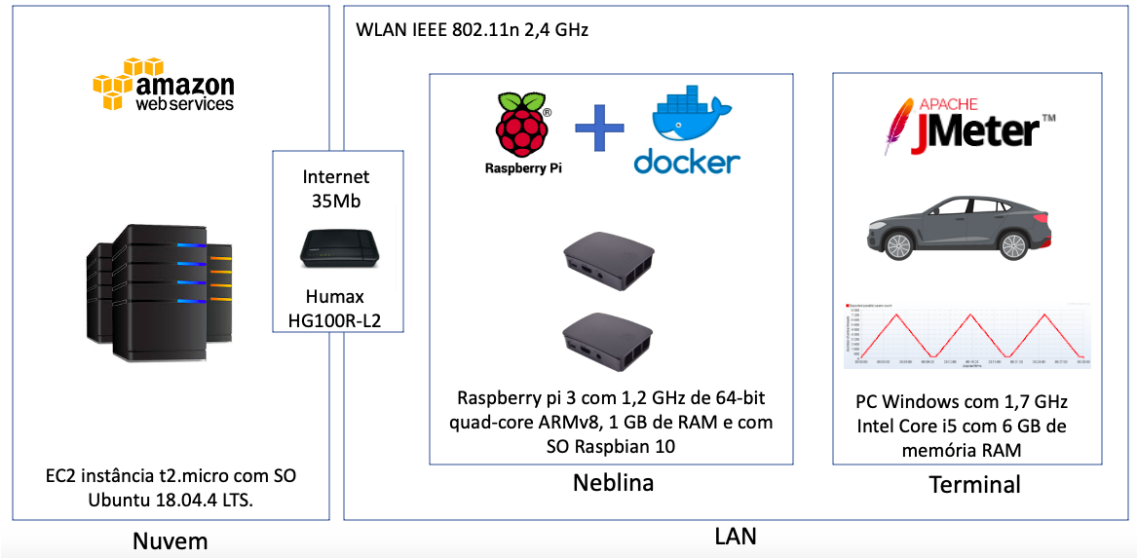


Figura 3.3: Desenho da infraestrutura utilizada no experimento com os componentes utilizados na nuvem e dentro da LAN.

Por último, com o objetivo de capturar uma tendência, seja de alta ou de baixa, definimos *Pa*, que são o pontos históricos utilizados como base para o algoritmo e *Pp*, que é o tempo de previsão para qual o algoritmo deve trabalhar, como 12 e 3, respectivamente. *Pa* possui o valor de 12 pois visa pegar as medidas dos últimos 60s e *Pp* como 3 por que buscamos o ponto futuro de 15s a frente.

### 3.4.2 Cenários

Para testar arquitetura foram propostos alguns cenários baseados em abordagens de trabalhos já existentes na literatura. Foi implementado e utilizado um algoritmo de elasticidade para cada cenário, que será executado pelo Orquestrador Edge. Assim, serão adotados quatro cenários e que serão descritos a seguir.

O primeiro cenário é adotar uma abordagem reativa baseada em um recurso computacional. Essa abordagem é uma abordagem comumente já utilizada em computação na nuvem e foi adotada por [52, 55]. O recurso adotado será a vazão *Ra* e chamaremos este cenário de Clássico.

Semelhante ao primeiro, segundo cenário também é adotar uma abordagem reativa, porém com uma combinação de recursos utilizando lógica fuzzy como feito por [56]. Para este cenário, utilizamos a vazão *Ra* e processamento *Rb* e chamaremos este cenário de Fuzzy.

O terceiro cenário irá adotar uma abordagem proativa utilizando um modelo de série

temporal como utilizado em [62]. Para esse algoritmo utilizamos ARMA, uma vez que já foi utilizado também em soluções na nuvem como em [26]. Chamaremos este cenário de ARMA.

No quarto cenário, usaremos outra abordagem proativa usando um modelo estocástico. Os modelos estocásticos foram categorizados no estudo de elasticidade por [2]. Em seu trabalho sobre elasticidade baseada em nuvem, [26] usa Cadeia de Markov de Tempo Contínuo (Continuous-Time Markov Chain - CTMC). Em um ambiente de borda, [48] apresenta uma solução com Processo de Decisão Markov (Markov Decision Process - MDP) e Aprendizagem por Reforço (Reinforcement Learning - RL). Nosso cenário foi feito usando Cadeia de Markov de Tempo Contínuo (Discrete-Time Markov Chain - DTMC) semelhante ao uso em [60]. Chamaremos esse cenário de Cadeia de Markov.

O último cenário irá utilizar o algoritmo proposto, que também faz uso de políticas proativas. Logo, chamaremos este cenário de SABANN.

### 3.4.3 Fase 1: Identificação de Recursos

A primeira fase consistiu em identificar quais recursos computacionais eram utilizados pela aplicação de forma que justificasse seu uso em um algoritmo de elasticidade de aplicações. Dentro do plano de carga de requisições já citado foram observados alguns recursos que tiveram suas medidas coletadas de um servidor de borda. Dentre os recursos observados os que tiveram significativa variação foram processamento e vazão.

Como visto da Figura 3.4, existe uma variação do processamento ao longo do tempo, porém observamos após dezenas de rodadas que sua média gira em torno de 31% de uso de processamento. Fato esse que confirma que a característica da aplicação não é uso excessivo de processamento.

A Figura 3.5 mostra que a vazão tem uma leve tendência de alta nos picos de carga, e algumas vezes elas também atingem o pico nesses momentos. Por vezes, observamos que a taxa de vazão obtida pelo Coletor Edge ficava em zero quando começava a ocorrer perda de requisições, sendo um comportamento similar ao travamento do acesso da rede ao servidor de borda. Também observamos que após dezenas de rodadas sua média ficava por volta de 5,5 Mbits/s.

Depois de vários testes definimos os limiares de elasticidade para a fase de execução completa. Com relação a vazão, para *scale-up* definimos 6,6 Mbits/s e para *scale-down* 4,4 Mbits/s. Especificamente no Cenário Fuzzy, onde também utilizamos processamento, definimos 35% como limiar de *scale-up* e 20% para *scale-down*. Munido dessas informa-

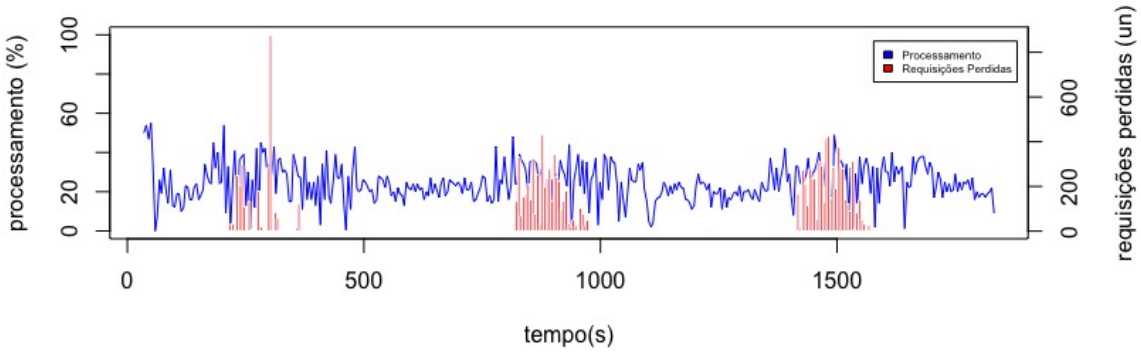


Figura 3.4: Comportamento do processamento do servidor de borda durante a execução do plano de carga de requisições em uma rodada.

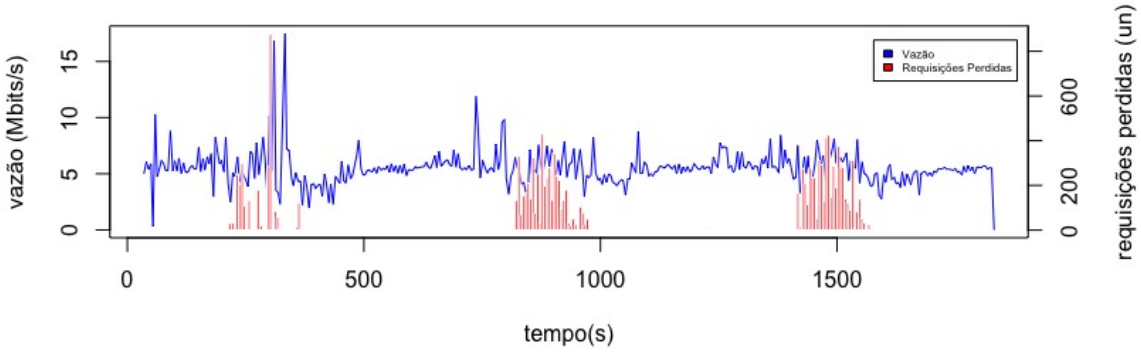


Figura 3.5: Exibição do desempenho da vazão no servidor de borda durante a execução do plano de carga de requisições planejado.

ções passamos a segunda fase que foi a execução completa do experimento.

Para o cenário da Cadeia de Markov, consideramos os dados históricos como uma série temporal. Dentro desta série, encontramos o ponto máximo do valor da taxa de transferência e os usamos como base para dividir as medições em áreas iguais e considerar cada uma como um estado na Cadeia de Markov. Criamos nossa matriz de estados com 10 estados. De acordo com nossos testes, esse valor foi escolhido porque cobre os valores de taxa de transferência em cada estado que deu mais significado para considerar uma mudança de estado. Com base nessa matriz de estado, criamos nossa matriz de transição que foi usada como referência para previsões de mudança de tendência. Para criação da Matriz de transição foram considerados os dados históricos, a origem e destino de cada transição, e foi calculado a propabilidade entre as transições. Em nossos testes, o estado 4 em média cobre uma faixa de $4,3$ a $5,6$ Mbits/s. Consideramos o limite de aumento quando o estado previsto era maior que 4 e o limite de aumento quando era menor que 4. Munidos dessas informações, passamos à segunda fase, que foi a execução completa do experimento.

### 3.4.4 Fase 2: Execução Completa

Esta fase se concentra na execução completa do experimento envolvendo os servidores de borda e o servidor da nuvem. Dentro dos cenários apresentados na Seção 3.4.2 foram realizadas dez rodadas, a média e o nível de confiança de 95% cada cenário.

A principal métrica de desempenho utilizada para a comparação dos cenários foi a taxa de requisições perdidas diante da carga de requisições geradas.

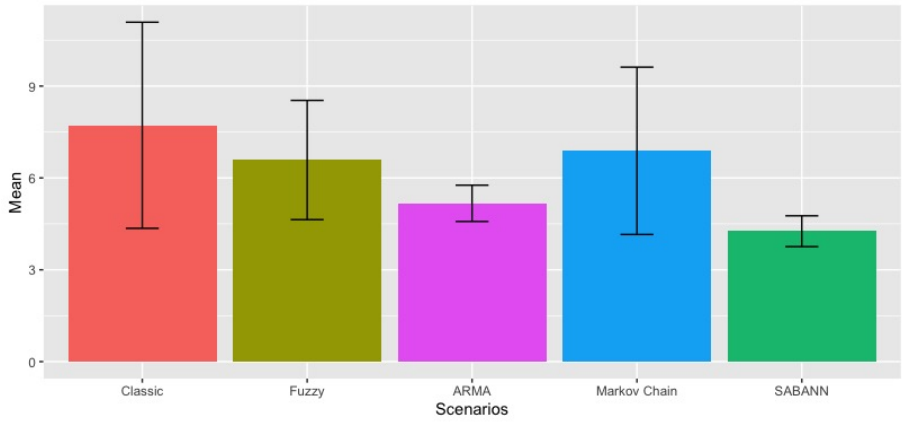


Figura 3.6: Percentual médio de requisições perdidas no experimento separada por cada cenário realizado.

A Figura 3.6 apresenta uma análise a respeito do percentual médio de requisições

perdidas em cada um dos cenários. Podemos observar que o cenário Clássico é o cenário onde existe a maior variação de taxa de requisições perdidas por rodada com um desvio-padrão de 3,36. Durante as rodadas desse cenário, observamos que ações de *scale-up* e *scale-down* foram executadas com tempo muito próximo uma da outra por diversas vezes, fazendo com que o benefício de escalar outra aplicação nem sempre fosse utilizado.

Durante as rodadas do cenário Fuzzy observamos que o fato de termos a combinação de duas variáveis como ponto a ser alcançado para a tomada de decisão de qual ação de elasticidade tomar, algumas vezes fez com que a quantidade de requisições perdidas se aumenta por não terem alcançado seu ponto de *scale-up*. Em outros momentos, ficou-se mais tempo que o necessário com mais de uma instância da aplicação ativa sem necessidade por não alcançarem o ponto de *scale-down*.

No cenário ARMA, no nosso experimento observamos que, com a carga de requisições projetada para que pudéssemos ter ações tanto de *scale-up* quanto de *scale-down*, os pontos futuros para tomada de decisão giravam em torno da média geral da rodada. Observamos também que por adotar uma política proativa, esse cenário por vezes conseguiu prever o momento de escalar uma nova instância da aplicação, fazendo com que a taxa de requisições perdidas obtida ao longo de 10 rodadas fosse melhor que os cenários Clássico e Fuzzy.

Com uma abordagem estocástica, o cenário da Cadeia de Markov com média de $6,88\%$ não apresentou resultados melhores que o Fuzzy com média de $6,58\%$, embora seja uma abordagem proativa. Levando em consideração o desvio padrão de $2,73$, a Cadeia de Markov não teve variação maior que o cenário Clássico. Em vários momentos, o algoritmo permaneceu no mesmo estado por um longo tempo (faixa de valores de transferência) e não previu os movimentos de aumento de escala.

A característica de adaptabilidade dos limiares por vezes fez com que os pontos de elasticidade do cenário SABANN sofressem alterações. Isso fez com que as ações de *scale-up* e *scale-down* acompanhassem a taxa de vazão que era utilizada nos servidores de borda. O algoritmo SABANN também possui uma abordagem proativa, tentando prever o próximo ponto de elasticidade da plataforma. Observamos que a combinação da abordagem proativa com os limiares adaptativos trouxe uma maior precisão do momento de escalar novas instâncias permitindo que a plataforma se antecipasse ao ponto de saturamento do servidor onde há perdas de requisições. Assim, o algoritmo SABANN se mostrou melhor que o cenário ARMA apresentando uma taxa média de requisições perdidas de 4,25% contra 5,17% do ARMA.

## 3.5 Discussão

Com relação à hipótese de pesquisa *H1*, observamos que os cenários ARMA e SABANN, que usaram uma política proativa, tiveram médias de $5,17\%$ e $4,25\%$, respectivamente, enquanto os cenários Clássico e Fuzzy, que usaram uma política reativa tiveram médias de $7,72\%$ e $6,58\%$, respectivamente. A exceção foi o cenário da Cadeia de Markov, que teve resultados inferiores comparados ao cenário Fuzzy, mas ainda melhores do que o cenário Clássico. Mesmo assim, em relação ao *H2*, os cenários ARMA e SABANN tiveram muito pouca variabilidade, ambos com desvios-padrão abaixo de $0,6$ enquanto Classic e Fuzzy os desvios estão acima de $1,9$. Novamente, a Cadeia de Markov foi uma exceção, pois também mostrou um desvio padrão acima de $1,9$. Por fim, constatamos que o algoritmo SABANN teve o melhor resultado na proposta de diminuir o número de requisições perdidas da camada terminal, reforçando nossa crença de que *H3* é verdadeiro.

Embora tenha-se conseguido reduzir o número de requisições perdidas utilizando a arquitetura ARTSIA e o algoritmo SABANN, existe alguns pontos nos quais estão abertos a discussões.

Nos trabalhos relacionados, podem ser observados que os autores utilizaram da medição de recursos computacionais diferentes para realizar a elasticidade de aplicações. Dentre tais recursos foram utilizados processamento, memória, vazão, entrada/saída de dados e disco rígido. Como já explicado no trabalho, a escolha deste projeto para utilização de vazão como recurso computacional para realizar elasticidade se dá pelas características da aplicação. Por essa razão, observamos que um balanceador de carga que recebe as requisições e redistribui para os servidores de aplicação não era eficiente, pois o mesmo se tornava um gargalo de requisições e deixava a aplicação indisponível. Esse problema foi resolvido com o Atualizador de Domínios, que junto ao roteador e um serviço de DNS, endereçava as requisições diretamente para as aplicações.

O módulo Coletor Edge realiza a coleta de medidas de cada servidor de borda. Essa coleta é feita utilizando as bibliotecas Java Sigar e a ferramenta linux Dstat, e leva aproximadamente 5 segundos para realizar cada coleta. A utilização de outras ferramentas poderia ser testadas de forma a diminuir este tempo.

O módulo Orquestrador Edge estabelece instruções de elasticidade de aplicações para camada de borda. Atualmente, ele realiza a leitura das medições dos servidores de borda, realiza o cálculo de predição e estabelece as novas instruções para a camada de borda. Os algoritmos de ML utilizados para realizar a predição, podem necessitar de mais dados e levam mais tempo no seu funcionamento. Logo, pode se tornar interessante dividir a fun-

ção do Orquestrador Edge em duas etapas: (1) offline, realizando a leitura das medições, o cálculo e armazenamento da predição e (2) online, realizando a leitura das predições e estabelecendo instruções. Desta forma, o tempo de cálculo não atrapalharia a definição de instruções e o Orquestrador Edge não viraria um gargalo.

Na construção do algoritmo SABANN utilizamos o ARNN por ser o algoritmo com maior pontuação na pesquisa [53]. Fizemos testes comparatórios entre ARMA e ARNN recebendo a mesma carga de requisições e verificamos que o ARNN tem um desempenho ligeiramente melhor. Porém é válido um estudo para testar outros tipos de algoritmo dentro do SABANN para verificar se é possível obter um desempenho melhor que o ARNN.

A elasticidade apresentada no presente trabalho se diz respeito a elasticidade horizontal. A elasticidade vertical não foi considerada pois em nosso trabalho utilizamos todo o potencial dos recursos da infraestrutura. Caso haja um experimento com mais de uma aplicação, a utilização de elasticidade vertical pode ser interessante, limitando os recursos da infra estrutura por aplicação.

Segundo o autor em [13], o Instituto Europeu de Padrões de Telecomunicações (ESTI) passou a utilizar a abreviatura MEC como Computação de Borda Multiacesso, ou, ESTI MEC. Comparando o ARTSIA com a arquitetura de referência do ESTI MEC podemos observar igualdades e diferenças. O ARTSIA segue os mesmos conceitos do ESTI MEC por utilizar uma arquitetura virtualizada, gerenciamento de rotas com DNS, orquestração de instâncias. Porém o ESTI MEC apresenta sua proposta componentes que não utilizamos, como por exemplo um controle de permissão de acesso, serviço de registro de aplicações e controle de regras de tráfego. ESTI MEC também foi construído para trabalhar com Network Functions Virtualisation (NFV).

Com relação a infraestrutura utilizada no experimento, na camada de borda, foram utilizados dois servidores para aplicações e um servidor de domínio. Embora tenha-se conseguido atingir o objetivo de diminuir a quantidade de requisições perdidas utilizando de elasticidade de aplicações, não se pode admitir estes resultados para todos os cenários. Um bom estudo que poderia ser feito seria utilizar dezenas de servidores de aplicações para verificar se existe a manutenção do comportamento.

# 4. Uma Investigação sobre Elasticidade de Aplicações em Computação de Borda Utilizando Aprendizado de Máquina

Neste capítulo, será realizado uma investigação sobre elasticidade de aplicações em EC utilizando algoritmos de ML.

## 4.1 Visão Geral

Baseado no conjunto de dados que foi gerado na Seção 3.4, será verificado a possibilidade de utilizar algoritmos de ML, de forma que possa diminuir a quantidade de requisições perdidas para aplicações ITS sensíveis ao tempo em uma abordagem de EC. Logo, o problema abordado será *como diminuir a quantidade de requisições perdidas através de elasticidade de aplicações*.

Os desafios em aberto (DA) e as principais contribuições (C) deste capítulo são:

1. (DA) Como reduzir a quantidade de requisições perdidas de aplicações ITS sensíveis ao tempo através de elasticidade de aplicações. (C) Será realizado um estudo utilizando ML sobre um conjunto de dados gerado a partir dos experimentos utilizando componentes reais visando melhor atender a elasticidade de aplicações em EC.

Será realizado uma análise exploratória do conjunto de dados gerado pela avaliação experimental feita na Seção 3.4. Visando cobrir a contribuição citada na Seção 1.5 sobre reduzir a quantidade de requisições perdidas em aplicações ITS sensíveis ao tempo, este capítulo aplicará algoritmos de ML visando encontra a melhor opção para reduzir requisições perdidas em aplicações ITS baseado no conjunto de dados do estudo. Esse estudo também pretende responder as seguintes questões de pesquisa:

QP1: É possível diminuir a quantidade de requisições perdidas utilizando algoritmo de ML?

QP2: Dentre os algoritmos de ML aplicados nesta pesquisa, qual é mais eficiente em diminuir a quantidade de requisições perdidas consumindo o mínimo de recursos computacionais?

## 4.2 Análise Exploratória dos Dados

Após diversas rodadas do experimento na Seção 3.4 foi gerado um conjunto de dados com os dados coletados da camada de borda e tratados para análise. O dicionário de dados do conjunto de dados pode ser visto na Tabela 4.1.

Tabela 4.1: Dicionário de dados.

| **Campo** | **Descrição** | **Un. Medida** |
|---|---|---|
| time | momento de coleta do dados | unidade |
| cpu | uso de processamento | % |
| mem | uso de memória | % |
| io | entrada/saída de dados | KB |
| net | vazão | Mbits/s |
| lost | indicador se houve (1) ou não (0) requisições perdidas | boleano |

O campo *time* se refere a cada momento de coleta que ocorreu no decorrer do tempo, que representa aproximadamente 5s. Os campos *cpu* e *mem* representam o percentual de uso de processamento e memória, respectivamente, coletados do servidor da camada de borda. O campo *io* contém a quantidade de entrada/saída de dados em KB dentro servidor da camada de borda. Já o campo *net* condiz com a vazão registrada pelo servidor de borda no momento da coleta. Por último, após vários testes realizados antes do experimento e utilizado como base para tomada de decisão de escalar uma nova instância, o campo *lost* é um indicador que determina se foi necessário escalar uma nova instância de aplicação ou não naquele momento de registro.

Buscando entender os dados foi utilizado o diagrama de pares padrão da Figura 4.1. Foi notado aqui que o único par de dados que apresentam algum tipo de correlação são *cpu* e *net*, e mesmo assim não foi possível ver nenhum tipo correlação entre elas e a variável independente *lost*.

Ao utilizar o mapa para representar a matriz de correlação na Figura 4.2, foi visto com

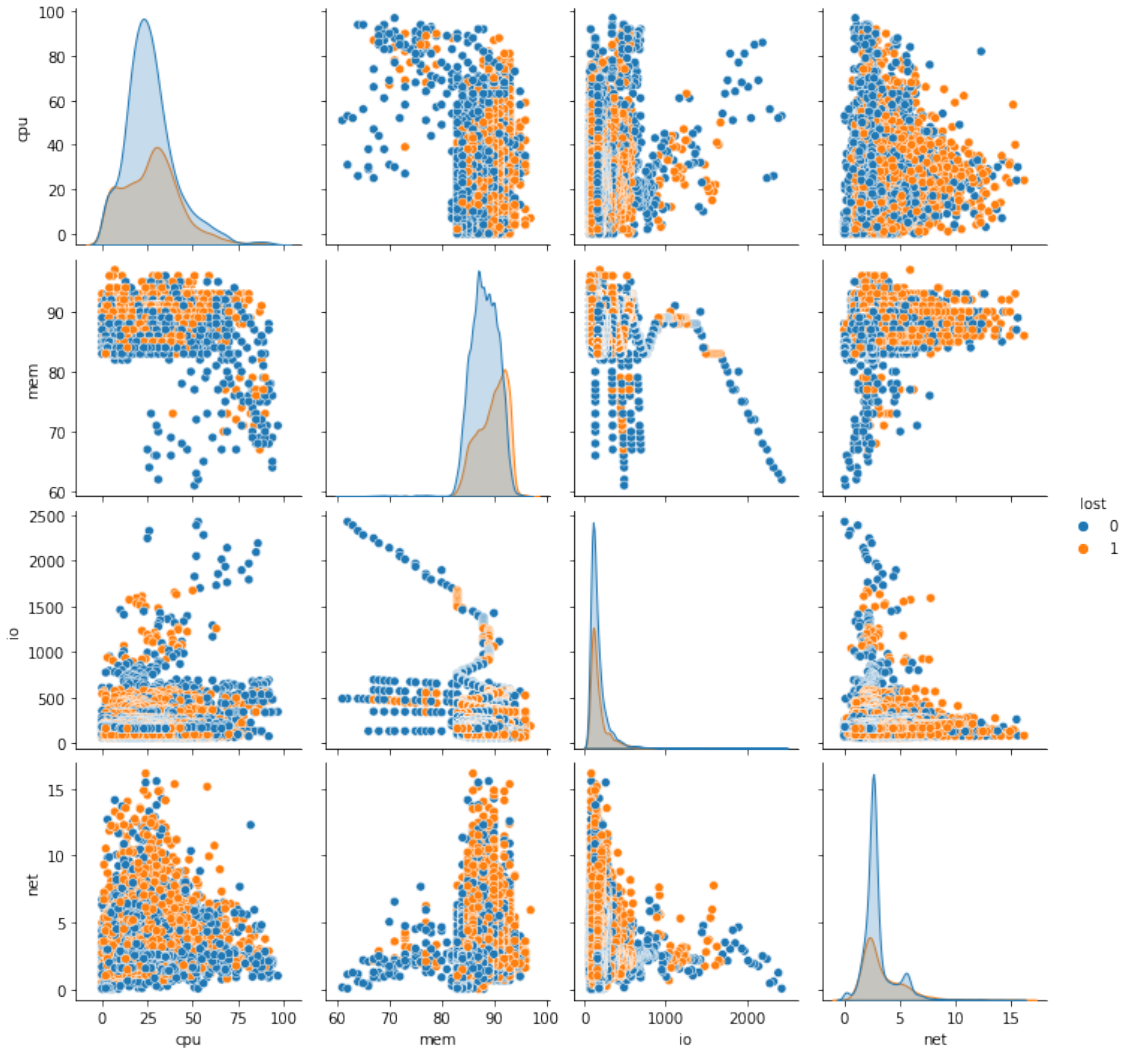


Figura 4.1: Diagrama de pares padrão apresentação a relação entre cada uma das variáveis do conjunto de dados.

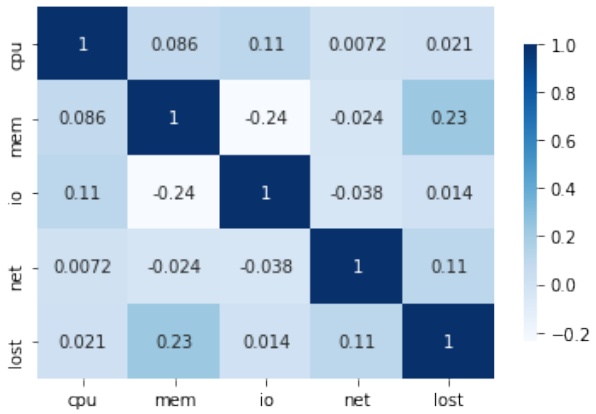


Figura 4.2: Mapa de correlação em cada uma das variáveis do conjunto de dados.

mais clareza que não existe quase correlação nenhuma entre as variáveis com o valor mais alto de correlação de 0.23 entre *mem* e *lost*.

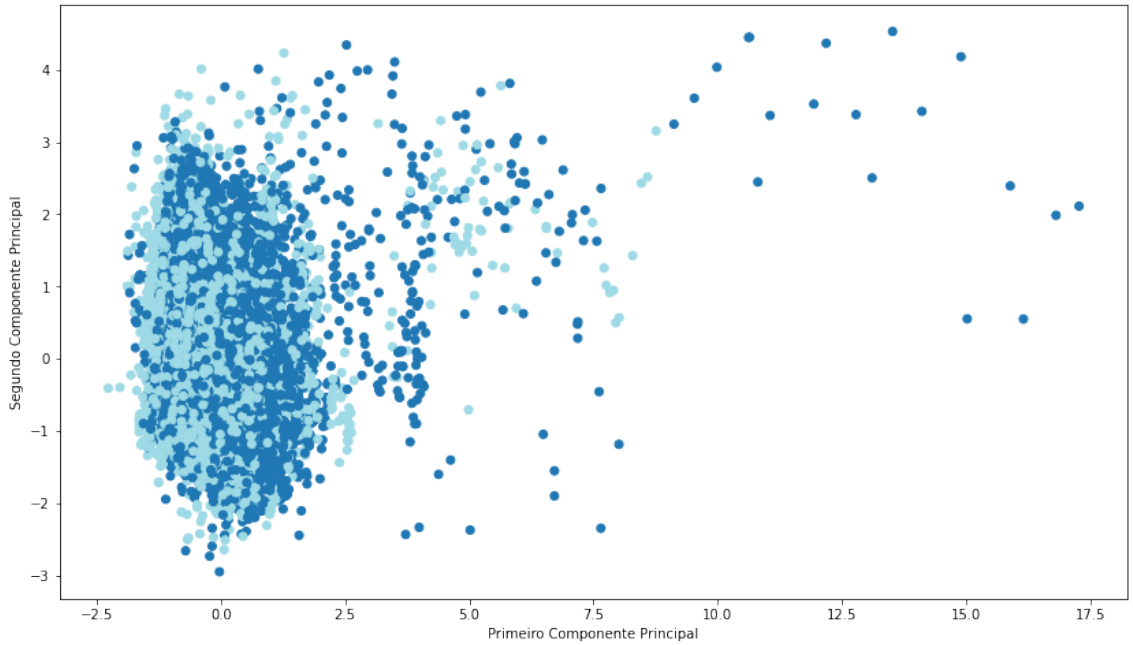


Figura 4.3: Representação da correlação dos componentes principais obtidos após a aplicação do PCA.

Buscando entender melhor a relação entre as variáveis foi utilizada a técnica não supervisionada de Análise de Componentes Principais (Principal Component Analysis - PCA) com o resultado na Figura 4.3. Como já observado antes, não foi encontrado nenhuma linearidade que explicassem as variáveis. Também foi obtido um percentual de explicação de variância de ambos componentes de 57.62%. Porém, segundo o gráfico do PCA, podemos observar que os pontos representando cada um dos componentes principais estão próximos um do outro, o que nos sugere que possa haver algum tipo de agrupamento. Por conta disto, resolvemos explorar os algoritmos de classificação.

## 4.3 Algoritmos de ML

Após a análise exploratória de dados foi realizado a aplicação de alguns algoritmos de ML no conjunto de dados de forma a obter o algoritmo que, aplicado junto ao ARTSIA, possa diminuir a quantidade de requisições perdidas oriunda da camada terminal.

Com base nos dados do conjunto de dados e na análise realizada na seção anterior foi seguido a linha de algoritmos de ML supervisionados de classificação. Foi feito a análise com os seguintes algoritmos: Logistic Regression, K-Nearest Neighbors, Random Forest, Support Vector Machine e Multilayer Perceptron que serão descritos a seguir.

### 4.3.1 Logistic Regression

Logistic Regression (LR) é um algoritmo de ML que utiliza técnicas estatísticas de regressão logística para criar um modelo de predição de valores baseados em uma variável categórica. A variável categórica geralmente é uma variável binária [21]. A regressão logística utiliza como base um conjunto de observações de variáveis explicativas contínuas e/ou binárias. O modelo de regressão utilizado é útil para modelar a probabilidade de um evento ocorrer em função de outros fatores. O modelo também utiliza de uma função estatística chamada *logit*.

A Equação 4.1 apresenta a fórmula de regressão logítica para variável categórica utilizada no conjunto de dados deste trabalho. Os valores dessa variável são 0 ou 1.

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}} \tag{4.1}$$

### 4.3.2 K-Nearest Neighbors

O algoritmo K-Nearest Neighbors (KNN) utiliza um método estatístico não-paramétrico para classificação e regressão. No caso de classificação, a saída do algoritmo é uma associação de classe, já no caso de regressão é um valor de propriedade [21]. O algoritmo KNN utiliza uma aprendizagem baseada em instância onde a função apenas aproxima localmente e os cálculos são adiados até a avaliação da função. Esse é um algoritmo que depende da distância para realizar a classificação. Os $k$ vizinhos mais próximos contribuem mais para a média do que os mais distantes.

Na Equação 4.2, é descrito a fórmula Euclidiana da distância entre pontos calculados pelo KNN. A variável $n$ é a quantidade de atributos que descrevem as instâncias $p$ e $q$, que por sua vez, são dois pontos $n$-dimensionais no plano.

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{4.2}$$

### 4.3.3 Random Forest

Random Forest (RF) é algoritmo que trabalha criando uma variedade de árvores de decisão, com o conjunto de observações separadas para treinamento, de forma a obter a combinação de árvores que classifica os dados com maior acurácia [21]. RF também

pode ser utilizada para regressão e predição. RF tem por característica reduzir a variância. Consequentemente, isso ocasiona um pequeno aumento no viés e alguma perda de interpretabilidade, mas o modelo final é considerado de alto desempenho.

Para cada árvore de decisão, a importância dos nós são calculados utilizando o critério *Gini*. A Equação 4.2 representa a fórmula deste cálculo, onde $p_kL$ proporção da classe k no nó esquerdo e $p_kR$ a proporção da classe k no nó direito.

$$Gini = N_L \sum_{i=1}^{k} p_kL(1 - p_kL) + N_R \sum_{i=1}^{k} (p_kR(1 - p_kR) \tag{4.3}$$

### 4.3.4 Support Vector Machine

Em ML, Support Vector Machine (SVM) é outro algoritmo de aprendizagem supervisionado usado para regressão e classificação. A partir de um conjunto de observações e atuando na parte de treinamento do conjunto, o SVM constrói um modelo que atribui novas classificações, se tornando um classificador binário não probabilístico [21]. Esse algoritmo utiliza de um conceito de hiperplano e cria um hiperplano para separar as variáveis de entrada no espaço. SVM podem fazer classificações lineares e não lineares. A partir do conjunto de dados, o algoritmo identifica e aplica o tipo de classificação.

### 4.3.5 Multilayer Perceptron

Se tratando de redes neurais, Multilayer Perceptron (MLP) é um algoritmo com uma ou mais camadas ocultas e um número indeterminado de neurônios. Na Seção 2.5.1 é fornecida uma breve explicação do seu funcionamento.

## 4.4 Metodologia

Para implementação dos algoritmos de ML foi utilizada a linguagem de programação *Python* com a biblioteca *sklearn*. As execuções dos algoritmos foram feitas na plataforma de nuvem do Google conhecida como Colab[15]. A instância usada foi a n1-highmem-2, que possui as seguintes especificações: Intel Xeon E5 v4 de 2.2 GHz com 13 GB de memória.

Foi utilizado o conjunto de dados para aplicar cada um dos algoritmos descritos na

[15]https://colab.research.google.com/

seção anterior. A variável independente do nosso modelo é *lost*. Quando seu valor é 1 indica que existia naquele momento a necessidade de instanciar uma nova aplicação de forma a não perder requisições. Quando seu valor é 0 indica que não existia a necessidade de instanciar novas aplicações. Os algoritmos irão trabalhar a classificação em base de duas classes: 0 e 1. Logo, a classe 1 é a mais importante a ser identificada por impactar o funcionamento da aplicação.

Para todos os algoritmos utilizados foram separados 60% dos dados do conjunto de dados para treino e 40% para realização de testes. Os dados de treino e teste foram validados utilizando o coeficiente de determinação $R_2 - score$ [8]. $R_2 - score$ é definido como a variação explicada sobre a variação total.

Os resultados da aplicação dos algoritmos sobre modelo foram comparados utilizando um relatório de classificação com as seguintes métricas: acurácia, precisão, cobertura e pontuação f1. A definição de cada métrica é baseada nos valores da Matriz de Confusão conhecidos como Verdadeiros Positivos (VP), Verdadeiros Negativos (VN), Falsos Positivos (FP) e Falsos Negativos (FN) [49].

A *acurácia* indica o desempenho geral do modelo. No geral, essa medida responde o quanto o classificador está correto, ou seja, de todas as classificações realizadas, quantas o modelo classificou corretamente. A Equação 4.4 descreve o cálculo da acurácia.

$$acr = \frac{VP + VN}{VP + FN + FP + VN} \tag{4.4}$$

A *precisão* é a proporção de classes positivas preditas corretamente em relação ao total de classes positivas preditas. Esta medida nos fornece a seguinte informação: das classificações marcadas como corretas, quantas efetivamente estavam corretas. É possível observar o cálculo da precisão na Equação 4.5.

$$prc = \frac{VP}{VP + FP} \tag{4.5}$$

A *cobertura* é a efetividade do classificador de identificar as classificações positivas. De toas as classificações em que a classe positiva foi marcada como esperada, quantas estão corretas. A Equação 4.6 representa a cobertura através de valores da Matriz de Confusão.

$$cob = \frac{VP}{VP + FN} \tag{4.6}$$

Por fim, a *pontuação f1* é uma medida que combina precisão e cobertura, de forma que possa ser representada em apenas um número a qualidade geral do modelo. O cálculo da pontuação f1 pode ser visto na Equação 4.7

$$f1 = \frac{2 * prc * cob}{prc + cob} \tag{4.7}$$

## 4.5 Resultados

Primeiro foi realizado a validação dos dados para cada algoritmo aplicado com o resultado na Tabela 4.2. Sabendo que quanto mais próximo de 100, mais os dados estão adequados para aplicação do algoritmo, todos os algoritmos tiveram uma validação no mínimo razoável, com exceção do KNN e RF que tiveram boas validações. Os valores de validação de treino e teste próximos também demonstram que não houve *overfitting* na aplicação dos algoritmos. O *overfitting* ocorre quando o modelo superestima as diferenças e ruídos do conjunto de dados, gerando bons resultados com dados conhecidos, porém, criando regras pouco generalistas para dados desconhecidos.

Tabela 4.2: Validação dos dados de treino e teste com os algoritmos de ML aplicados

| **algoritmo** | **treino** | **teste** |
|---|---|---|
| LR | 69% | 68% |
| KNN | 80% | 70% |
| RF | 99% | 73% |
| SVM | 65% | 64% |
| MLP | 67% | 65% |

O algoritmo KNN tem a características de agrupar os valores no espaço utilizando $k$ instâncias como base. No experimento foram testados alguns valores de $k$ de forma a obter os melhores valores de métricas de classificação como pode ser visto na Figura 4.4. Foram utilizados os valores de 1 à 40 para $k$. Para comparar cada valor de $k$ etilizamos a taxa de erro, que consiste na média das diferenças entre os valores preditos e os valores de teste. Foi verificado que o valor que apresentou a menor taxa de erro foi $k = 4$. Também foi observado quanto maior o valor de $k$ a partir de $k = 4$, maior o valor da taxa de erro.

Também foi realizado testes com o algoritmo RF. RF cria uma série de árvores de decisões para ser utilizado no conjunto de dados de treinamento. No algoritmo a quantidade de árvores de decisões criadas é chamada de estimadores. A Figura 4.5 mostra os testes

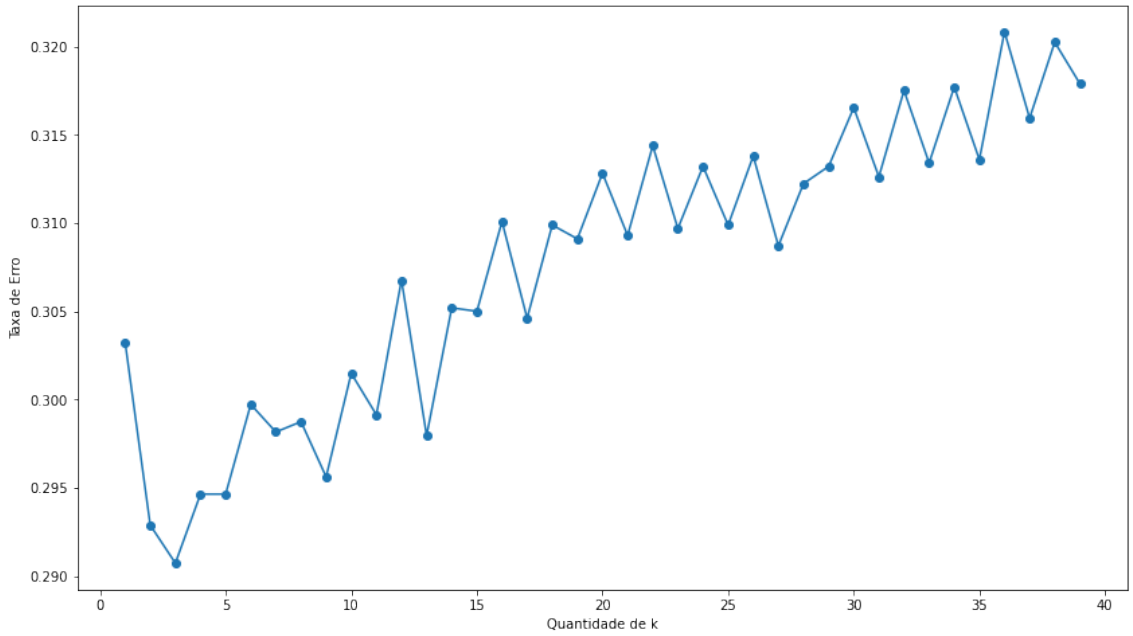


Figura 4.4: Evidência dos testes realizados para obter o melhor valor de $k$ onde no eixo X apresenta os valores de $k$ testados e no eixo Y a taxa de erro apresentado pelo KNN para cara um dos valores de $k$.

realizados com alguns estimadores. Foram utilizados valores de 1 à 500 para estimadores, com um intervalo de 10 em 10. A taxa de erro segue o a mesma definição utilizada nos testes de KNN. Observamos que a partir de uma quantidade de estimadores igual a 5 não existe muita variação de taxa de erro. Na aplicação para comparação com outros algoritmos utilizamos a quantidade de 460 estimadores.

Foram feitos testes na abordagem com redes neurais. Para os testes foi verificado que quando utilizado três camadas ocultas no algoritmo MLP pra o conjunto de dados ARTSIA, melhora a eficiência do algoritmo. Também foram realizados testes com a quantidade de neurônios como mostra a Figura 4.6. Para cada uma das camadas ocultas testamos de 1 à 100 neurônios. Novamente, o conceito de taxa de erro foi o mesmo utilizado em KNN. Foi verificado que a partir de 10 neurônios a taxa de erro não apresenta grandes variabilidades. Para efeitos de comparação foram utilizados 100 neurônios em cada camada oculta.

Após o ajuste dos algoritmos de ML, a Tabela 4.3 mostra os resultados para as métricas de avaliações. O algoritmo SVM se destaca de forma positiva por apresentar os melhores valores de *acurácia* e *precisão*, o que pode ser visto como um bom classificador, porém também se destaca de forma negativa pelos valores pífios em *cobertura* e *pontuação f1*, que demonstra sua falta de efetividade na classificação. O algoritmo MLP por mais testes que tenham sido feitos para buscar as melhores configurações do algoritmo

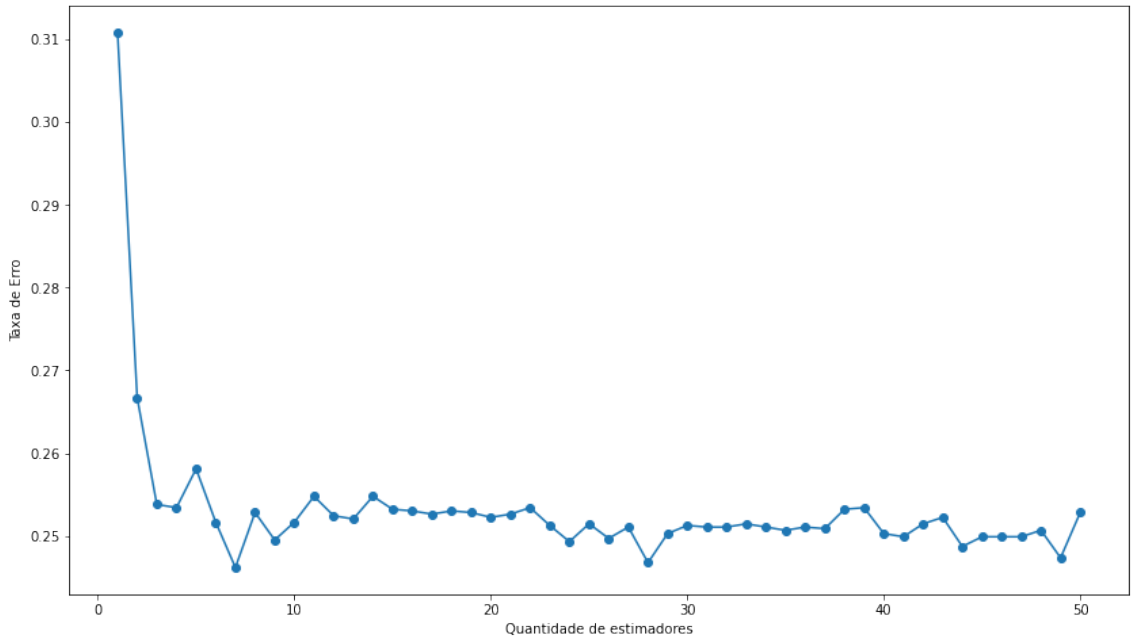


Figura 4.5: Análise da taxa de erros apresentada pelo RF para cada uma das quantidades de estimadores investigados.

para o presente cenário, se mostrou o algoritmo com resultado mais fraco. LR, KNN e RF tiveram valores próximos tanto de *acurácia* quanto *precisão*, mas o RF se destaca por ter o maior valor de *cobertura* e *pontuação f1*.

Tabela 4.3: Resultado para as métricas de avaliação dos algoritmos de ML aplicados predição de requisições.

| **algoritmo** | **acurácia** | **precisão** | **cobertura** | **pontuação f1** |
|---|---|---|---|---|
| LR | 67% | 64% | 27% | 38% |
| KNN | 69% | 67% | 35% | 46% |
| RF | 71% | 65% | 58% | 61% |
| SVM | 78% | 92% | 1% | 1% |
| MLP | 61% | 55% | 19% | 29% |

## 4.6 Discussão

Respondendo a questão de pesquisa *QP*1, foi verificado que sim, é possível diminuir a quantidade de requisições perdidas utilizando algoritmo de ML. A redução de quantidade de reqisições perdidas está relacionada com o momento correto em que é instanciado uma nova aplicação para atender a demanda. Os algoritmos KNN, RF e SVM apresentaram *acurácia* acima de 69% e *precisão* acima de 65%, com o SVM classificando com o maior valor de *precisão* alcançando os 92%. Com relação a questão de pesquisa *QP*2

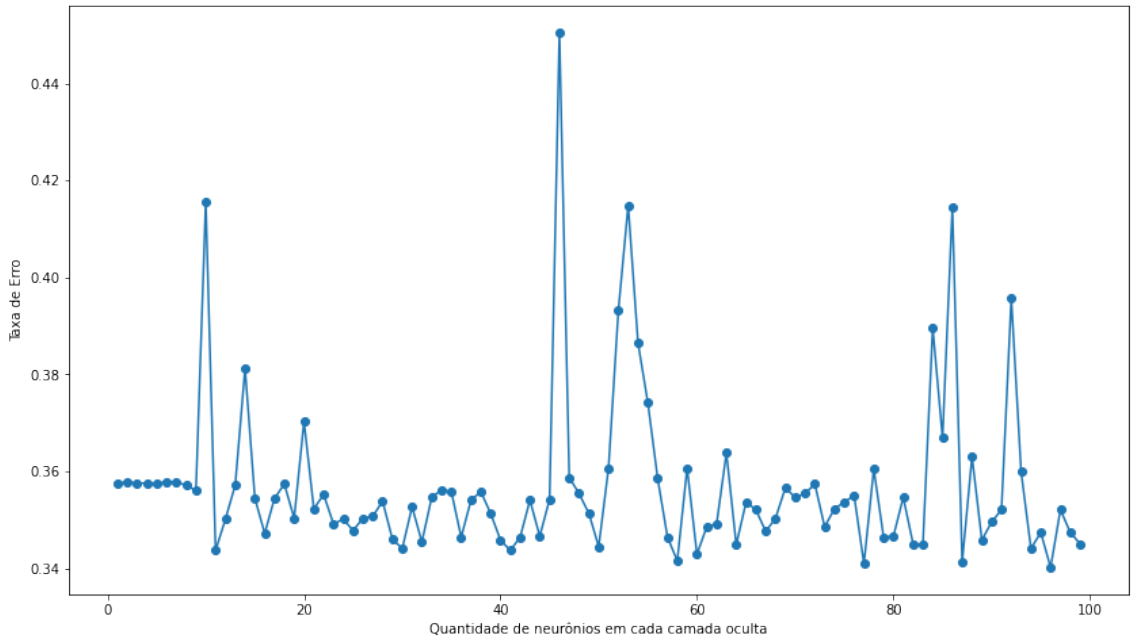


Figura 4.6: Análise da taxa de erros apresentada pelo MLP para cada uma das quantidades de neurônios de cada camada oculta (foram utilizadas 3 camadas) informados nos experimentos realizados.

foi verificado que, dentre os algoritmos utilizados na pesquisa, o algoritmo de ML que se mostrou mais eficiente em diminuir a quantidade de requisições perdidas consumindo o mínimo de recursos computacionais é o RF. Apesar de não apresentar os maiores valores em *acurácia* e *precisão*, ele apresentou bons valores com 71% e 65% respectivamente. Isso significa sua quantidade de acertos nos momentos em que precisa se criar uma nova instância da aplicação e consequentemente diminuir a quantidade de requisições perdidas. Apesar do SVM ter maiores valores *acurácia* e *precisão*, sua *cobertura* de 1% demonstra que o modelo irá instanciar uma nova aplicação praticamente todo o tempo, independente da necessidade real do sistema de borda. Isso leva a um consumo de recurso absurdo de forma desnecessária. Por fim, RF apresentou *pontuação f1* de 61%, que é uma métrica harmônica entre *precisão* e *cobertura*.

Por mais que este trabalho demonstre que conseguiu encontrar um algoritmo de ML que reduza a quantidade de requisições perdidas consumindo o mínimo de recursos computacionais, existe algumas questões que tem espaço para discussões.

De acordo com os testes realizados o algoritmo MLP foi utilizado com 3 camadas e 100 neurônios em cada uma. Poderia ter um estudo onde explora-se mais a estrutura deste algoritmo de forma a obter melhores resultados. Poderia também ser explorado algoritmos de aprendizado profundo e aprendizado por reforço visando conseguir maiores valores tanto de *precisão* quanto de *cobertura*.

# 5. TSITS - Uma Aplicação de Sistema de Transporte Inteligente usando Computação de Borda Móvel

O objetivo deste capítulo é apresentar a arquitetura TSITS (Time-Sensitive Intelligent Transportation System) assim como um estudo de caso de uma aplicação ITS sendo executada na arquitetura. Também realizamos um experimento comparando os resultados do acesso da aplicação ITS no TSITS e na Computação na Nuvem (Cloud Computing - CC) tanto através de banda larga quanto de 4G.

## 5.1 Visão Geral

Neste capítulo, será comparado os contextos de paradigmas de computação CC e MEC diante do funcionamento de uma aplicação ITS. Sendo assim o problema abordado será *como fazer a melhor utilização do paradigma de computação MEC*.

Os desafios em aberto (DA) e as principais contribuições (C) deste capítulo são:

1. (DA) Propostas de arquiteturas detalhadas de como aplicar os paradigmas de EC com aplicações ITS. (C) Será proposta a arquitetura TSITS explicando em detalhes cada componente, suas formas de trabalho e as integrações entre eles;
2. (DA) Realização de experimentos utilizando componentes reais. (C) Será mostrado o resultado de uma avaliação experimental fazendo uso do TSITS e comparando o desempenho da aplicação ITS que fornece pontos de interesse sendo executando tando na MEC quanto em CC;

Na Seção 5.2 é exibido os trabalhos relacionados de Computação de Borda Móvel com aplicações ITS. Baseado nas contribuições descritas na Seção 1.5, o presente capítulo aborda na Seção 5.3 o TSITS, uma arquitetura para MEC, detalhando cada componente, suas ações, suas interações e seus tempos de execução. Também é apresentado na

Seção 5.4, um estudo de caso de uma aplicação ITS sensível ao tempo no contexto de cidades inteligentes. A avaliação experimental na Seção 5.5, descreve as configurações da infraestrutura utilizada no experimento, o método e as métricas, assim como o resultado abrangendo cenários para CC e MEC. Por conseguinte, a Seção 5.6 revela uma discussão sobre alguns pontos de atenção do trabalho.

## 5.2 Trabalhos Relacionados

Baseado na literatura, é possível encontrar trabalhos sobre arquiteturas e aplicações para ITS baseadas em MEC.

O artigo [11] apresenta o sistema VTracer que contém métodos para compressão de trajetórias online e rastreamento de veículos. O VTracer utiliza de smartphones para tarefa de coleta de dados para uma posterior mineração. É utilizado um centro de dados para obter uma representação da trajetória de veículos em movimento livre de ruídos. Foi realizado um experimento onde o sistema foi instalado em um smartphone, implantado em um veículo, e foram feitos percursos pela cidade no objetivo de testar o sistema.

Os autores em [65] utilizam MEC para melhorar requisitos de latência e reconhecimento de contexto em sistemas ciber-físicos de transporte. Para tarefas de computação intensiva, os autores fazem uso de DL. Foi desenvolvido uma rede neural convolucional empilhada que consiste em camadas convolucionais de fatoração alternadas com camadas de compressão. Através de um experimento, eles comprovaram que o modelo utilizado pode manter a alta precisão enquanto se mantém portátil.

O CAMEVAN, uma arquitetura para redes veiculares sobre MEC, desenvolvida pelo trabalho [24], visa avaliar os recursos disponíveis em tempo real e fazer alocações para o recurso mais viável. Foi realizada uma abordagem matemática com algoritmo de decisão multicritério e validados os resultados com experimentos usando um banco de dados de recursos em nuvem. Os resultados apresentados demonstraram que uma classificação algorítmica de recursos físicos se aproxima de resultados experimentais e fornece um meio de delegar tarefas a um recurso disponível.

## 5.3 Arquitetura TSITS (Time-Sensitive Intelligent Transportation System)

Para apoiar nossa pesquisa em um aplicativo ITS usando MEC, projetamos uma arquitetura especializada chamada TSITS conforme a Figure 5.1. TSITS é uma variante da

arquitetura ARTSIA descrita na Seção 3.1, logo, apresentaremos as suas diferenças.

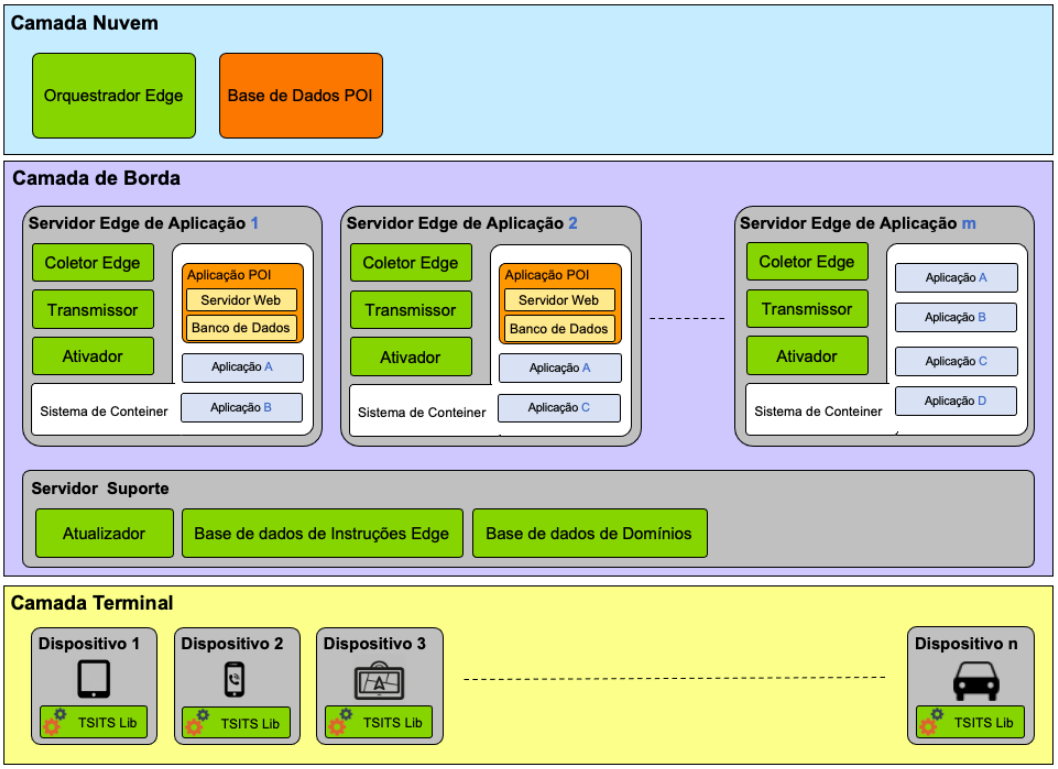


Figura 5.1: Camadas e componentes da arquitetura TSITS.

A camada terminal é onde os clientes e extensores de nosso aplicativo estão localizados. Os clientes são consumidores de serviços, obtendo pontos de interessse da cidade. Extensores são aplicativos de terceiros desenvolvidos para consumir nossos serviços usando o TSITS Lib. Essas interações de aplicações de ITS ocorrem por meio da comunicação da camada terminal com a camada de borda.

A camada de borda é a camada principal da arquitetura e onde ocorre a maior parte do processamento. Dispositivos móveis (clientes) se comunicam com aplicações diretamente após serem localizados com o auxílio de TSITS Lib. Ainda na camada de borda, existem servidores de aplicação. Nossa aplicação de estudo de caso, que chamamos de Aplicação POI, está implantada nestes servidores, assim como outras aplicações ITS que fornecem informações de interesse da cidade, que em nossa arquitetura, representadas na Figura 5.1, são chamados de Aplicativo A, Aplicativo B, entre outros. Cada servidor de borda possui um sistema de contêiner onde é possível virtualizar várias aplicações ITS no mesmo servidor de borda. O sistema de contêiner permite que os recursos do servidor (processamento, memória, armazenamento, etc.) sejam compartilhados entre as aplicações. Com esse compartilhamento, o sistema de contêiner adapta automaticamente o uso de recursos para cada aplicação de acordo com sua carga de trabalho. Porém, como os recursos de um servidor de borda são limitados, a arquitetura assume que existem servidores

*m*, de forma que um aplicativo pode ser escalado entre eles.

A camada de nuvem tem os serviços para gerenciar a infraestrutura de aplicações na camada de borda e dar suporte as aplicações ITS fornecidas.

A arquitetura TSITS divide o sistema MEC em seis módulos: TSITS Lib, Atualizador, Coletor Edge, Transmissor, Ativador e Orquestrador Edge. Assim como ARTSIA, cada módulo é independente e toda a interação entre os módulos é feita de forma assíncrona. Os módulos Coletor Edge, Transmissor e Orquestrador Edge já foram descritos junto ao ARTSIA. Todas as variáveis adicionais usadas na arquitetura TSITS em relação ao ARTSIA são descritas na Tabela 5.1.

Tabela 5.1: Variáveis usadas na arquitetura e algoritmo proposto.

| **Variável** | **Descrição** |
|---|---|
| *Tal* | intervalo para atualizar domínios |
| *Tabe* | intervalo para atualizar instruções de borda |

### 5.3.1 TSITS Lib

O módulo TSITS Lib é encapsulado em uma biblioteca que deve ser incorporada aos aplicativos ITS desenvolvidos por terceiros para integração com o sistema MEC proposto. O objetivo dessa biblioteca é fornecer um mapa com os endereços dos servidores de borda e quais aplicações eles fornecem, como um servidor DNS. Com isso, a responsabilidade de um serviço de fornecimento de domínio é transferida para o dispositivo móvel, liberando a camada de borda de se preocupar com o gerenciamento de domínios. O intervalo de atualização do banco de dados, *Tal*, é definido pelo aplicativo de terceiros.

### 5.3.2 Módulo Atualizador

O módulo Atualizador tem a responsabilidade de atualizar dois importantes bancos de dados para o melhor funcionamento do sistema MEC, a Base de Dados de Instruções Edge e a Base de Dados de Domínio. O primeiro é onde eles encontram instruções para qual aplicativo precisa executar aumento ou redução de escala e em qual servidor. O segundo, por outro lado, contém informações sobre domínios ativos e aplicações que devem ser usados por TSITS Lib. A cada intervalo *Tabe*, é realizada uma coleta de informações do camada de nuvem e atualizadas as duas bases de gerenciamento da camada de borda.

Alguns trabalhos que usam EC seguem o modelo de nuvem e usam balanceadores de carga [55,56]. Essa solução é eficiente ao trabalhar com recursos como o uso de processamento e/ou memória como entrada para algoritmos de elasticidade. Conforme declarado na seção de introdução, o foco do TSITS está em aplicações de ITS em que o tempo de resposta precisa ser curto devido à importância da entrega rápida de informações. Esse requisito torna a taxa de transferência um recurso principal para cálculos de elasticidade. Nesse caso, o balanceador se torna um gargalo, tornando a solução ineficiente. Portanto, a TSITS Lib torna-se uma boa opção, pois permite que aplicativos de terceiros saibam exatamente quais servidores estão disponíveis para acessar as aplicações diretamente.

### 5.3.3 Módulo Ativador

O módulo Ativador tem a responsabilidade de habilitar/desabilitar o uso de um aplicativo (aumento/redução). O funcionamento do Ativador no TSISTS é diferente do ART-SIA. Cada Servidor Edge possui um módulo Ativador. A cada instante, *Ta*, o ativador consulta o banco de dados de instrução do Edge para obter instruções, definidas pelo orquestrador de borda, no qual o servidor de borda, em que o ativador está atualmente, requer ações de elasticidade para as aplicações. Supondo que todas as informações necessárias estejam na camada de borda, podemos alcançar algum nível de independência de camada. Essa independência é importante no contexto de ITS e é um diferencial em comparação com outros trabalhos, como [52].

## 5.4 Estudo de Caso

O ambiente base do estudo de caso é uma cidade. O estudo é sobre como o cidadão pode obter informações sobre o que acontece na cidade para tomar a decisão de viajar no transporte público ou privado.

Os POIs (points-of-interest) da cidade são importantes para veículos e pedestres. Em nossa abordagem atendems a essas duas categorias de consumidores. Vários veículos já saem de fábrica com computadores de bordo, ou pelo menos sistemas de GPS embutidos. Esses sistemas GPS usam sistemas de software de fontes públicas ou privadas. Ambos os softwares podem usar nossa biblioteca para se integrar com nossa proposta de arquitetura MEC e obter os POIs da região. Os serviços públicos também possuem sistema de software específicos, dependendo de sua natureza. As viaturas policiais estão equipadas com sistemas especializados, onde por vezes é necessário conhecer o caminho a seguir para entrar na ação policial. As ambulâncias também são equipadas com sistemas de software

especializados para saber o melhor deslocamento, que muitas vezes pode significar uma decisão de vida ou morte. Outros casos são veículos utilizados por bombeiros, juízes, serviços de água e esgoto ou serviços de iluminação pública.

Uma grande vantagem para o serviço público em geral seria que a alimentação desses POIs poderia ser concentrada em um único local assim como sua distribuição. Os veículos particulares para passageiros e veículos de serviço também podem ser integrados ao TSITS. Sistemas já conhecidos na área de transporte, como Uber [16], Waze [17] e Google Maps [18] poderiam se integrar ao TSITS para fornecer aos seus usuários, POIs na região em que estão se deslocando. Os POIs também podem ser fixados como edifícios, como delegacias de polícia, hospitais, praças públicas, supermercados, estacionamentos, bombeiros, padarias e floristas. Dependendo de cada região, são inúmeras as possibilidades de edifícios de interesse dos cidadãos que se deslocam no trânsito. No entanto, os POIs também podem ser móveis. Por exemplo, um bloqueio de rua pode ser visto como um POI móvel. Dependendo da região, os POIs móveis podem ser protestos, eventos, canteiros de obras, etc.

Os serviços POI são fornecidos através do uso do TSITS Lib. A proposta de utilização dessa biblioteca do TSITS é abrangente, pois pode ser utilizada tanto por dispositivos veiculares como GPS e OBUs quanto por dispositivos móveis como smartphones e tablets. Dessa forma, os pedestres com seus dispositivos móveis também se beneficiariam com a prestação desse serviço. A TSITS Lib é responsável por manter conhecimento de todos os serviços que são prestados pela arquitetura TSITS, bem como dos endereços de cada servidor MEC ativo espalhado pela cidade.

Para se conectar a rede e acessar as aplicações, os dispositivos móveis podem se conctar atavés dos diversos pontos de acesso (AP) estratégicamente espalhados pela cidade. Os APs podem ser colocados em parques, praças, postes, etc. Cada AP cobre uma área da região. A área coberta depende do hardware do AP, que pode ser diferente, portanto, sua colocação deve ser planejada com cuidado. Portanto, veículos e pedestres podem se conectar ao TSITS através do AP de acordo com sua posição. Se o usuário estiver viajando, pode ser necessário conectar-se por meio de um novo AP, permanecendo na mesma rede e consumindo as mesmas aplicações.

A posição dos servidores do MEC não está diretamente relacionada aos principais atores do sistema, que são veículos e pedestres. Ter um dispositivo físico com um volume muito maior do que os APs, impõe limitações aos locais de sua instalação. Podemos

[16] https://www.uber.com
[17] https://www.waze.com/pt-BR
[18] https://www.google.com.br/maps

instalar servidores MEC em cabines quadradas, parques, postes ou unidades à beira da estrada. Alguns APs também funcionam como pontes, o que nos ajuda a cobrir os serviços em uma área mais ampla. No entanto, deve-se tomar cuidado para que a distância e o número de pontes não aumentem a latência do serviço.

### 5.4.1 O Serviço

O serviço visa a utilização de dados fornecidos por dispositivos móveis, como localização e raio dos POIs da cidade. Eles recebem uma lista de todos esses pontos a serem tratados pelo aplicativo móvel de terceiros.

Esse fluxo é representado pela atividade *Atualizar POIs* da Figura 5.2. A atividade *Atualização de Domínio* não é uma atividade da aplicação, mas uma atividade de interação entre os componentes da arquitetura onde a TSITS Lib sempre mantém os domínios do servidor atualizados para que os aplicativos no dispositivo terminal (smartphone, tablet, OBU) enviem solicitações para servidores MEC disponíveis. A solicitação é representada na Figura 5.2 pela requisição *GET/pois/lat/long/radius|list[POI]*, é realizada como serviços REST usando o método GET e passando a latitude, longitude e raio de interesse como os parâmetros de endereço nos quais os servidores MEC devem responder com uma lista de POIs. Com essa lista, o aplicativo de terceiros posiciona o usuário com os POIs de acordo com a finalidade do aplicativo.

No ciclo de serviço, supomos que haja uma fonte de dados para o aplicativo no servidor em nuvem. Não é o foco deste artigo discutir essa fonte de dados, mas encontramos na literatura várias maneiras diferentes de obtê-la, como selecionar POIs do OpenStreetMap para classificar uma área de risco de crime [12], tratando falsos positivos obtidos de sistemas ITS como uma marcação de tráfego interno [57], e modelagem de um sistema de POIs através de redes sociais [30].

Partindo desse pressuposto, de tempos em tempos os servidores do MEC buscam informações do servidor em nuvem e sempre mantêm sua base local atualizada com os POIs da cidade.

## 5.5 Avaliação Experimental

Nesta seção, avaliamos o comportamento do aplicativo POI diante de um grande número de solicitações. Por meio de uma avaliação experimental, comparamos o comportamento do aplicativo em cenários que usam MEC e CC. Por meio de uma análise

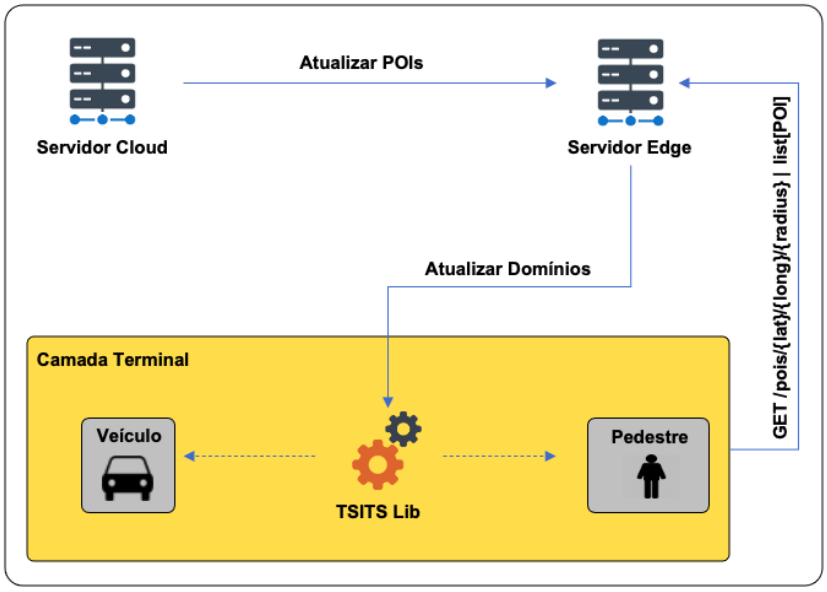


Figura 5.2: Fluxo de como o aplicativo POI é atualizado com informações recentes e como os dispositivos móveis fazem solicitações de aplicativo.

quantitativa da taxa de perda de solicitações, rendimento e métricas de tempo de resposta, buscamos avaliar a seguinte hipótese: *as aplicações MEC podem atender a um número maior de solicitações por segundo do que as aplicações em nuvem.*

### 5.5.1 Configurações

Na implementação, usamos o protocolo HTTP para comunicação. O aplicativo desenvolvido fornece serviços por meio do estilo de arquitetura REST. No ambiente MEC, cada aplicativo é um contêiner instanciado no Docker Engine. Os servidores MEC são computadores Raspberry Pi 3 com $1,2$ GHz 64-bit quad-core ARMv8, 1 GB de RAM e com Raspbian 10 OS instalado. Enquanto isso, no ambiente CC, cada aplicativo é uma VM que é executada em um serviço denominado EC2 de AWS Services. Os servidores em nuvem são instâncias de t2.xlarge com $3,0$ GHz quad-core x86, 16 GB de RAM e Ubuntu 18.04 OS instalado.

Para testar a aplicação ITS e nossa arquitetura, três cenários foram propostos:

- 4G: executar a aplicação em um ambiente de nuvem usando a tecnologia de rede 4G. Esse cenário seria o mais comum, pois a maioria das pessoas na cidade já possuem serviços de internet.
- Banda larga: executar a aplicação em um ambiente de nuvem usando uma conexão de banda larga em uma rede LAN. Esse cenário representa o caso em que uma MAN já havia sido distribuída pela cidade para oferecer uma conexão a dispositivos

móveis, para acessar os serviços de ITS que seriam hospedados na nuvem.

- MEC: executar a aplicação em ambiente MEC. Esse cenário é o proposto pelo estudo de caso.

### 5.5.2 Métodos

Para emular dispositivos móveis na camada terminal, uma ferramenta chamada Apache JMeter na versão 5.3 foi usada. Com essa ferramenta, foi possível emular vários dispositivos móveis emitindo um grande número de solicitações ao aplicativo ITS avaliado. Durante um intervalo de tempo de 30 minutos, emulamos um carregamento de solicitação com um mínimo de 300 threads simultâneos e um pico de 5000 threads, variando com o tempo. A mesma carga foi imposta em todos os cenários. Dentro dos cenários apresentados na Seção 5.5.1, várias rodadas foram realizadas para calcular a média e o intervalo de nível de confiança de 95% para cada cenário e métrica.

As métricas de desempenho usadas para comparar os cenários em que a porcentagem de solicitações perdidas devido a uma carga de solicitações geradas (taxa de perda de solicitação), o tempo médio de resposta das solicitações (tempo de resposta) e a média de solicitações por segundo (taxa de transferência) com suporte em cada cenário.

### 5.5.3 Resultados

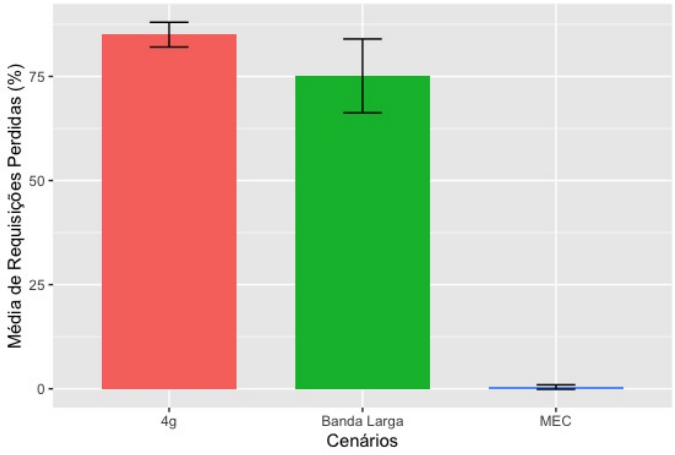


Figura 5.3: Número de solicitações enviadas ao aplicativo POI que foram perdidas nos cenários 4G, Banda Larga e MEC.

A Figura5.3 mostra a porcentagem de solicitações perdidas em cada cenário do experimento. Os cenários 4G e banda larga mostram resultados muito ruins com 85% e 75, 1% de solicitações perdidas. Embora os cenários 4G sejam ainda piores do que o cenário de Banda Larga, ele ainda apresentou menos variação durante as rodadas com um desvio

padrão de $2,97$ contra $8,87$ no cenário de Banda Larga. A comparação ainda é péssima quando feita com o cenário do MEC. Depois de executar o aplicativo na LAN, houve uma pequena perda de solicitações com uma média de $0,41\%$.

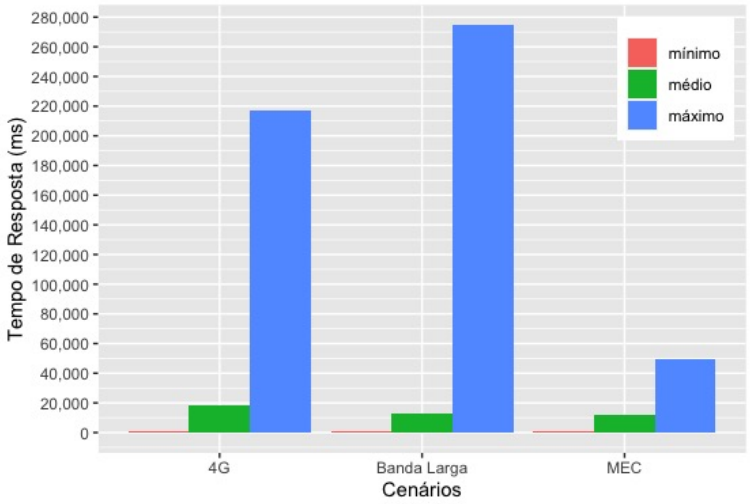


Figura 5.4: Representação do tempo mínimo, médio e máximo de resposta para solicitações de aplicação POI em cenários 4G, Banda Larga e MEC.

A Figura 5.4 mostra a média dos tempos de resposta mínimo, médio e máximo para cada um dos cenários em milissegundos (ms). Ao lidar com o tempo mínimo de resposta, todos os cenários mostraram resultados semelhantes abaixo de $1,2$ segundos. O tempo médio de resposta também permaneceu muito próximo entre 11 segundos e 18 segundos. Quanto ao tempo máximo de resposta, a diferença entre os cenários aumenta com 4G respondendo em aproximadamente 274 segundos, Banda larga com cerca de 217 segundos e MEC em torno de 49 segundos.

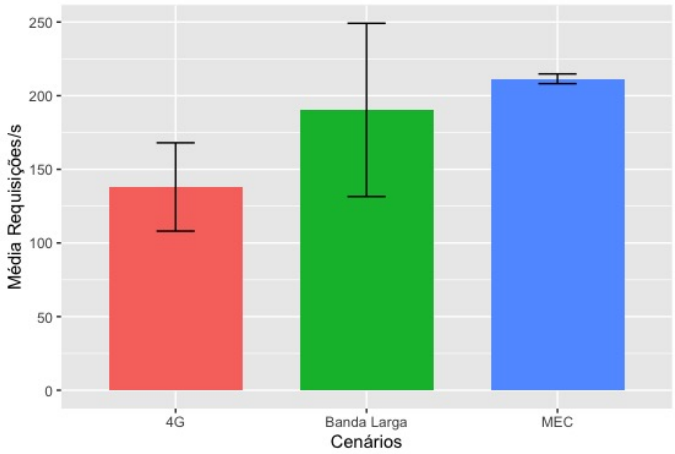


Figura 5.5: Número de solicitações por segundo suportadas pela aplicação POI em cenários 4G, Banda Larga e MEC.

Finalmente, a Figura 5.5 mostra o número médio de solicitações por segundo (req/s).

O pior cenário novamente foi 4G com 138 req/s e o melhor cenário foi MEC com 201 req/s. Em relação à variação do MEC, foi o cenário mais consistente com um desvio padrão igual a $3,32$ enquanto a Banda Larga foi o que apresentou a maior variação igual a $58,84$.

## 5.6 Discussão

Como resultado da análise na Seção 5.5.3, avaliamos a hipótese de pesquisa como verdadeira, pois o cenário MEC atende a um número maior de solicitações por segundo do que os cenários em que o aplicativo está na nuvem, conforme mostrado na Figura 5.5. Devido a uma maior largura de banda na LAN, observamos que quando geramos um grande número de solicitações, os servidores na nuvem demoram muito para responder às solicitações conforme mostrado na Figura 5.4 e justificam a grande quantidade de perda de requisições como visto na Figura 5.3.

Por mais que tenha sido comprovado que uma aplicação ITS sensível ao tempo atenda um maior número de requisições estando um ambiente MEC do que em ambiente CC, independente do tipo de conexão, existem pontos que ainda que merecem uma discussão.

Na arquitetura do TSITS é proposto um Servidor de Suporte que tem duas funcionalidades: (1) buscar e armazenar as instruções de elasticidade para o sistema de borda móvel e (2) buscar e armazenar as atualizações relacionadas ao domínios dos servidores de borda móvel. Uma melhoria de infraestrutura que poderia ser feita é extinguir este servidor e dividir suas funcionalidades entre o servidor de borda móvel e o dispositivo móvel. Os servidores de borda móvel assumiriam a funcionalidade de buscar e armazenar as instruções de elasticidade para o sistema de borda móvel, afinal, o módulo Ativador que consta nestes servidores é que utilizam os dados de instruções. Com relação a funcionalidade de atualização de dados de domínios poderia ficar a cargo do dispositivo móvel através da biblioteca TSITS Lib, mudança que, transferiria parte do processamento do sistema de borda móvel para os dispositivos móveis, assim como, tornaria a atualização de domínio mais rápida.

# 6. Conclusão

Este trabalho apresentou uma proposta chamada ARTSIA composta por uma arquitetura MEC e um algoritmo de elasticidade de aplicações em MEC chamado SABANN, que tem por objetivo suportar aplicações ITS que possuem necessidade de respostas rápidas devido à importância da informação no tempo. A abordagem proposta combina conceito de MEC e técnicas de elasticidade de aplicações para criar sistemas que mitiguem a quantidade de requisições perdidas oriundas da camada terminal. Foi demonstrado através de uma experimentação real cinco cenários de elasticidade utilizando a mesma aplicação ITS (aplicação de fornecimentos de POIs), quatro delas com métodos e políticas de trabalhos já existentes e a quinta baseada no SABANN. Foi proposta uma solução diferenciada baseada em ARNN e uma técnica de limiares adaptativos de elasticidade. Através dos experimentos realizados, foi verificado uma redução na quantidade de requisições perdidas com o uso do algoritmo proposto em relação aos outros algoritmos comparados, com uma taxa de $4,25\%$ de requisições perdidas, evidenciando a importância do presente trabalho para a área de MEC com suporte as aplicações ITS sensíveis ao tempo.

Fizemos uma investigação sobre o conjunto de dados gerado pelo experimento ARTSIA. Através de um análise exploratória dos dados, aplicamos alguns algoritmos de ML de classificação e concluímos que dos algoritmo de ML avaliados, o algoritmo é mais eficiente em diminuir a quantidade de requisições perdidas consumindo o mínimo de recursos computacionais é o algoritmo Random Forest com 71% de acurácia e 61% de pontuação f1.

Também foi apresentado um estudo de caso baseado em uma aplicação ITS usando a arquitetura MEC. Utilizamos como aplicação ITS com um serviço que busca pontos de interesse na cidade que pode ser consumido por aplicativos móveis de terceiros. Para a arquitetura MEC, foi proposto o TSITS de forma detalhada, descrevendo o funcionamento de cada componente. Por meio de uma avaliação experimental, desenhada para representar o estudo de caso, foi demonstrado o comportamento da aplicação ITS utilizando a

arquitetura MEC com TSITS, utilizando o CC em uma conexão de banda larga e o CC em uma conexão 4G. Por meio dos resultados do experimento, foi observado que um aplicativo MEC oferece uma média de 201 req/s, número maior de requisições por segundo do que os aplicativos em nuvem, bem como uma taxa de $0,41\%$ de perda de requisição muito menor.

Baseado no presente trabalho, seguem as sugestões para trabalhos futuros:

1. Sugestões para o ARTSIA
    (a) Poderia ser considerado a aplicação do Random Forest, algoritmo que teve o melhor desempenho na investigação realizada no Capítulo 4, no Orquestrador Edge, de forma a obter uma quantidade de requisições perdidas menor
    (b) Seria interessante que fosse feito um experimento utilizando mais de uma aplicação ao mesmo tempo recebendo cargas de requisições diferentes
    (c) Poderia ser utilizando uma carga de trabalho mais adequada as características de aplicações ITS
    (d) Poderia utilizar o SABANN ou outro algoritmo de ML junto ao Kubernetes ou K3s[19]
2. Sugestões para investigação com ML
    (a) Além dos algoritmos já aplicados na investigação, seria também de grande contribuição se fossem aplicados algoritmos de aprendizado profundo e aprendizado por reforço de forma a diminuir a quantidade de requisições perdidas com o mínimo uso de recursos computacionais
3. Sugestões para o TSITS
    (a) Seria interessante que houve uma variação nos cenários assim como a utilização de vários tipos de dispositivos para representar a camada móvel como tablets, smartphones e aparelhos de GPS
    (b) Poderia ser desenvolvido outras aplicações com requisitos diferentes para entender os variados comportamentos da MEC
    (c) Futuramente, poderia ser aplicado um cenário utilizando 5G

[19] https://k3s.io/

# Referências Bibliográficas


[1] Mohammad Aazam, Sherali Zeadally, and Khaled A Harras. Fog computing architecture, evaluation, and future research directions. *IEEE Communications Magazine*, 56(5):46–52, 2018.

[2] Yahya Al-Dhuraibi, Fawaz Paraiso, Nabil Djarallah, and Philippe Merle. Elasticity in cloud computing: state of the art and research challenges. *IEEE Transactions on Services Computing*, 11(2):430–447, 2017.

[3] Naylor G Bachiega, Paulo SL Souza, Sarita M Bruschi, and Simone Do RS De Souza. Container-based performance evaluation: A survey and challenges. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 398–403. IEEE, 2018.

[4] Paolo Bellavista, Antonio Corradi, and Alessandro Zanni. Integrating mobile internet of things and cloud computing towards scalability: Lessons learned from existing fog computing architectures and solutions. *International Journal of Cloud Computing*, 6(4):393–406, 2017.

[5] U Narayan Bhat. *An introduction to queueing theory: modeling and analysis in applications*. Birkhäuser, 2015.

[6] Kashif Bilal, Osman Khalid, Aiman Erbad, and Samee U Khan. Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers. *Computer Networks*, 130:94–120, 2018.

[7] Charles C Byers. Architectural imperatives for fog computing: Use cases, requirements, and architectural techniques for fog-enabled iot networks. *IEEE Communications Magazine*, 55(8):14–20, 2017.

[8] A Colin Cameron and Frank AG Windmeijer. An r-squared measure of goodness of fit for some common nonlinear regression models. *Journal of econometrics*, 77(2):329–342, 1997.

[9] Antonio Celesti, Davide Mulfari, Antonino Galletta, Maria Fazio, Lorenzo Carnevale, and Massimo Villari. A study on container virtualization for guarantee quality of service in cloud-of-things. *Future Generation Computer Systems*, 99:356–364, 2019.

[10] Moumena Chaqfeh, Nader Mohamed, Imad Jawhar, and Jie Wu. Vehicular cloud data collection for intelligent transportation systems. In *2016 3rd Smart Cloud Networks & Systems (SCNS)*, pages 1–6. IEEE, 2016.

[11] Chao Chen, Yan Ding, Zhu Wang, Junfeng Zhao, Bin Guo, and Daqing Zhang. Vtracer: When online vehicle trajectory compression meets mobile edge computing. *IEEE Systems Journal*, 14(2):1635–1646, 2019.

[12] Paweł Cichosz. Urban crime risk prediction using point of interest data. *ISPRS International Journal of Geo-Information*, 9(7):459, 2020.

[13] Luis M Contreras and Carlos J Bernardos. Overview of architectural alternatives for the integration of etsi mec environments from different administrative domains. *Electronics*, 9(9):1392, 2020.

[14] Pedro Cruz, Felipe F da Silva, Roberto Goncalves Pacheco, Rodrigo De Souza Couto, Pedro Braconnot Velloso, Miguel Elias Mitre Campista, Luís Henrique Maciel Kosmalski Costa, et al. Sensingbus: Using bus lines and fog computing for smart sensing the city. *IEEE Cloud Comput.*, 5(5):58–69, 2018.

[15] Vitor Goncalves da Silva, Marite Kirikova, and Gundars Alksnis. Containers for virtualization: An overview. *Applied Computer Systems*, 23(1):21–27, 2018.

[16] Jingyun Feng, Zhi Liu, Celimuge Wu, and Yusheng Ji. Mobile edge computing for the internet of vehicles: Offloading framework and job scheduling. *IEEE vehicular technology magazine*, 14(1):28–36, 2018.

[17] Walid A Hanafy, Amr E Mohamed, and Sameh A Salem. A new infrastructure elasticity control algorithm for containerized cloud. *IEEE Access*, 7:39731–39741, 2019.

[18] Simon Haykin. *Redes neurais: princípios e prática*. Bookman Editora, 2007.

[19] Jianhua He, Jian Wei, Kai Chen, Zuoyin Tang, Yi Zhou, and Yan Zhang. Multitier fog computing with large-scale iot data analytics for smart cities. *IEEE Internet of Things Journal*, 5(2):677–686, 2017.

[20] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

[21] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[22] Zhiyuan Jiang, Sheng Zhou, Xueying Guo, and Zhisheng Niu. Task replication for deadline-constrained vehicular cloud computing: Optimal policy, performance analysis, and implications on road traffic. *IEEE Internet of Things Journal*, 5(1):93–107, 2017.

[23] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235, 2019.

[24] Zachary W Lamb and Dharma P Agrawal. Analysis of mobile edge computing for vehicular networks. *Sensors*, 19(6):1303, 2019.

[25] Sebastian Lehrig, Richard Sanders, Gunnar Brataas, Mariano Cecowski, Simon Ivanšek, and Jure Polutnik. Cloudstore—towards scalability, elasticity, and efficiency benchmarking and analysis in cloud computing. *Future Generation Computer Systems*, 78:115–126, 2018.

[26] Keqin Li. Quantitative modeling and analytical calculation of elasticity in cloud computing. *IEEE Transactions on Cloud Computing*, 2017.

[27] Fang Liu, Guoming Tang, Youhuizi Li, Zhiping Cai, Xingzhou Zhang, and Tongqing Zhou. A survey on edge computing systems and tools. *Proceedings of the IEEE*, 107(8):1537–1562, 2019.

[28] Gaocheng Liu, Shuai Liu, Khan Muhammad, Arun Kumar Sangaiah, and Faiyaz Doctor. Object tracking in vary lighting conditions for fog based intelligent surveillance of public spaces. *IEEE Access*, 6:29283–29296, 2018.

[29] Lei Liu, Chen Chen, Qingqi Pei, Sabita Maharjan, and Yan Zhang. Vehicular edge computing and networking: A survey. *Mobile Networks and Applications*, pages 1–24, 2020.

[30] Shudong Liu. User modeling for point-of-interest recommendations in location-based social networks: The state of the art. *Mobile Information Systems*, 2018, 2018.

[31] Marcelo C Medeiros, Timo Teräsvirta, and Gianluigi Rech. Building neural network models for time series: a statistical approach. *Journal of Forecasting*, 25(1):49–75, 2006.

[32] Thiago Medeiros and Carlos Alberto Campos. Artsia: Escalabilidade de aplicações para sistemas inteligentes de transportes em um ambiente de computação em neblina.

In *Anais Estendidos do X Simpósio Brasileiro de Engenharia de Sistemas Computacionais*, pages 73–80. SBC, 2020.

[33] Thiago C. Medeiros, Elton Soares, and Carlos A. V. Campos. Artsia: Scalability of intelligent transportation system applications in a fog computing environment (em submissão). *IEEE International Conference on Communications*, 2021.

[34] Thiago C. Medeiros, Elton Soares, and Carlos A. V. Campos. An intelligent transportation system application using mobile edge computing (em submissão). *IEEE International Conference on Communications*, 2021.

[35] Rodolfo I Meneguette. A vehicular cloud-based framework for the intelligent transport management of big cities. *International Journal of Distributed Sensor Networks*, 12(5):8198597, 2016.

[36] Quang Tran Minh, Chanh Minh Tran, Tuan An Le, Binh Thai Nguyen, Triet Minh Tran, and Rajesh Krishna Balan. Fogfly: A traffic light optimization solution based on fog computing. In *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*, pages 1130–1139, 2018.

[37] Debadatta Mishra and Purushottam Kulkarni. A survey of memory management techniques in virtualized systems. *Computer Science Review*, 29:56–73, 2018.

[38] Pedro A Morettin and Clélia Toloi. Análise de séries temporais. In *Análise de séries temporais*. Blucher, 2006.

[39] Carla Mouradian, Diala Naboulsi, Sami Yangui, Roch H Glitho, Monique J Morrow, and Paul A Polakos. A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 20(1):416–464, 2017.

[40] Attila M Nagy and Vilmos Simon. Survey on traffic prediction in smart cities. *Pervasive and Mobile Computing*, 50:148–163, 2018.

[41] Michael A Nielsen. *Neural networks and deep learning*, volume 2018. Determination press San Francisco, CA, 2015.

[42] Ayman Noor, Devki Nandan Jha, Karan Mitra, Prem Prakash Jayaraman, Arthur Souza, Rajiv Ranjan, and Schahram Dustdar. A framework for monitoring microservice-oriented cloud applications in heterogeneous virtualization environments. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 156–163. IEEE, 2019.

[43] ONUBR. População mundial deve chegar a 9,7 bilhões de pessoas em 2050, diz relatório da onu. https://nacoesunidas.org/populacao-mundial-deve-chegar-a-97-bilhoes-de-pessoas-em-2050/. Accessed: 2020-07-13.

[44] Opeyemi Osanaiye, Shuo Chen, Zheng Yan, Rongxing Lu, Kim-Kwang Raymond Choo, and Mqhele Dlodlo. From cloud to fog computing: A review and a conceptual live vm migration framework. *IEEE Access*, 5:8284–8300, 2017.

[45] Kai Peng, Victor Leung, Xiaolong Xu, Lixin Zheng, Jiabin Wang, and Qingjia Huang. A survey on mobile edge computing: focusing on service adoption and provision. *Wireless Communications and Mobile Computing*, 2018, 2018.

[46] Pawani Porambage, Jude Okwuibe, Madhusanka Liyanage, Mika Ylianttila, and Tarik Taleb. Survey on multi-access edge computing for internet of things realization. *IEEE Communications Surveys & Tutorials*, 20(4):2961–2991, 2018.

[47] Stuart RUSSEL and Peter Norvig. Inteligência artificial. tradução de regina célia smille. *Rio de Janeiro: Campus Elsevier*, 2013.

[48] Hani Sami, Azzam Mourad, Hadi Otrok, and Jamal Bentahar. Fscaler: Automatic resource scaling of containers in fog clusters using reinforcement learning. In *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pages 1824–1829. IEEE, 2020.

[49] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information processing & management*, 45(4):427–437, 2009.

[50] Arthur Souza, Nélio Cacho, Ayman Noor, Prem Prakash Jayaraman, Alexander Romanovsky, and Rajiv Ranjan. Osmotic monitoring of microservices between the edge and cloud. In *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 758–765. IEEE, 2018.

[51] STATISA. Number of smartphone users worldwide from 2016 to 2021. https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. Accessed: 2020-09-28.

[52] Salman Taherizadeh, Vlado Stankovski, and Marko Grobelnik. A capillary computing architecture for dynamic internet of things: Orchestration of microservices from edge devices to fog and cloud providers. *Sensors*, 18(9):2938, 2018.

[53] Ahmed Tealab. Time series forecasting using artificial neural networks methodologies: A systematic review. *Future Computing and Informatics Journal*, 3(2):334–340, 2018.

[54] Van Loc Tran, Anik Islam, Jeevan Kharel, and Soo Young Shin. On the application of social internet of things with fog computing: a new paradigm for traffic information sharing system. In *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 349–354. IEEE, 2018.

[55] Chih-Lung Tseng and Fuchun Joseph Lin. Extending scalability of iot/m2m platforms with fog computing. In *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, pages 825–830. IEEE, 2018.

[56] Fan-Hsun Tseng, Ming-Shiun Tsai, Chia-Wei Tseng, Yao-Tsung Yang, Chien-Chang Liu, and Li-Der Chou. A lightweight autoscaling mechanism for fog computing in industrial applications. *IEEE Transactions on Industrial Informatics*, 14(10):4529–4537, 2018.

[57] James Van Hinsbergh, Nathan Griffiths, Phillip Taylor, Alasdair Thomason, Zhou Xu, and Alex Mouzakitis. Vehicle point of interest detection using in-car data. In *Proceedings of the 2nd ACM SIGSPATIAL International Workshop on AI for Geographic Knowledge Discovery*, pages 1–4, 2018.

[58] Kulkarni V.G. *Introduction to Modeling and Analysis of Stochastic Systems*. Springer, 2012.

[59] Shangguang Wang, Jinliang Xu, Ning Zhang, and Yujiong Liu. A survey on service migration in mobile edge computing. *IEEE Access*, 6:23511–23528, 2018.

[60] Antoni Wilinski. Time series modeling and forecasting based on a markov chain with changing transition matrices. *Expert Systems with Applications*, 133:163–172, 2019.

[61] Saniya Zahoor and Roohie Naaz Mir. Virtualization and iot resource management: A survey. *International Journal of Computer Networks and Applications*, 5(4):43–51, 2018.

[62] Alessandro Zanni, Stefan Forsstrom, Ulf Jennehag, and Paolo Bellavista. Elastic provisioning of internet of things services using fog computing: An experience report. In *2018 6th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 17–22. IEEE, 2018.

[63] Fan Zhang, Xuxin Tang, Xiu Li, Samee U Khan, and Zhijiang Li. Quantifying cloud elasticity with container-based autoscaling. *Future Generation Computer Systems*, 98:672–681, 2019.

[64] Wenyu Zhang, Zhenjiang Zhang, and Han-Chieh Chao. Cooperative fog computing for dealing with big data in the internet of vehicles: Architecture and hierarchical resource management. *IEEE Communications Magazine*, 55(12):60–67, 2017.

[65] Junhao Zhou, Hong-Ning Dai, and Hao Wang. Lightweight convolution neural networks for mobile edge computing in transportation cyber physical systems. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(6):1–20, 2019.

[66] Zhenyu Zhou, Haijun Liao, Bo Gu, Kazi Mohammed Saidul Huq, Shahid Mumtaz, and Jonathan Rodriguez. Robust mobile crowd sensing: When deep learning meets edge computing. *IEEE Network*, 32(4):54–60, 2018.