UNIVERSIDADE FEDERAL DE PERNAMBUCO

WELLISON RAUL MARIZ SANTOS

PROACTIVE ADAPTATION OF MICROSERVICE-BASED APPLICATIONS

Recife

2024

WELLISON RAUL MARIZ SANTOS

PROACTIVE ADAPTATION OF MICROSERVICE-BASED APPLICATIONS

A PhD thesis submitted by **Wellison Raul Mariz Santos** in partial fulfilment of the requirements for the degree of Doctor of Philosophy in the Postgraduate Course in Computer Science at the Universidade Federal do Pernambuco.

Concentration Area: Distributed Systems and Machine Learning

Supervisor: Nelson Souto Rosa

Co-supervisor: George Darmiton da Cunha Cavalcanti

Recife

2024

.Catalogação de Publicação na Fonte. UFPE - Biblioteca Central

SANTOS, Wellison Raul Mariz.
Proactive adaptation of Microservice-based Applications / Wellison Raul Mariz Santos. - Recife, 2024.
152f.: il.

Tese (Doutorado) - Universidade Federal de Pernambuco, Centro de Informática, Programa de Pós-Graduação em Ciências da Computação, 2024.
Orientação: Nelson Souto Rosa.
Coorientação: George Darmiton da Cunha Cavalcanti.

1. Proactive Self-adaptive Systems; 2. Auto-Scaling; 3. Microservices; 4. Time Series Forecasting; 5. Ensemble Learning; 6. Cloud Computing. I. ROSA, Nelson Souto. II. CAVALCANTI, George Darmiton da Cunha. III. Título.

UFPE-Biblioteca Central

Folha de aprovação: Inserir a folha de aprovação enviada pela Secretaria do curso de Pós-Graduação. A folha deve conter a **data de aprovação**, estar **sem assinaturas** e em formato **PDF**.

I dedicate this work to my mother and wife, who always encouraged and supported me, never sparing any effort to help me.

ACKNOWLEDGEMENTS

I am incredibly grateful to my wife, Mayane Mariz, who has unwavering support and belief in me throughout the journey of completing my thesis. Your love, patience, and encouragement have been the cornerstone of my success. Furthermore, her faith in me, even during times of doubt and stress, gave me the strength and motivation to pursue my academic goals. The sacrifices she has made and her endless encouragement have been the backbone of my success. I could not have done this without you by my side.

I want to express my deepest gratitude to my family for their unwavering support and love throughout my studies. I sincerely thank my father, Paulo Alves, my mother, Maria de Fátima, and my brother, Wesley Arthur. Your constant source of strength and inspiration has been invaluable to me. Your love, patience, and steadfast belief in me have been the foundation of my success. I could not have done this without you. Your encouragement and understanding have meant the world to me.

I am incredibly grateful to my advisor, Dr. Nelson Rosa, for his unwavering trust, availability, understanding, and support throughout the development of my work. His insightful discussions have guided me in the best possible way, and his ethical, professional, and human approach has created an environment that prioritises growth. Above all, I am thankful for his willingness to share his knowledge and expertise.

I also express my profound gratitude to my co-advisor, Dr. George Cavalcanti, for his invaluable support and guidance throughout my research journey. His expertise and constructive feedback have been crucial in refining my work and pushing the boundaries of my understanding. I am deeply thankful for his mentorship and its positive impact on my academic journey.

I also want to express my deep gratitude to all the friends I met at Universidade Federal do Pernambuco.

I want to thank the Fundação de Amparo à Ciência e Tecnologia do Pernambuco (FACE-PE), which provided me with financial resources to perform this work.

Finally, I want to thank the Universidade Federal do Pernambuco professors for their invaluable knowledge, which was crucial to my development as a researcher.

“The best way to predict the future is to create it" - (This sentence is usually attributed to Peter Drucker)

ABSTRACT

Proactive auto-scaling of Microservice-based Applications has become popular in industry and academia. Proactive systems analyse historical data patterns to estimate future trends, assuming they will occur again. Early detection of potential problems, like high latency, enables prompt action, including service replication, to fix the issues before they arise. Several studies propose proactive auto-scaling systems for microservices. However, they have design limitations in their forecasting systems that may negatively impact forecast runtime accuracy. For example, all these systems rely on a single forecasting model for the prediction task. Using a single forecasting model increases the risk of inaccurate estimates, leading to unsuitable interventions that could harm the customer experience. This work presents **PMA** (**P**roactive **M**icroservices **A**uto-scaler), a MAPE-K-based auto-scaling system that uses forecasting models to anticipate and avoid microservices performance issues. PMA offers three models to address existent design limitations: univariate, multivariate and a Multiple Predictor Systems strategy that uses multiple models for prediction. Several experiments were performed to evaluate PMA and compare its performance to Predict Kube (PK), a leading adaptive industry tool. In 93.75% of the experiments, PMA outperformed PK for managing the applications. This work aims to improve proactive microservices auto-scaling systems, addressing some of their current design limitations to develop a more accurate and reliable forecasting system.

Keywords: Proactive Self-adaptive Systems. Auto-Scaling. Microservices. Time Series Forecasting. Ensemble Learning. Cloud Computing.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ACF *Autocorrelation Function*

API Application Programming Interface

AR *Auto-Regressive*

ARIMA AutoRegressive Integrated Moving Average

ARV Average Relative Variance

Bi-LSTM Bi-directional Long Short-term Memory

BM Best Model

CD Critical Difference

CFA Classical Forecasting Approach

CMT Container Management Tool

DARNN Dual-Stage Attention-Based RNN

DBN Deep Belief Networks

DeepState Deep State Space Model

DL Deep Learning

DS Dynamic Selection

DTW Dynamic Time Warping

DW Dynamic Weighting

DWS Dynamic Weighting With Selection

ES Exponential Smoothing

GARCH Generalised Autoregressive Conditional Heteroskedasticity

GRU Gate Recurrent Unit

HPA Horizontal Pod Auto-scaler

HTM	Hierarchical Temporal Memory
IM	Intermediate Model
LR	Linear Regression
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MBA	Microservice-based Application
MET	Model Effort Time
MFT	Multivariate Forecasting Tool
ML	Machine Learning
MLP	Multilayer Perceptron
MLR	Multinomial Logistic Regression
MPS	Multiple Predictor Systems
MRA	Multiple Regression Analysis
MSE	Mean Square Error
OB	Online Boutique
PK	Predict Kube
QHD	Quarkus-HTTP-Demo
QoS	Quality of Service
RF	Random Forest
RL	Reinforcement Learning
RMSE	Root Mean Square Error
RoC	Region of Competence
SARFIMA	Seasonal Autoregressive Fractionally Integrated Moving Average

SARIMA	Seasonal Autoregressive Integrated Moving Average
SLA	Service Level Agreement
SLO	Service Level Objective
SMAPE	Symmetric Mean Absolute Percentage Error
SSA	Singular Spectrum Analysis
SVM	Support Vector Machine
SVR	Support Vector Regressor
TFT	Temporal Fusion Transformer
TSF	*Time Series Forecasting*
VM	Virtual Machine
WM	Worst Model
XGBoost	eXtreme Gradient Boosting

LIST OF SYMBOLS

AE_w	Auto-scaling efficiency
α	The amplitude of the periodic pattern
$AN_\nu(\mu)$	Absolute number of deployed instances of μ in the measurement interval ν
AN_{max}	The maximum number of pods a service allocates considering all its experiments.
APD	Accuracy Percentage Difference between two forecasting strategies
B	The constant that defines the number of active users added/removed to each i-th interval
C	The total cost of an experiment
CD	Critical Difference between two algorithms
CM	A competence measure
C_{max}	Normalisation factor that maps the C to a scale between 0 and 1
CMV	Current Metric Value
χ_{ν_i}	The number of requests in the time interval ν_i
$\chi_{nu}(i-1)$	The number of active users in previous interval
Δ	A list of learning algorithms
d_k	A distance vector
$dist_i$	The distance between ψ_{i-th} and x_j
DMV	The desired metric value
	DV The defined resource demand goal for the microservice
D_{sel}	A validation dataset
E	A set of accuracies of a pool P

E_{max}	The highest accuracy of in E
E_{min}	The lowest accuracy of in E
ϵ_α	The baseline accuracy of a forecasting approach
ϵ_β	The accuracy of the proposed approach
η	The set of forecasting times of a model
FV	The future forecasting resource demand
γ	The duration of the periodic pattern
H	A set of time windows
K	The size of the RoC
L	A set of time window sizes
λ	A set of time series database
λ'	A set of preprocessed time series database
M	A set of all microservices of an application that need to be scaled
N	A set of measurement intervals in the experiment
NPC	The number of current pods deployed
NR	The number of runs of an experiment
ω	An adjustable weight for the desired C and the RT ratio
P	A pool of forecasting models
$\hat{P}$	A pool of selected forecasting models
ϕ	A set of fitting times of a model
$p(t_i)$	The value forecasted to the observation t_i by the model *p*
$p_i(x_j)$	The value forecasted to the test pattern (x_j) by the model p_i
$\hat{P}(x_j)$	The final forecasting of the MPS

q	A critical value
R	The ratio between the *FV* and *DV*
RR	Number of required replicas for a microservice
$Rand(\mathbb{N}(0,1))$	A normal random noise with mean zero and unit variance in the workload
r_c	A random number
RT	The 95^{th} percentile of response time
RT_{max}	The normalisation factor that maps the RT to a scale between 0 and 1
r_{sign}	A random number
S	A similarity measure
s_c	A scale factor to inject a normal random noise with mean zero and unit variance in the workload
T	A time series
T_i	The i-th observation of a time-series T
T_{min}	Minimum value from a time series T
T_{max}	Maximum value from a time series T
T_{norm}	The normalised value of an observation of a time series
Υ	A set of trained models
υ_l^i	A model i with a sliding window size of l
x_j	A test pattern

CONTENTS

1 INTRODUCTION

This chapter presents the context of adaptation in microservices. It also outlines the primary motivations for this thesis, the problem being addressed, a summary of partial solutions, a proposal and unique contributions. Finally, it explains how the rest of this thesis is structured.

1.1 CONTEXT AND MOTIVATION

Microservices have become the *de-facto* method for building cloud-native applications as they align with the demands of cloud architectures. Microservices is a developing architectural style where an entire application is divided into several autonomous services (LEWIS; FOWLER, 2014). Each service performs a specific business operation, running as a separate process and communicating with each other over well-defined Application Programming Interfaces (APIs) (LEWIS; FOWLER, 2014; HASSAN; ALI; BAHSOON, 2017). Developing Microservice-based Applications (MBAs) benefits development teams by offering enhanced scalability, high availability, flexibility and resilience to failure (LARRUCEA et al., 2018).

Microservices have become widely adopted in the industry. A survey (VAILSHERY, 2021) revealed that 85% of large companies with 5000+ workers are adopting a microservices architecture. Some successful companies adopting this approach include Alibaba, Amazon, Facebook, Netflix, and Twitter (GAN et al., 2019b; LUO et al., 2021). Another survey (IBM, 2021) demonstrated that companies implementing microservices architecture had experienced significant benefits. These benefits include a 30% increase in customer satisfaction, a 29% increase in customer retention, a 28% reduction in time-to-market, a 28% improvement in application quality and a 27% increase in scalability.

Microservices have also been employed in various cloud areas such as serverless, fog, edge, and containerised applications (LI; XIA, 2016; TAHERIZADEH; STANKOVSKI, 2017; ABDULLAH et al., 2021; SCHULER; JAMIL; KüHL, 2021; TOKA et al., 2021; DANG-QUANG; YOO, 2022). They have been used to build applications such as data storage and analysis, customer relationship management, e-commerce and finance (IBM, 2021). However, properly adopting microservices is onerous and involves many challenges that must be considered, such as the complexity of learning microservices, security concerns, run-time performance degradation and the need for expert engineering alongside distributed data management (IBM, 2021).

1.2 PROBLEM STATEMENT

Managing microservices is more complex than managing a single monolithic application. Microservices were developed to meet new business requirements, including updating, adding, and replicating services without compromising the software while running. However, operating in such a dynamic environment increases the chances of Service Level Agreement (SLA) violations, resulting in financial penalties and performance degradation (TOFFETTI et al., 2015). For example, updating a microservice may introduce a bug and result in temporary unavailability of the application, which was not agreed upon in the SLA. Adaptation is a common approach to handling run-time microservice performance degradation, such as high response time and service unavailability. Adaptation is when an organism changes its structure or functions to fit its environment better (DA; DALMAU; ROOSE, 2011). Auto-scaling is a widely employed adaptation technique for microservices (ALIPOUR; LIU, 2017; TOKA et al., 2021; QASSEM et al., 2023). It dynamically adjusts application resources to meet workload demand. Auto-scaling can be done by replicating/removing microservices (horizontal scaling) or allocating or withdrawing their resources (vertical scaling). Adaptations in MBAs often occur in response to changes in expected behaviour, such as deploying new microservices replicas to handle unexpected high demand (AL-MASRI, 2018).

Such changes can be noticed after they occur (reactive) or predicted in advance (proactive). The latter has recently drawn more attention in the scientific community (KAKADE et al., 2023; SHIM et al., 2023; QASSEM et al., 2023) and companies[1] due to its benefits (MORENO et al., 2015; ANGELOPOULOS et al., 2016; MARTíNEZ et al., 2017) over the reactive approach. Proactive adaptation gives more time to explore larger solution spaces and allows the managed system to be adapted before it saturates (MARTíNEZ et al., 2017).

Changes can be noticed after they occur (reactive) or predicted in advance (proactive). The latter has recently drawn more attention in the scientific community (KAKADE et al., 2023; SHIM et al., 2023; QASSEM et al., 2023) and companies[2] due to its benefits (MORENO et al., 2015; ANGELOPOULOS et al., 2016; MARTíNEZ et al., 2017) over the reactive approach. Some benefits of proactive adaptation include more time to explore larger solution spaces and allowing the system to adapt before it saturates (MARTíNEZ et al., 2017). Meanwhile, proactive approaches can also introduce unnecessary overhead and complexity when predictions are inaccurate,

[1] https://dysnix.com/predictkube
[2] https://dysnix.com/predictkube

potentially leading to resource wastage or premature adaptations.

Some challenges that the adoption of proactive auto-scaling tools for microservices helps overcome include:

Challenge 1: Addressing inefficient resource management that compromises the scalability and resilience of microservices. The lack of proactiveness in MBAs management can lead to inefficient resource allocation and compromise the scalability and resilience of the application. Without continuous monitoring and adjustment, services may not scale appropriately in response to demand variations, leading to overloads during peak times and under-utilisation during slow periods. This not only negatively impacts performance and operational costs but also the managed system's ability to quickly adapt to changes in workload or to recover from failures efficiently, compromising service continuity and quality.

Challenge 2: Dealing with late problem detection to avoid degraded user experience. If issues are not addressed proactively, they may only be noticed after significantly impacting the managed system. Identifying problems late can result in unplanned downtime and make it harder to resolve issues, as finding and fixing the root cause becomes more challenging. Furthermore, a lack of proactivity can lead to inconsistent performance and availability of services, which can harm user experience.

Nonetheless, the proactive auto-scaling performance relies heavily on the accuracy of the predictive model. Thus, developing a proactive, adaptive tool requires more caution than a reactive one. Several approaches have been proposed to anticipate performance degradation issues through forecasting. However, as demonstrated in this thesis, they suffer from design limitations that could negatively impact run-time forecast performance, leading to drops in accuracy. Low prediction accuracy can lead to two potential undesirable outcomes. Firstly, it may fail to detect performance degradation. Secondly, it may predict a false positive. In either case, relying on such an estimation to take action can lead to issues such as high latency, unavailability, and failures. Thus, improving the forecast performance in proactive auto-scaling tools remains an open research question.

Existing adaptive proactive tools use different forecasting algorithms in their designs. Each work presents its chosen algorithm as the silver bullet for the prediction task. However, as the free lunch theorem (YAO; DAI; SONG, 2019) demonstrates, no model is best for all scenarios, and the effectiveness of auto-scaling proactive tools can be questionable without understanding the algorithms used. Likewise, they also employ different forecasting strategies, but there has yet to be a consensus among studies on which techniques are most appropriate in different scenarios. Furthermore, current adaptive proactive tools only use a single algorithm in their prediction tool, increasing the risk of inaccurate estimates. Therefore, addressing these limitations in forecasting algorithms and strategies is crucial to improving the overall performance and reliability of proactive auto-scaling tools.

Considering the challenges and research limitations identified, the main research problem addressed in this thesis is:

How to adapt Microservice-based Applications (MBAs) proactively?

Therefore, based on the problem mentioned above, the following hypothesis is investigated:

A multi-model self-adaptive tool that employs diverse learning algorithms and prediction techniques enhances the reliability of microservice auto-scaling by mitigating single-model dependency risks while improving prediction accuracy by combining multiple models.

The research questions guiding this thesis are derived from the hypothesis:

- **RQ1.1** - Which algorithms and prediction techniques are the most accurate and cost-effective (in terms of quick fitting and forecasting) for microservices forecasting, and how can these findings be applied to enhance proactive tools?
- **RQ1.2** - Can the proactive auto-scaling of MBAs be improved by delegating the forecasting task to several models?

1.3 PARTIAL SOLUTIONS

Numerous studies have demonstrated the effectiveness of proactive auto-scaling for auto-scaling MBAs. For example, Alipour & Liu (2017) forecasted CPU usage with Machine Learning (ML) models to assist resource auto-scaling. Podolskiy et al. (2018) compared several ML and statistical models to predict microservices series. Coulson, Sotiriadis & Bessis (2020) proposed a self-adaptive pipeline that auto-scales microservices based on future traffic estimated by a Deep Learning (DL) model. Marie-Magdelaine & Ahmed (2020) forecasted traffic with a DL model to auto-scaling microservices. Fontana de Nardin et al. (2021) reorganised the deployment of microservices applications to reduce energy consumption based on CPU usage forecasts from statistical models. Toka et al. (2021) and Dang-Quang & Yoo (2022) forecasted workload with the same purpose as *Alipour & Liu*.

Although several studies propose proactive auto-scaling tools for MBAs, they have design limitations that may negatively impact forecast run-time accuracy. For example, previous works often do not clearly explain why they chose a particular forecasting algorithm (ALIPOUR; LIU, 2017; PODOLSKIY et al., 2018; GALANTINO et al., 2021; MOHAMED; EL-GAYAR, 2021). Furthermore, some works are limited to specific forecasting algorithms, such as ML, without considering DL or statistical ones (YADAV; Rohit; YADAV, 2021; GOLI. et al., 2021; DANG-QUANG; YOO, 2021). At the same time, they claim their chosen algorithms are the silver bullet for microservice proactive auto-scaling, but if different algorithms are deemed best, none are indeed the one.

Furthermore, all previous works rely on a single model for the forecasting task. Although some of the studies mentioned above (TOKA et al., 2021; HUANG et al., 2021) have employed multiple models, they have applied the Classical Forecasting Approach (CFA) (SILVA; NETO; CAVALCANTI, 2021). This approach evaluates a set of learning algorithms and selects only the one with the highest forecast accuracy. However, the no free lunch theorem (YAO; DAI; SONG, 2019) demonstrates that no single model can be optimal for all scenarios. Therefore, relying solely on one model increases the risk of inaccurate estimates, which can lead to inappropriate interventions that may harm the customer experience. Linked to this, several research fields have reported that utilising a multi-model approach improves prediction accuracy compared to CFA (WIDODO; BUDI, 2011; KOURENTZES; BARROW; CRONE, 2014; ADHIKARI; VERMA; KHANDELWAL, 2015; MOURA; CAVALCANTI; OLIVEIRA, 2021).

Other limitations of current auto-scaling tools include employing a limited scope of microservices performance metrics to trigger adaptations, e.g., only CPU usage is considered (ALIPOUR; LIU, 2017; PODOLSKIY et al., 2018; ROSSI; CARDELLINI; PRESTI, 2020; MARIE-MAGDELAINE; AHMED, 2020; Fontana de Nardin et al., 2021; DANG-QUANG; YOO, 2021; TOKA et al., 2021). Also, some works focus only on forecast accuracy, ignoring model fit and forecasting times (COULSON; SOTIRIADIS; BESSIS, 2020; ROSSI; CARDELLINI; PRESTI, 2020; TOKA et al., 2021). At run-time, these times are crucial in auto-scaling tools operating in pay-as-you-go environments as microservices. The forecasting time indicates the duration of the predicting task, while the fit time determines the training duration. Both times impact the tool performance and the cloud budget allocation.

1.4 OBJECTIVES

This thesis aims to develop a proactive and adaptive tool for auto-scaling microservices. The tool employs different learning algorithms and prediction strategies to improve reliability by reducing the risks associated with relying on a single forecasting model and enhancing forecasting accuracy by combining multiple models.

To achieve this main objective, the following specific objectives are defined:

- Conduct a comparative analysis between different classes of learning algorithms for auto-scaling microservices.
- Perform a comparative study of different forecasting strategies for auto-scaling microservices.
- Propose a new forecasting strategy to enable multiple models into the forecaster component.

1.5 PROPOSAL

This thesis answers the previously mentioned research questions by introducing **PMA** (**P**roactive **M**icroservices **A**uto-scaler), a generic, holistic, adaptive, proactive tool that applies horizontal auto-scaling to managing MBAs. PMA is structured as a MAPE-K (IBM, 2006) control loop. It collects and preprocesss microservice performance metrics. Then, it uses the preprocessed performance metrics as input to predict demand using forecasting models. After, it inspects the estimation to identify any goal violations, e.g., microservice μ_1 will demand 90% of CPU usage, violating the threshold of 80%. Next, it reviews the violations and determines the best adaptive action to restore the performance of the managed system. Finally, if needed, PMA applies horizontal auto-scaling actions to adapt the microservices.

The main component of PMA is its prediction component. PMA offers three forecasting strategies: univariate, multivariate, and Multiple Predictor Systems (MPS). These strategies cannot be employed simultaneously. The univariate strategy estimates are based on historical data from a single performance metric, e.g., CPU usage. Conversely, the multivariate strategy uses historical data from multiple performance metrics, e.g., CPU usage and memory. The MPS strategy employs multiple models and can be implemented univariate or multivariate. Each strategy has advantages and disadvantages, which are discussed in more detail throughout the thesis. It is worth noting that PMA is the first self-adaptive proactive tool to provide three different forecasting strategies.

The proposed MPS strategy is a technique that helps to reduce uncertainties when selecting a single forecasting model (KOURENTZES; BARROW; CRONE, 2014). Its main idea is to improve the accuracy and reliability of estimates by combining different forecasting models. This means that the forecasting process is not dependent on a single source, which makes the tool more adaptable. The MPS models are trained to predict distinct microservice behaviours. Instead of having all models trained on general microservice behaviour, MPS creates models that are experts in predicting specific patterns. For example, one model may be an expert in predicting weekend workload, while another may be an expert in predicting evening hours. Therefore, when it is time to forecast, MPS can detect the current pattern and choose the most suitable model to estimate.

1.6 CONTRIBUTIONS

The main contribution of this thesis is PMA, a MAPE-K-based self-adaptive tool that proactively adapts microservices at run-time. PMA is a generic, holistic, adaptive, proactive tool that aims to manage microservices through horizontal auto-scaling. It offers three forecasting strategies: univariate, multivariate, and MPS. These strategies estimate the usage of performance metrics for a particular microservice at run-time. Based on this estimation, the tool decides whether an adaptation is needed. If necessary, the adaptation consists of executing scaling-in/scaling-out operations. PMA is the first self-adaptive proactive tool to offer three different forecasting strategies for auto-scaling microservices. Likewise, it is the first mechanism to use MPS.

Other contributions of this thesis are as follows:

- **An analysis of popular learning algorithms for forecasting microservices time series.** The analysis aims to guide the selection of learning algorithms for designing new adaptive and proactive microservice systems. It compares ten learning algorithms from the three most prominent predicting classes (i.e., ML, DL and statistical). The evaluated algorithms are a statistical algorithm (AutoRegressive Integrated Moving Average (ARIMA)) (PRYBUTOK; YI; MITCHELL, 2000)), five DL ones (Dual-Stage Attention-

Based RNN (DARNN) (QIN et al., 2017), Deep State Space Model (DeepState) (RANGAPURAM et al., 2018), DeepAR (SALINAS et al., 2020), Long Short-Term Memory (LSTM) (MA et al., 2020) and Temporal Fusion Transformer (TFT) (LIM et al., 2021)) and four traditional ML (Multilayer Perceptron (MLP) (HAYKIN, 2001), Support Vector Regressor (SVR) (DRUCKER et al., 1997), Random Forest (RF) (BREIMAN, 1996) and eXtreme Gradient Boosting (XGBoost) (CHEN; GUESTRIN, 2016)). The evaluation considers not only accuracy but also the fit and forecasting time of the algorithms. However, since no measure of competence computes model fitting and forecasting times, this thesis proposes a new competence measure called Model Effort Time (MET) to address this gap. This analysis presents the first comparison of well-known learning algorithms from three main predicting classes for predicting microservices performance metrics.

- **A comparative study of popular forecasting strategies for auto-scaling microservices.** This comparative study addresses a research gap comparing univariate and multivariate proactive auto-scaling of microservices applications. The Predict Kube (PK), one production-grade solution, is compared to a proposed custom-made proactive auto-scaling multivariate system named Multivariate Forecasting Tool (MFT). They were evaluated to adapt four popular open source benchmark applications (Daytrader (DAYTRADER, 2024), Quarkus-HTTP-Demo (QHD) (QUARKUS-HTTP-DEMO, 2024), Online Boutique (OB) (ONLINE-BOUTIQUE, 2024) and Travels (TRAVELS, 2024)) considering three forecasting horizons (1, 3, and 5 minutes). This comparative study is the first to evaluate both TSF strategies (i.e., univariate and multivariate) for proactive auto-scaling of microservices at run-time.

- **Multiple Predictor Systems (MPS) for predicting microservices performance metrics.** It introduces a new strategy for forecasting microservices time series by selecting the most suitable forecasters from a pool of models for each prediction. In the proposal, the Generation phase can generate homogeneous pools (several learning algorithms of the same type) or heterogeneous pools (combining different learning algorithms). In the Selection phase, the generated pool can be chosen through dynamic selection algorithms, such as Dynamic Selection (DS), Dynamic Weighting (DW), and Dynamic Weighting With Selection (DWS) (MENDES-MOREIRA et al., 2009), or statically combined using Mean and Median (MOURA; CAVALCANTI; OLIVEIRA, 2021). The Integration phase can differ depending on the Selection approach taken. It may not be required

(DS), or it can involve using a simple mean (Mean and Median) or a weighted mean (DW and DWS) to combine pool predictions. The proposed MPS is the first strategy that uses multiple predictors to improve the accuracy and reliability of a microservices prediction system.

1.7 OUT OF SCOPE

The research addresses issues related to the proactive auto-scaling of MBAs. Hence, the investigation does not include the following item associated with the topic.

Online learning. Model retraining is expected to be required in production due to the dynamic nature of microservices. Microservices are designed to be adaptable and flexible over time. However, this flexibility can cause changes in their behaviour, which may require their models to be retrained. Some reasons for retraining include microservice life-cycle evolution, application demand changes, and microservices topology reorganisation.

Decentralised control mechanisms. This thesis does not delve into decentralised control loop architecture, which distributes decision-making processes across multiple nodes, potentially increasing system resilience and scalability.

1.8 WORK ORGANISATION

This thesis is structured as follows.

Chapter 2 introduces the fundamental concepts used in this thesis, such as self-adaptive tools, microservices, time series and MPS.

Chapter 3 proposes the PMA, the main contribution of this thesis. The chapter begins with an overview of the PMA and presents its architecture. Next, it details the design and implementation of MAPE-K-based components: Monitor, Analyser, Planner, and Executor.

Chapter 4 presents a comparative analysis between different forecasting algorithms for microservices time series prediction. The chapter begins by introducing and discussing the problem of literature. Then, it outlines the experimental protocol of the comparative analysis. After that, the results of the analysis are presented and discussed. This chapter complements and enhances the understanding of the PMA univariate forecasting module.

Chapter 5 delves into a comparative study between univariate and multivariate forecasting for auto-scaling microservices. The chapter begins by presenting the literature problem and exposing the experimental protocol of the study. After that, the results are presented and discussed. This chapter enhances and expands the comprehension of the PMA multivariate forecasting module.

Chapter 6 proposes an MPS to forecast microservices time series. The MPS selects the most suitable forecasters for each prediction from a pool of models. The chapter begins by contextualising the problem and objectives of the chapter. Then, it shows the proposed method and details the experimental protocol applied. After that, it presents an experimental evaluation and discusses its results. This chapter provides a better understanding of how the PMA MPS forecasting module works.

Chapter 7 presents the experimental evaluation of the PMA. The chapter begins by introducing the experimental evaluation objectives. Then, it details the experimental protocol employed. After that, it presents the PMA evaluation outcomes and debates its results. Finally, it discusses how the PMA results correlate to Chapters 4, 5, and 6 results.

Chapter 8 summarised existing works related to PMA. The chapter begins by introducing self-adaptive reactive systems. Then, it covers proactive auto-scaling systems designed for microservices. After that, it shows proactive works for cloud applications. Next, it presents a comparative analysis of all works.

Finally, Chapter 9 brings the conclusion of this thesis. The chapter begins by showing the final remarks. Next, it presents the contributions and some limitations of this thesis. Then, it concludes by debating ways to broaden the research in future work.

1.9 READING SCRIPT

Consider using the following reading guides based on your familiarity with the discussed topics to facilitate comprehension of the presented thesis.

Readers with a background in time series forecasting and MPS should read the chapters sequentially as the thesis is organised. If the reader already has a background in time series and MPS, it will help them to understand the proposed forecasting strategies of PMA. Afterward, the readers can move on to a detailed explanation of each strategy provided in specific chapters.

Readers with little or no time series knowledge should first read the background (Chapter 2), followed by the PMA overview (Section 3.1). After, it is suggested that the chapters be read in the following order: 4, 5, 6, 3, 7, 8 and 9. This reading timeline first introduces forecasting strategies, making the PMA easier to understand.

Readers with knowledge of time series but none of MPS should first read the background, followed by the PMA overview (Section 3.1). After, it is recommended to read the chapters in the following order: 6, 3, 4, 5, 7, 8 and 9. This way of reading allows the reader to understand MPS and brings the background needed to understand the rest of the chapters.

2 BACKGROUND

This chapter overviews all concepts essential to understanding the proposal presented in this thesis. Section 2.1 introduces basic concepts about self-adaptive tools, emphasising the MAPE-K architecture. Section 2.2 discusses Microservice-based Applications (MBAs), Virtual Machines (VMs), containers and Container Management Tools (CMTs). Section 2.3 shows time series concepts and steps related to the forecasting process. Section 2.4 presents primary concepts about Multiple Predictor Systems (MPS).

2.1 SELF-ADAPTIVE TOOLS

Deploying and maintaining cloud-based systems without assistance has become challenging due to their complexity (HUEBSCHER; MCCANN, 2008). Various approaches have been suggested to aid run-time management, with autonomic computing being the standout.

Autonomic Computing aims to improve computer systems by reducing the necessity for human maintenance and increasing their adaptability (HUEBSCHER; MCCANN, 2008). Adaptation is when an organism changes its structure or functions to fit its environment better (DA; DALMAU; ROOSE, 2011). Therefore, adaptive managed systems adapt to changes in their execution environment to continue reaching their predefined goals (WEYNS et al., 2013).

Salehie & Tahvildari (2009) proposed a six-question taxonomy for eliciting requirements from self-adaptive tools.

- ***Where* does the need for change lie?** Which artefacts, layers, and granularity levels are affected? For instance, where would adaptation be necessary if there is a sudden change in workload?
- ***When* is the adaptation applied?** When must the adaptation be applied, and how often and at any time? Adaptive tools can be reactive or proactive:
  - **Reactive** adaptation takes place after a problem occurs;
  - **Proactive** adaptation occurs before the problem occurs.
- ***What* element and artefact need to be modified?** For instance, a microservice, acting as the element, may require additional computational resources, which are the artefacts.

- ***Why* does the adaptive tool need to act?** For instance, the adaptive system needs to act to decrease managed system response time and avoid Service Level Agreement (SLA) violations.

- ***How* does the adaptive tool adjust the managed system?** Any actions taken to bring the managed system back to its intended state or goal can be considered as *how*, including load balancing, fault tolerance, configuration management, etc. There are two prominent scaling strategies: horizontal scaling and vertical scaling.

  - **Horizontal scaling**, also known as scaling in/out, replicates/removes components. For instance, the adaptive tool instantiates a new VM due to a drastic workload increase.
  - **Vertical scaling**, also known as scaling up/down, increases/decreases the resources of the components. For instance, the adaptive tool reduces the number of vCPUs allocated to a VM due to low demand.
  - **Hybrid scaling** combines vertical and horizontal strategies to adapt the managed system.

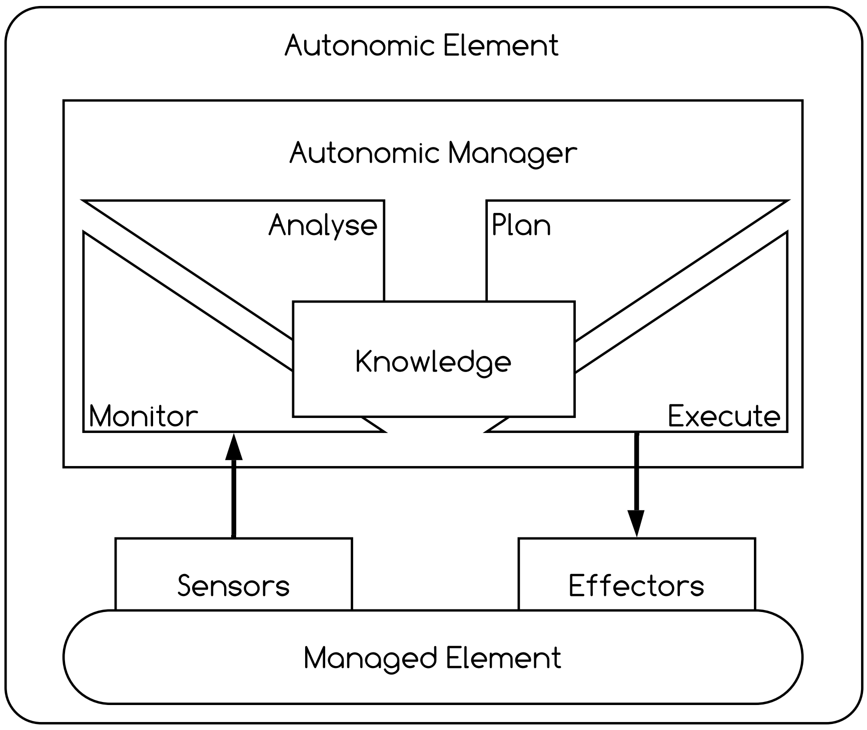
- ***Who* controls the adaptive process?** This question pertains to the extent of human-level participation required for the self-adaptive tool to operate effectively. For instance, cloud provider managers, system administrators, self-adaptive tool configurators, and application developers can all play crucial roles in the adaptive loop, contributing expertise, making decisions, and intervening manually when necessary.

2.1.1 Feedback Loop

The *Feedback Loops* are well-known architectures for developing adaptive tools (SALEHIE; TAHVILDARI, 2009; WEYNS et al., 2013). Figure 2.1 presents one of these architectures, known as MAPE-K. MAPE-K (KEPHART; CHESS, 2003; IBM, 2006; SALEHIE; TAHVILDARI, 2009) is a reference model that consists of cyclic phases described in the following.

- The *Managed Element* is a system that the Autonomic Manager manages at run-time.
- The *Monitor* collects, correlates, groups and filters information (metrics, symptoms, and policies) from the Managed Element through *Sensors*.

Figure 2.1 – MAPE-K architecture.



- The *Analyser* checks the collected information, seeking changes in the usual behaviour of the managed system.
- The *Planner* proposes actions that maintain the objectives of the Managed Element. It identifies the necessary changes and how to implement them.
- The *Executor* executes the proposed actions defined by the Planner through *Effectors*.
- *Knowledge* is a passive element used to store data from the four phases of the Autonomous Manager. It can maintain historical data, performance metrics, resource usage policies, and other relevant information..

2.2 MICROSERVICES

Microservices is a developing architectural style where an entire application is divided into several autonomous services (LEWIS; FOWLER, 2014). Inspired by service-oriented computing, microservices address the limitations found in monolithic architecture to meet new market demands. Monolithic architecture involves designing and developing a complete application as a single unit module. On the other hand, a Microservice-based Application (MBA) consists of a set of autonomous, independently deployed, and well-defined specific business functionalities services (TAIBI et al., 2017). The use of microservices brings development benefits (NEWMAN, 2015):

- Microservices applications comprise independent modules that can implement specific technologies, *eliminating technology lock-in and enabling software modularisation*. Therefore, developers can leverage the best technologies to reach established performance levels or address context-specific problems.
- Microservices have well-defined specific business functionalities. If a failure occurs, only part of the MBA, such as customer registration, is affected without breaking the entire application, thereby *improving fault tolerance*.
- The decoupling of microservice architecture enables services to be scaled individually, *improving the availability and resilience to varying workloads*.

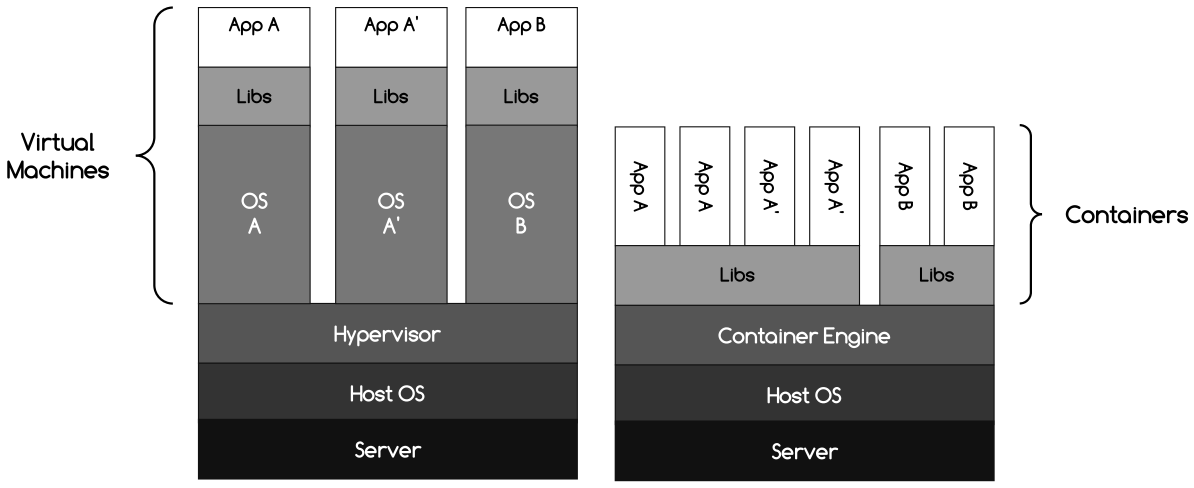
Along with the benefits, some challenges are inherent in using microservice-based architectures. MBAs are distributed systems whose highly dynamic components can be updated, removed, added, and replicated while the system executes. Therefore, MBAs require additional solutions to support continuous monitoring and deployment, testing, versioning and deprecating, and state management (LARRUCEA et al., 2018). One way to face these challenges is through self-adaptive tools (ANGELOPOULOS et al., 2016; ALIPOUR; LIU, 2017; SAMPAIO et al., 2019).

2.2.1 Containers and Virtual Machines

Cloud applications often use virtualisation techniques to achieve scalability (MALHOTRA et al., 2014). These applications can be deployed on VMs or containers. A VM can be defined as an efficient and isolated copy of a real machine (LAURENO, 2006). A *Hypervisor* is a software layer located between the hardware and the operating system that manages and allocates hardware resources from the actual machine (the host) to the VM (the guest), allowing multiple operating systems to run on the same device (DESAI et al., 2013). *Containers* offer a similar concept of virtualisation but are a lighter alternative because they consume fewer resources and are faster to provision (PAHL, 2015).

Figure 2.2 compares Hypervisors and containers. Hypervisor virtualisation is suitable when cloud applications require different operating systems, e.g., Ubuntu and Windows. If this is not a requirement, containers stand out because they share the operating system, binaries and libraries.

Figure 2.2 – Hypervisors vs Containers.



VMs and containers offer scalability but at different levels. For instance, suppose a scenario where a growth in application throughput can demand its adaptation, i.e., scaling out. Furthermore, consider that the growth tends to occur in specific system parts rather than being widespread (SAMPAIO et al., 2019). If the application is a microservice, each service could be deployed on a different VM, e.g., registering customers in VM_1 and shopping products in VM_2. Nevertheless, provisioning up a VM can take anywhere from one to over ten minutes (LI; WANG; RUAN, 2019). On the other hand, containers are a well-established technology for deploying microservices (PAHL, 2015). Containers do not allocate extra resources like the operating system, reducing expenses and speeding up provision compared to hypervisors. As a result, a single physical host can support hundreds of containers as opposed to a limited number of VMs (BERNSTEIN, 2014).

2.2.1.1 *Container Management Tool*

Managing deployment, scaling, storage, life-cycle, and communication becomes increasingly complex as the number of containers increases. To address these issues, Container Management

Tools (CMTs) like Kubernetes[1] and Docker Swarm[2] have emerged (FLORIO; NITTO, 2016).

2.2.1.2 *Kubernetes*

Kubernetes (K8s) (KUBERNETES, 2024a) is the *de-facto* standard for managing and orchestrating the deployment of containerised applications. It is an open-source container orchestration engine that automates application deployment, scaling, and management. In K8s, pods are the smallest deployable units of computing that can be created and managed. Each pod can contain one or multiple containers.

2.2.1.3 *Docker Swarm*

Docker Swarm is an open-source tool that manages and orchestrates resources like containers. Docker Swarm turns a group of nodes into a Docker Cluster. It abstracts the management complexity, offering a set of operations, such as deployment, update, and replication of containers, through a single Application Programming Interface (API). Docker Swarm has some advantages, such as being native to Docker and easy to configure and use (SOPPELSA; KAEWKASI, 2016).

2.2.2 Self-adaptive Tools For Microservices

Self-adaptive tools are commonly applied to handle the dynamism of managed systems like MBA. Two popular self-adaptive tools are detailed in the following:

2.2.2.1 *Horizontal Pod Auto-Scaler*

K8s has a reactive controller for adapting pods named Horizontal Pod Auto-scaler (HPA)[3]. HPA automatically scales the allocated resources, replicating or removing pods of the application to satisfy the service demand (HIGHTOWER; BEDA; BURNS, 2017). HPA triggers an adaptation when there is a mismatch between the current and desired metric value. Equation 2.1 defines the HPA trigger approach:

[1] https://kubernetes.io/
[2] https://docs.docker.com/engine/swarm/
[3] https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

$$\mathrm{RR} = \lceil \mathrm{NCP} \times \frac{\mathrm{CMV}}{\mathrm{DMV}} \rceil, \tag{2.1}$$

where RR is the number of required replicas to meet the current demand, NCP is the number of current pods deployed, and CMV and DMV are the current and desired metric values.

2.2.2.2 Predict Kube

Predict Kube (PK)[4] is an artificial intelligence-based predictive auto-scaling tool designed to optimise resource allocation and performance management in K8s environments. PK enhances K8s orchestration by incorporating predictive analytics with a proactive, data-driven resource management strategy. It integrates with the K8s ecosystem through Kubernetes Event-driven Auto-scaling (KEDA)[5] and uses Prometheus[6] as a database.

PK leverages *Time Series Forecasting* (TSF) techniques to predict future resource demands based on historical data. This proactive approach allows the tool to anticipate workload fluctuations and adjust resource allocation, minimising response times and improving overall performance. PK collects and preprocesses historical performance metrics and applies machine learning algorithms to model the managed system behaviour. Then, it uses the model forecasts to identify potential issues. It proactively applies horizontal auto-scaling actions to optimise and maintain managed system performance if issues are found.

2.3 TIME SERIES

A time series comprises observations, typically measured at successive intervals and uniformly spaced in time (LORIDO-BOTRAN; MIGUEL-ALONSO; LOZANO, 2014). Equation 2.2 defines a time series:

$$T = \{t_1, t_2, ..., t_n\}, \tag{2.2}$$

where t_1, t_2 and t_n are time-series observations, and *n* is the size of the time series.

4 https://dysnix.com/predictkube
5 https://keda.sh/
6 prometheus.io

Time series examples include daily temperature values measured by a sensor, application or server performance metric values per minute, hourly battery level values of an electronic device, and monthly traffic values transmitted by an internet service provider.

An essential characteristic of a time series is that its observations depend on each other, making it possible to analyse and model these dependencies (EHLERS, 2007). Time series analysis is widely used in various fields to tackle problems, including temporal association (SOMPOLINSKY; KANTER, 1986), time series grouping (KISILEVICH et al., 2010), time series classification (GRAVES et al., 2006) and TSF (ALIPOUR; LIU, 2017).

Time series can also be classified as univariate or multivariate. A univariate time series is a series with a single time-dependent variable, also called a feature. A multivariate series has two or more time-dependent features. In multivariate series, each variable depends not only on its past values but also on the other variables.

2.3.1 Time Series Forecasting

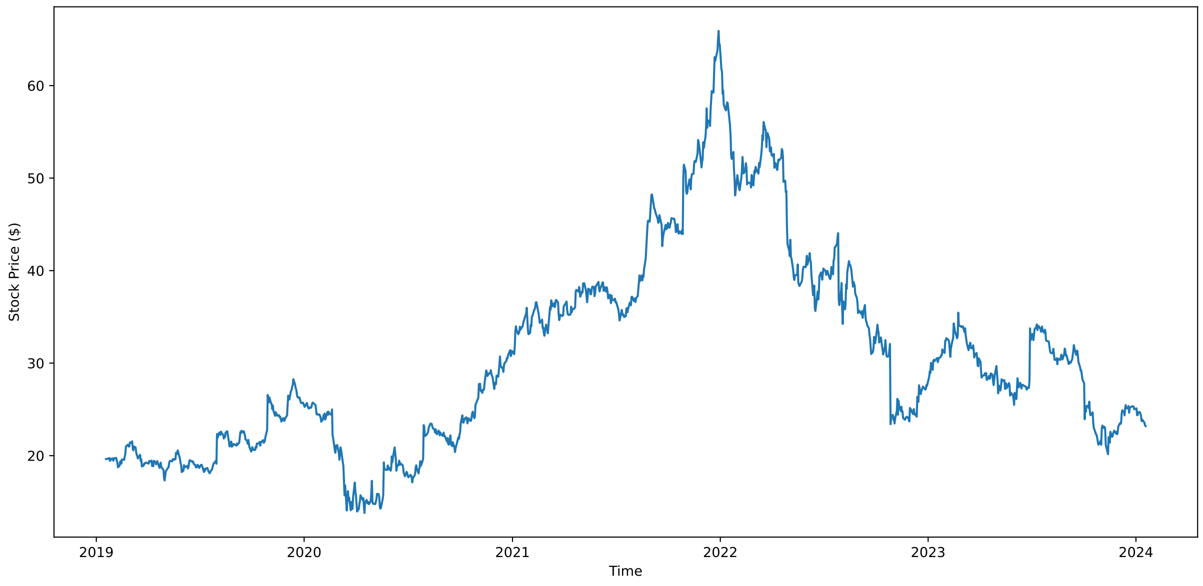
This thesis focuses on *Time Series Forecasting* (TSF) of microservices performance metrics. TSF involves building a model that can accurately represent the behaviour of a given series and utilise it to make predictions. Three popular classes of algorithms have been used in the literature for creating representative TSF models: Machine Learning (ML), Deep Learning (DL) and statistical (MARIE-MAGDELAINE; AHMED, 2020; ROSSI; CARDELLINI; PRESTI, 2020; MOHAMED; EL-GAYAR, 2021).

Training a time series model involves four phases: data acquisition, preprocessing, model training, and prediction.

Data acquisition collects historical data on the behaviour to be modelled, e.g., the CPU usage, memory or data traffic. As mentioned at the beginning of Section 2.3 explained, the collected data must be organised as a time series. Figure 2.3 shows Nasdaq Stock data organised as a time series.

Preprocessing prepares the data to train the learning algorithm by performing two tasks: data scaling and restructuring. The scaling modifies the data values in a standardised way for smaller ranges. It is essential for the convergence of models, avoiding issues like unstable gradients during training (ZHANG; PATUWO; HU, 1998) and preventing performance drops in scale-sensitive algorithms such as Support Vector Machine (SVM) (SHALABI; SHAABAN; KASASBEH, 2006). Commonly applied scaling algorithms include min-max normalisation, z-

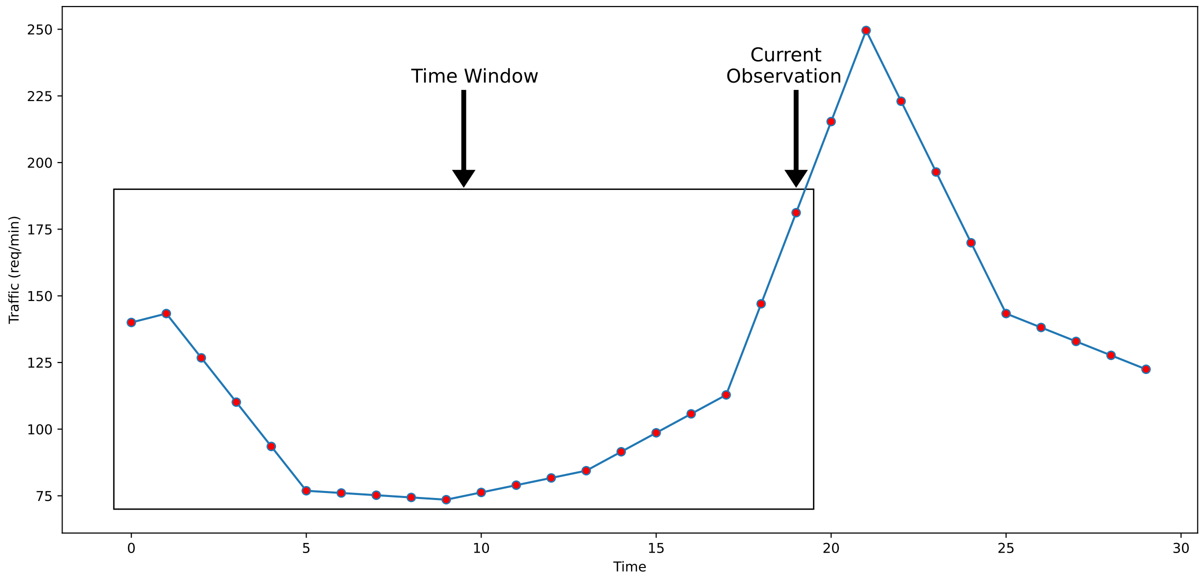
Figure 2.3 – Nasdaq stock series.



score normalisation and decimal scaling (SHALABI; SHAABAN; KASASBEH, 2006).

Next, the scaled data must be restructured as a supervised learning problem. An alternative for restructuring is the sliding window method (YU et al., 2014), which transforms the time series into fixed-size sliding windows. The fixed size is equivalent to the time series lags. Lags, or past observations of the time series, are typically the most important data for predicting the current observation. For instance, computer sales in March may be related to sales in February. Figure 2.4 shows a time window composed of 20 lags.

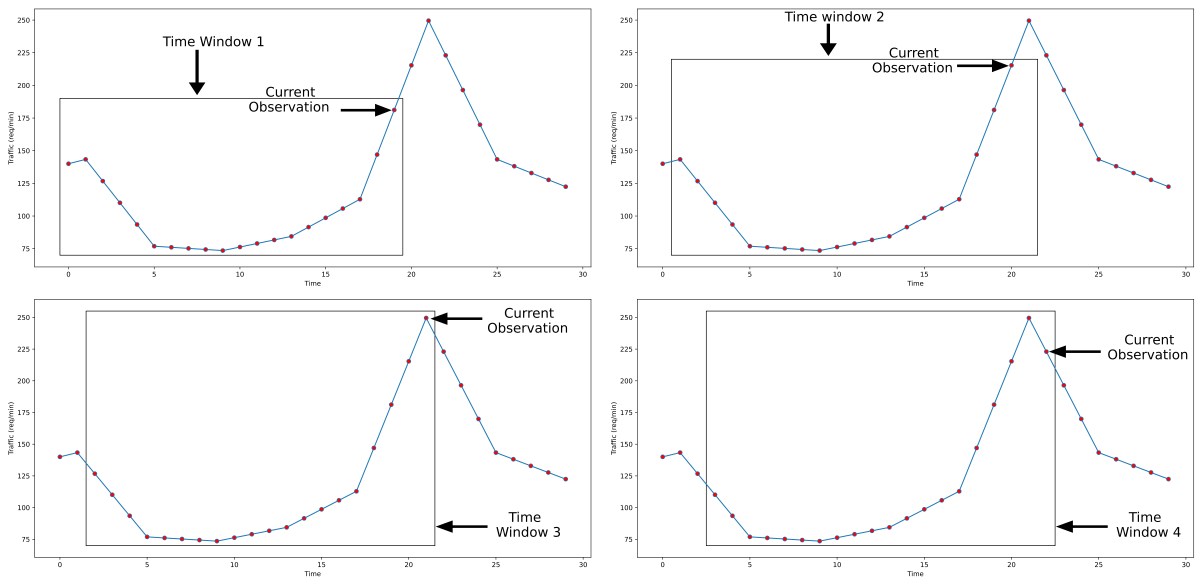
Figure 2.4 – Time window with 20 lags.



The auto-correlation function (SCARGLE, 1989) or optimisation algorithms (Ribeiro et al.,

2011) are alternatives for defining lags. The example of selling computers involves continuous lags. However, there may be cases where the lags are not continuous. For instance, the number of scientific publications in Brazil during July may be related to the one in May and March. Therefore, although June and April are temporarily close to July, they do not correlate. After selecting the lags, the entire time series is transformed into sliding windows, as illustrated in Figure 2.5.

Figure 2.5 – Sliding window method.



The preprocessed data is typically divided into three samples: training, validation, and testing (KUMAR; SINGH, 2018; WONG, 2018). The training sample is used as input to the training algorithm. The validation sample validates the model hyper-parameters. The testing sample validates the trained model, as it was not exposed to the algorithm during training. Some approaches in the literature use only two samples, i.e., training and testing without validation (NIKRAVESH; AJILA; LUNG, 2017).

The data splitting helps to detect the problem of model over-fitting. Over-fitting is a common issue where a model has high accuracy in predicting data used during training (training and validation samples) but lower accuracy in predicting new data, i.e., testing samples. The most common percentages used for splitting data into training and testing are 60% and 40% (AJILA; BANKOLE, 2016; NIKRAVESH; AJILA; LUNG, 2017; KUMAR; SINGH, 2018) or 80% and 20% (ALIPOUR; LIU, 2017). For three samples, the split is typically 60%, 20%, and 20% (WONG, 2018). After preprocessing, samples are ready for algorithm training.

Model training involves creating a model that accurately represents the behaviour of a time series. Various learning algorithms exist in the literature, each utilising different training approaches. However, they all aim to maximise model behaviour representation of the time series by performing an iterative training process.

The iterative training process can be performed in several ways, including the grid search method (CHANG; LIN, 2011). The grid search steps are defined as follows:

1. Choose a set of parameters;
2. Select a subset of these parameters for training;
3. The training sample is used to train the model;
4. The validation sample is used in the trained model;
5. An accuracy metric evaluates the forecasting of Step 4;
6. Repeat Steps 2 through 5 until there are no more parameters;
7. Choose the set of parameters with the highest forecast accuracy.

Step 5 involves calculating the forecasting accuracy, which measures the difference between the forecast/classification and its actual value. Many accuracy metrics are available, such as Root Mean Square Error (RMSE), Symmetric Mean Absolute Percentage Error (SMAPE), Average Relative Variance (ARV) or Mean Absolute Error (MAE) (WILLMOTT et al., 1985; AHMED et al., 2010; SILVA et al., 2018).

Finally, the model can make forecasts/classifications in the prediction step.

2.4 MULTIPLE PREDICTOR SYSTEMS

Multiple Predictor Systems (MPS), also called *ensemble*, have been commonly employed for TSF. The basic idea of the ensemble is to combine the strengths of different learning algorithms to build a more accurate and reliable forecasting system (QIU et al., 2017). As reported in (WIDODO; BUDI, 2011; KOURENTZES; BARROW; CRONE, 2014; ADHIKARI; VERMA; KHANDELWAL, 2015; MOURA; CAVALCANTI; OLIVEIRA, 2021), MPS obtains better accuracy than approaches based on only one predictor. MPS encompass three phases: Generation, Selection and Integration. In the Generation phase, a pool of forecasting models is trained. The pool is called homogeneous when a single learning algorithm trains all models; otherwise, it is heterogeneous. In the second phase, one or more models from the pool are selected.

Lastly, the Integration phase provides the final system forecast by combining forecasts from the different models chosen earlier. Chapter 6 provides more details about these phases.

2.5 CONCLUDING REMARKS

This chapter provided an essential overview to understand the proposal presented in this thesis. Initially, basic concepts about self-adaptive tools, emphasising the MAPE-K architecture, were presented. Then, MBAs, VMs, containers, and CMTs were discussed. Next, time series concepts and steps related to the forecasting process were described. Finally, an introduction to MPS was provided.

3 PROACTIVE MICROSERVICES AUTO-SCALER

This chapter introduces the PMA in detail. Section 3.1 overviews PMA and the Managed System. Next, Sections 3.2, 3.3, 3.4 and 3.5 present the Monitor, Analyser, Planner and Executor, respectively.

3.1 OVERVIEW

PMA (**P**roactive **M**icroservices **A**uto-scaler)[1] is a generic, holistic and adaptive proactive tool that applies horizontal auto-scaling to manage microservices. PMA uses forecasting models to predict microservice performance metrics at run-time. The estimated performance metrics are used to decide whether an adaptation is necessary. If needed, the adaptation consists of executing scaling-in/scaling-out actions. PMA has three forecasting strategies: univariate, multivariate, and Multiple Predictor Systems (MPS).

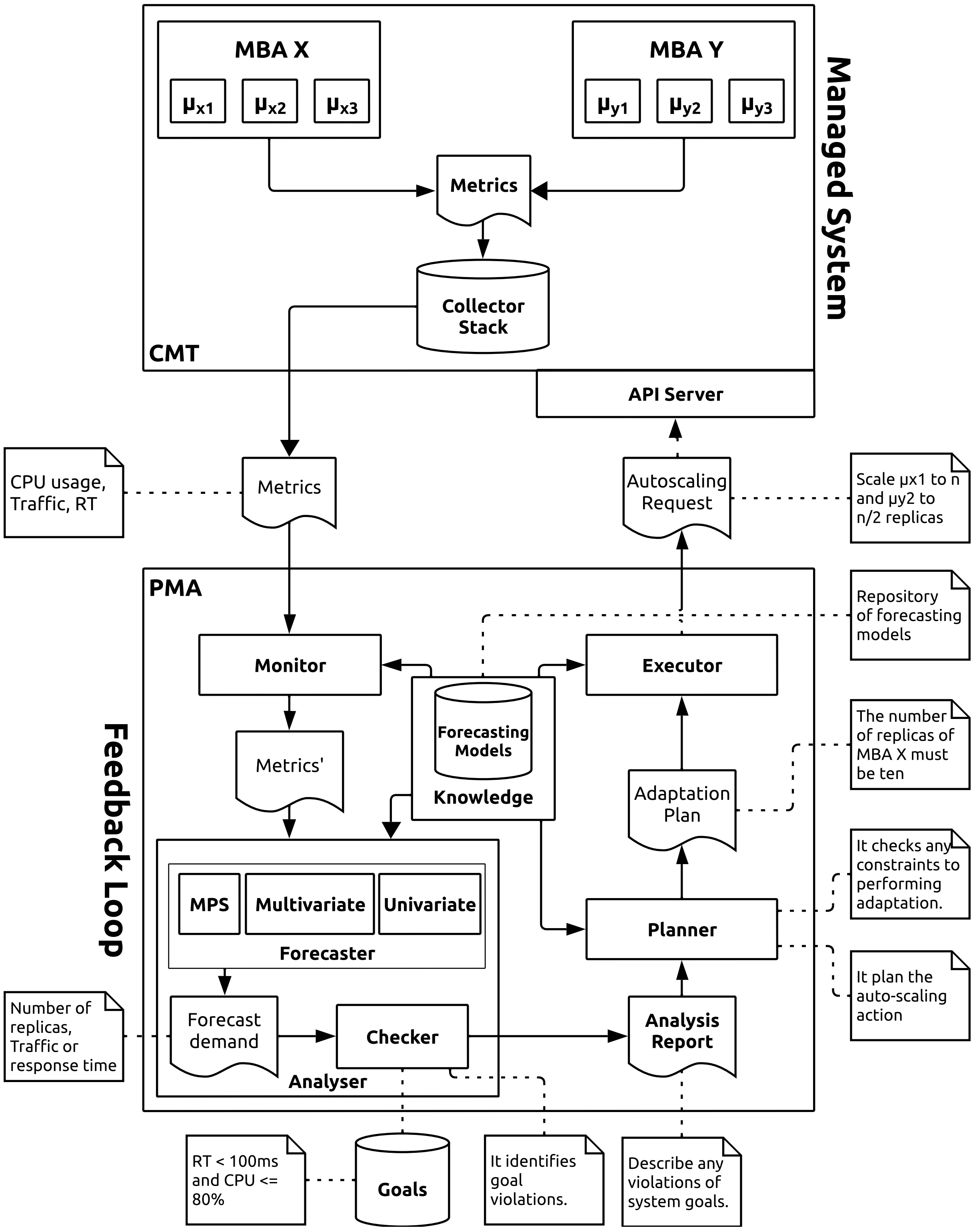
Figure 3.1 provides an overview of the PMA and the Managed System. The *Managed System* consists of applications, denoted as *MBA X* and *MBA Y* executed in a *Container Management Tool (CMT)* like Kubernetes or Docker. As mentioned in Section 2.2, each Microservice-based Application (MBA) has a set of microservices, denoted as μ_{x1}, μ_{x2}, and μ_{x3}, whose execution generates performance metrics including CPU usage, memory, response time, traffic. These metrics are stored inside the *Collector Stack*.

The *Knowledge base* is a passive component for storing data such as *forecasting models* and *PMA settings*, e.g., microservices to be adapted and performance metrics to be collected. The *Monitor* fetches the performance metrics, preprocesss them, and makes them available to other adaptive components. The *Analyser* uses the preprocessed performance metrics as input to forecast demand using forecasting models. The *Analyser* can employ three distinct forecasting strategies: *univariate*, *multivariate*, and *MPS*. However, it can only operate with one strategy at a time. The Forecaster predicts the demand for each microservice and stores the outcome in a *Metric Forecast* report. The *Checker* inspects the Metric Forecast, identifies goal violations and generates an *Analysis Report*. For instance, microservice μ_{x1} will demand 90% of CPU usage, violating the threshold of 80%.

The *Planner* reviews the Analysis Report and determines the best action to restore the managed system. The Planner also has strategies to dump actions that may affect the MBA.

[1] PMA source-code is publicly available on the GitHub repository: https://github.com/gfads/PMA

Figure 3.1 – General overview of PMA.


MBA X
μx1
μx2
μx3
MBA Y
μy1
μy2
μy3
Managed System
Metrics
Collector Stack
CMT
API Server
CPU usage, Traffic, RT
Metrics
Autoscaling Request
Scale μx1 to n and μy2 to n/2 replicas
PMA
Repository of forecasting models
Monitor
Executor
Forecasting Models
Knowledge
Feedback Loop
Metrics'
Adaptation Plan
The number of replicas of MBA X must be ten
MPS
Multivariate
Univariate
Forecaster
It checks any constraints to performing adaptation.
Planner
It plan the auto-scaling action
Number of replicas, Traffic or response time
Forecast demand
Checker
Analysis Report
Analyser
RT < 100ms and CPU <= 80%
Goals
It identifies goal violations.
Describe any violations of system goals.

For instance, it will block an action identical to a previous one executed within a short time to ensure the stability of the MBA. The Planner outcome is an *Adaptation Plan* that includes scaling in/out commands. Finally, the *Executor* performs the commands using the CMT aid.

The following sections present components and details that make up PMA.

3.2 MONITOR

The Monitor fetches and processes performance metrics from the Collector Stack and makes them available to other feedback loop components. The following monitoring parameters are configurable:

- **Manageable microservices.** The PMA configurator can customise it to auto-scale all or only certain microservices.
- **Performance metrics.** The PMA configurator can choose which performance metrics to use for auto-scaling. PMA can use any metrics stored within the Collector Stack, e.g., CPU usage, response time or the number of replicas per service.
- **Fetch sample format.** PMA needs different metrics structures for auto-scaling. The Monitor provides two formats: aggregate and individual. The individual metric is the current performance metric value. For instance, μ_{x1} microservice runs with three replicas. The aggregate metric is a set of observations of a performance metric, similar to a time series (see Section 2.3), e.g., CPU usage in the last 10 minutes collected every minute. As a result, the aggregated metric requires a historical interval, such as 10 minutes, and periodicity, e.g., 1 minute. PMA uses individual metrics in Planner for demand calculation and aggregated metrics in Forecast Demand to predict the performance metrics.
- **Waiting time.** The time between monitoring cycles.

3.3 ANALYSER

The Analyser aims to proactively identify whether the allocated resources, such as the number of replicas and other performance metrics, are adequate to satisfy the estimated demand of each microservice. The Analyser is composed of two modules: Forecaster Demand and Checker. The Forecast Demand predicts the performance metric(s) of the services, e.g., estimated response time in the next 5 minutes. The Checker reviews whether the estimated performance metric(s) violates managed system objectives, e.g., CPU usage at 90%, but the goal is 80%.

The following sections present the components of Analyser in detail.

3.3.1 Forecaster

The Forecaster anticipates the performance metric(s) demand of each service. It can estimate demand using three forecasting strategies: univariate, multivariate and MPS.

The univariate strategy estimates are based on historical data from a single performance metric, e.g., traffic data. On the other hand, the multivariate prediction uses multiple performance metrics for the same purpose, e.g., a combination of response code, request URL, and number of bytes sent. The MPS employs multiple models, and each model can be trained using either the univariate or multivariate strategy. The main idea of MPS is to increase the accuracy and reliability of predictions by combining various forecasting models. By doing so, forecasting is delegated to multiple models, making the self-adaptive tool less dependent on a single model.

As presented in Section 2.3.1, four steps must be followed to train forecast models. After the models are trained, the Forecaster uses the processed historical data from the Monitor as input to estimate the performance metric. For instance, the CPU usage or the number of pods in the next 5 minutes. PMA is highly configurable, allowing any performance metrics. Furthermore, any approach can be used to train forecasting models as long as the fundamentals of the strategies are maintained (see Sections 2.3 and 2.3.1).

This work adopted *Time Series Forecasting* (TSF) as a training approach due to its popularity for proactive auto-scaling tools of microservices (GOLI. et al., 2021; DANG-QUANG; YOO, 2022; KAKADE et al., 2023). For each microservice to be adopted, PMA requires one (univariate and multivariate) or multiple (MPS) offline-trained models.

3.3.2 Checker

The Checker verifies whether the estimated demand of each microservice violates managed system goals. The Checker uses a Ratio (R) between the predicted and expected managed system values to detect violations, similar to the approach proposed by the Horizontal Pod Auto-scaler (HPA) (see Section 2.2.1.2). Ratio (R) is defined in Equation 3.1:

$$R = \frac{FV}{DV}, \tag{3.1}$$

where *FV* is the future forecasting resource demand, while *DV* is the defined resource demand goal for the microservice.

R is used to classify the state of the microservice. The Checker uses an interval rather than relying on a fixed value set by the PMA configurator to minimise the occurrence of false positives. Therefore, Checker has upper and lower limits to a managed system goal, e.g., a response time of 80ms has a lower limit of 70ms and an upper limit of 90ms. More replicas are needed if R $>$ upper limit. On the other hand, if the lower limit $\leq$ R $\leq$ upper limit, then the microservice has the ideal number of replicas. Lastly, if the R $>$ lower limit, the microservice has more replicas than required.

3.4 PLANNER

The *Planner* determines which adaptive actions should be applied in the MBA. It can propose horizontal auto-scaling actions, e.g., scaling in/out of replicas. Actions are taken to address violations found during analysis, such as microservices with too few or too many replicas. Creating additional replicas is required to maintain the defined goal if the number of replicas is low. Conversely, optimisation of resources requires adaptation when there are extra replicas.

The planner calculates the number of replicas per microservice to meet future demand, following the HPA's approach (see Section 2.2.1.2). Equation 3.2 defines the HPA:

$$\mathrm{RR} = \lceil \mathrm{NCP} \times \frac{\mathrm{CMV}}{\mathrm{DMV}} \rceil, \tag{3.2}$$

where RR is the number of required replicas to meet the current demand, NCP is the number of current pods deployed, and CMV and DVM are the current and desired metric values, respectively.

The Planner also implements a module to prevent successive adaptations and ensure the managed system stability. Its primary purpose is to avoid making identical changes to the same microservice within a short period. This module is critical because each adaptation can add extra strain on the managed system. If not handled properly, this could result in a continuous cycle of adjustments, making the original issue even worse.

The module utilises a structure similar to an etcd[2] key-value store to maintain the status of adaptations. Each microservice has an entry, and within each entry, the last adaptation

[2] https://etcd.io/

action, such as scaling-in and scaling-out, along with their respective timestamps, is recorded. Before triggering a new adaptation, the records serve as a reference point for the Planner that checks the timestamp of the last occurrence of the specific adaptation for the microservice in question. If this previous adaptation occurred within a predefined cool-down period, the Planner blocks the new adaptation.

The cool-down is the minimum time interval required between two identical adaptations. Each auto-scaling action can have a designated cool-down period. For instance, the auto-scaling tool may require a 5-minute cool-down period for scaling-in and a 4-minute cool-down period for scaling-out. Likewise, the module also manages each microservice and adaptive action individually. For example, *MBA X* A has been blocked for 2 minutes after a scaling-in and 4 minutes after scaling-out adaptations. At the same time, *MBA Y* has been blocked for 3 minutes after a scaling-out action.

3.5 EXECUTOR

The Executor translates and carries out adaptation plans created by the Planner. An *adaptation plan* is a high-level command sequence that needs to be translated into ones recognised by a CMT, such as Kubernetes and Docker Swarm. After the translation, the Executor uses the *Invoker* to send requests with low-level commands to the CMT Management Application Programming Interface (API).

The Management API changes the CMT by modifying the number of replicas of the microservice depending on the scaling action. The Management API also informs the Executor whether the command execution was successful. Figure 3.2 shows process details inside the Executor.

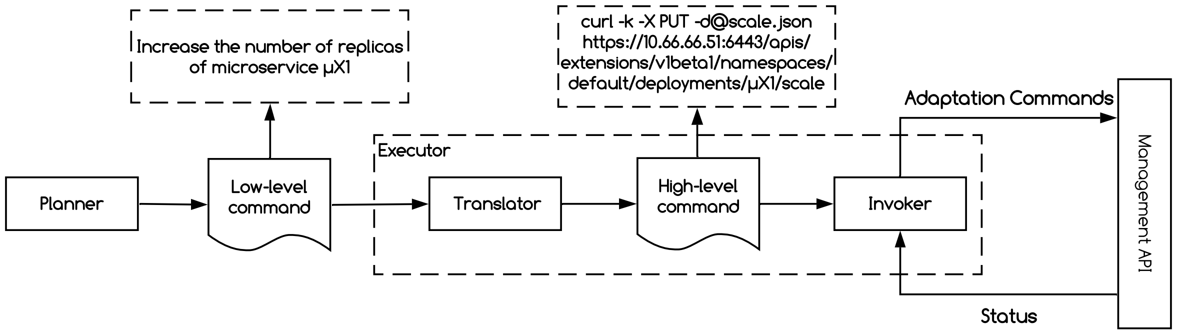

Increase the number of replicas of microservice μX1
curl -k -X PUT -d@scale.json https://10.66.66.51:6443/apis/extensions/v1beta1/namespaces/default/deployments/μX1/scale
Adaptation Commands
Executor
Planner
Low-level command
Translator
High-level command
Invoker
Management API
Status

Figure 3.2 – Internal processes of the Executor.

3.6 CONCLUDING REMARKS

This chapter began with an overview of PMA. Then, it outlined its phases, including the Monitor, Analyser, Planner, and Executor. Likewise, the operational process between these components was presented. It involves monitoring, forecasting, analysing performance metrics, and planning and executing scale adaptations based on these forecasts.

4 LEARNING ALGORITHMS COMPARATIVE ANALYSIS

This chapter presents a comparative analysis of different classes of learning algorithms for univariate forecasting of microservices performance metrics. Section 4.1 explains how this chapter is related to PMA. Section 4.2 states and motivates the problem the comparative analysis solves. Section 4.3 details the experimental setup used. Section 4.4 presented an experimental evaluation to assess the performance of the different learning algorithms. Section 4.5 covers the outcomes and results of the study.

4.1 CONTEXT

The PMA (**P**roactive **M**icroservices **A**uto-scaler) has an Analyser phase, which uses three prediction strategies to assist decision-making. One of these strategies is the univariate approach, which involves independently using individual performance metrics and their predictive capabilities for auto-scaling microservices. The comparative analysis presented in this chapter serves as a precursor to implementing the PMA univariate outlined in Chapter 3.

While past research has explored univariate approaches (PODOLSKIY et al., 2018; TOKA et al., 2021; GOLI. et al., 2021), there has yet to be a consensus on the predictive algorithms for microservices, including their forecasting accuracy and their prediction and training costs.

This analysis seeks to establish a solid foundation for the subsequent implementation of univariate predictive modelling techniques within the PMA. It aims to verify that the selected algorithms work effectively and can identify both positive and negative factors that impact predictive modelling results. As a result, this thesis is expected to improve the univariate forecasting process, allowing it to produce more accurate forecasts and better support data-driven decision-making.

4.2 PROBLEM STATEMENT

Time Series Forecasting (TSF) is a dominant approach to predict performance degradation in proactive auto-scaling of microservices (KANG; LAMA, 2020; YADAV; Rohit; YADAV, 2021). Typically, there are three main classes of prediction algorithms employed for TSF: Machine Learning (ML) (GOLI. et al., 2021), Deep Learning (DL) (COULSON; SOTIRIADIS; BESSIS,

2020) and statistical algorithms (YADAV; Rohit; YADAV, 2021). Some approaches employ only a single learning algorithm class, e.g., ML (ALIPOUR; LIU, 2017; GOLI. et al., 2021) or statistical (ALIPOUR; LIU, 2017; ROSSI; CARDELLINI; PRESTI, 2020; Fontana de Nardin et al., 2021; GALANTINO et al., 2021). At the same time, others use algorithms from different classes and pick up the one with the highest forecast accuracy, an approach denoted as the Classical Forecasting Approach (CFA), e.g., ML and DL (COULSON; SOTIRIADIS; BESSIS, 2020; MOHAMED; EL-GAYAR, 2021), DL and statistical (DANG-QUANG; YOO, 2021; WANG, 2021), statistical and ML (PODOLSKIY et al., 2018; KANG; LAMA, 2020; YADAV; Rohit; YADAV, 2021) and statistical, ML and DL (HUANG et al., 2021; TOKA et al., 2021).

However, most previous works claim their algorithms are the silver bullet for the proactive auto-scaling of microservices, but if different algorithms are deemed better, none are indeed the one. Also, these approaches usually demonstrate at least one of the following weak points:

1. Poor justification of the selected algorithms (ALIPOUR; LIU, 2017; PODOLSKIY et al., 2018; GALANTINO et al., 2021; MOHAMED; EL-GAYAR, 2021);

2. Limited evaluation by not contemplating the main classes of prediction algorithms or employing a short list of algorithms (YADAV; Rohit; YADAV, 2021; GOLI. et al., 2021; DANG-QUANG; YOO, 2021);

3. Only considering specific microservices performance metrics (ALIPOUR; LIU, 2017; PODOLSKIY et al., 2018; ROSSI; CARDELLINI; PRESTI, 2020; MARIE-MAGDELAINE; AHMED, 2020; Fontana de Nardin et al., 2021; DANG-QUANG; YOO, 2021; TOKA et al., 2021);

4. Only focused on forecast model accuracy, disregarding model fit and forecasting times (COULSON; SOTIRIADIS; BESSIS, 2020; ROSSI; CARDELLINI; PRESTI, 2020; TOKA et al., 2021), crucial in auto-scaling tools operating in pay-as-you-go environments as microservices;

5. Employ real-world restricted time series, focusing only on specific behavioural changes such as spikes in response time, or synthetic series (ALIPOUR; LIU, 2017; PODOLSKIY et al., 2018; COULSON; SOTIRIADIS; BESSIS, 2020; KANG; LAMA, 2020).

Furthermore, choosing the best learning algorithm from the vast range available, each with unique features that distinguish them in different time series can be a challenging task(SILVA et al., 2018).

This chapter deals with all these weaknesses to guide the selection of learning algorithms, streamlining the design of new adaptive microservice tools. It compares ten known learning algorithms from the three most prominent predicting classes, i.e., ML, DL and statistical. Also, it evaluates not only the accuracy but also considers the fit and forecasting time of the algorithms. Finally, 40 time series extracted from microservices operating in production in a large-scale deployment within the Alibaba Cluster (LUO et al., 2021) were used to evaluate the algorithms. These series consist of metrics commonly used to identify changes in the expected behaviour and trigger the microservices adaptation process (ALIPOUR; LIU, 2017; ROSSI; CARDELLINI; PRESTI, 2020; HUANG et al., 2021; MOHAMED; EL-GAYAR, 2021), namely CPU usage, memory, response time and traffic.

The following research questions guide this comparative analysis:

RQ4.1 - What are the most accurate algorithms for predicting microservices time series?

RQ4.2 - Does the accuracy of the algorithms depend on the microservice time series to be forecasted?

RQ4.3 - Which are the most accurate and faster models for fitting and forecasting microservices time series?

4.3 EXPERIMENTAL PROTOCOL

This section outlines the experimental protocol for comparing the three commonly adopted predicting classes for forecasting microservice performance metrics. The description covers the adopted datasets, learning algorithms, training procedures, evaluation metrics and statistical analysis.

4.3.1 Datasets

The experiments use real-world microservices time series collected from Alibaba production clusters[1] (LUO et al., 2021). This dataset was chosen because it comprises time series data from nearly twenty thousand microservices running over ten thousand bare-metal nodes during

[1] More detailed information about the Alibaba database is available in a public repository: https://github.com/alibaba/clusterdata/tree/master/cluster-trace-microservices-v2021

twelve hours in 2021. It includes performance metrics such as CPU usage, memory, response time and traffic.

All four metrics are widely adopted for proactive microservice auto-scaling (GALANTINO et al., 2021; MOHAMED; EL-GAYAR, 2021; YADAV; Rohit; YADAV, 2021). For each metric, a subset of the time series was randomly selected. Next, the ten least similar series from the subset were chosen according to the DTW algorithm (MORI; MENDIBURU; LOZANO, 2016; YOSHIDA; CHAKRABORTY, 2017). The selected traffic and response time series also cover different communication mechanisms among microservices: inter-process communication (STEEN; TANENBAUM, 2017), remote invocation (SRIRAMAN; WENISCH, 2018) and indirect communication (GAN et al., 2019a). CPU usage and memory series are composed of values measured every 30 seconds, resulting in an average of 1440 points. Traffic and response time series are composed of values measured every 1 minute, resulting in an average of 720 points per series. Appendix A provides a detailed description of the time series.

The min-max normalisation (SHALABI; SHAABAN; KASASBEH, 2006; de Amorim; CAVALCANTI; CRUZ, 2023) was adopted to scale the series to the interval [0, 1]. Equation 4.1 defines the min-max normalisation:

$$T_{norm} = \frac{T_{i-th} - T_{min}}{T_{max} - T_{min}} \tag{4.1}$$

where T_{i-th} is the original value of a time series observation; T_{min} and T_{max} are minimum/maximum values obtained from the time series; T_{norm} is the normalised value of T_{i-th}.

The series was also divided as follows: 60% points for training (7 hours and 12 minutes), 20% for validation (2 hours and 24 minutes) and the last 20% for testing (2 hours and 24 minutes) as done by Rangapuram et al. (2018), Wong (2018) and de Oliveira et al. (2020).

4.3.2 Parameters Setting

The comparative analysis investigated one statistical algorithm (AutoRegressive Integrated Moving Average (ARIMA) (PRYBUTOK; YI; MITCHELL, 2000)), five DL ones (Dual-Stage Attention-Based RNN (DARNN) (QIN et al., 2017), DeepAr (SALINAS et al., 2020), Deep State Space Model (DeepState) (RANGAPURAM et al., 2018), LSTM (MA et al., 2020) and TFT (LIM et al., 2021)) and four traditional ML ones (Multilayer Perceptron (MLP) (HAYKIN, 2001), Random Forest (RF) (BREIMAN, 1996), Support Vector Regressor (SVR) (DRUCKER et al.,

1997) and eXtreme Gradient Boosting (XGBoost) (CHEN; GUESTRIN, 2016)).

ML and ARIMA algorithms were selected because they are commonly used for forecasting microservices performance metrics (PODOLSKIY et al., 2018; PRACHITMUTITA et al., 2018; ROSSI; CARDELLINI; PRESTI, 2020; GOLI. et al., 2021; MOHAMED; EL-GAYAR, 2021; DANG-QUANG; YOO, 2021; YADAV; Rohit; YADAV, 2021). Also noteworthy is the variability of the ML learning techniques: a neural network, a support vector machine and two ensemble learning approaches. DL algorithms are also adopted for microservices performance metrics forecasting, but existing solutions mainly focus on Long Short-Term Memory (LSTM) (COULSON; SOTIRIADIS; BESSIS, 2020; DANG-QUANG; YOO, 2021; TOKA et al., 2021). Therefore, the DL subset was extended to cover other popular algorithms (DARNN (QIN et al., 2017), DeepAr (SALINAS et al., 2020), DeepState (RANGAPURAM et al., 2018), TFT (LIM et al., 2021)) as done by Elsayed et al. (2021). These algorithms were selected due to their higher forecast accuracies in the univariate series, which is the series category this chapter investigates. Temporal Fusion Transformer (TFT) (VASWANI et al., 2017) adoption is also noteworthy in this extended list as it is a Transformer-type learning algorithm that has not yet been employed for proactive auto-scaling of microservices.

The time series were organised utilising six time-sliding window lag sizes (L = {10, 20, 30, 40, 50 and 60}) selected using the *Autocorrelation Function* (ACF) (BOX et al., 2015). Due to the high number of hyper-parameter combinations, notably in algorithms like XGBoost with 6,561 combinations, only a subset of 60 random combinations is used per training, following the approach used by Lim et al. (2021).

4.3.3 Metrics

Mean Square Error (MSE) (Equation 4.2), a widely adopted competence measure, was used to assess forecast *model accuracy* (ARMSTRONG; COLLOPY, 1992; de Mattos Neto et al., 2014; OLIVEIRA; SILVA; NETO, 2022). MSE is defined in Equation 4.2:

$$MSE = \frac{1}{|T|}\sum_{i=1}^{T}(t_i - p(t_i))^2, \tag{4.2}$$

where t_i is the real value of the series, $p(t_i)$ is the value forecasted to the observation t_i by the model *p*, and $|T|$ is the number of observations.

As far as the authors know, no measure of competence currently computes model fitting

Table 4.1 – Hyper-parameters used in training models along with their source.

Algorithms	Hyper-parameters
ARIMA	Autoarima library
DARNN	‘enconder’: [16, 32, 64, 128, 256], ‘decoder’: [16, 32, 64, 128, 256] (QIN et al., 2017)
DeepAr	‘encoder’: [8], ‘decoder’: [8], ‘batch’: [64], ‘learning_rate’: [0.0001], ‘layers’: [3], ‘lstm_nodes’: [40] (SALINAS et al., 2020)
DeepState	The algorithm itself selects the hyper-parameters (RANGAPURAM et al., 2018)
LSTM	‘batch_size’: [64, 128], ‘epochs’: [1, 2, 4, 8, 10], ‘hidden_layers’: [2, 3, 4, 5, 6], ‘learning_rate’: [0.05, 0.01, 0.001], (COULSON; SOTIRIADIS; BESSIS, 2020)
MLP	‘hidden_layer_sizes’: [2, 5, 10, 15, 20], ‘activation’: [‘logistic’], ‘solver’: [‘adam’], ‘max_iter’: [1000], ‘num_exec’: 10 (OLIVEIRA; SILVA; NETO, 2022)
RF	‘min_samples_leaf’: [1, 5, 10], ‘min_samples_split’: [2, 5, 10, 15], ‘n_estimators’: [100, 500, 1000] (ESPINOSA et al., 2021)
TFT	‘dropout_rate’: [0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.9], ‘learning_rate’: [0.0001, 0.001, 0.01], ‘num_heads’: [1, 4], ‘batch’: [64, 128, 256] (LIM et al., 2021)
SVR	‘gamma’: [0.001, 0.01, 0.1, 1] ‘kernel’: [‘rbf’, ‘sigmoid’] ‘epsilon’: [0.1, 0.001, 0.0001] ‘C’: [0.1, 1, 10, 100, 1000, 10000] (OLIVEIRA; SILVA; NETO, 2022)
XGBoost	‘col_sample_by_tree’: [0.4, 0.6, 0.8], ‘gamma’: [1, 5, 10], ‘learning_rate’: [0.01, 0.1, 1], ‘max_depth’: [3, 6, 10], ‘n_estimators’: [100, 150, 200], ‘reg_alpha’: [0.01, 0.1, 10], ‘reg_lambda’: [0.01, 0.1, 10], ‘subsample’: [0.4, 0.6, 0.8] (PRASETYO; HOGANTARA; ISNAINIYAH, 2021)

and forecasting times. Therefore, this chapter proposes a new competence measure called MET to fill this gap. Model Effort Time (MET) is the average sum of the fitting and forecasting time. MET is defined in Equation 4.3:

$$MET = \frac{1}{NR}\sum_{i=1}^{NR}(\phi_i + \eta_i), \tag{4.3}$$

where ϕ_i and η_i are the fitting and forecasting times of the model in run i, and NR is the number of runs defined as 30.

MET is calculated using only the best model hyper-parameters per series. The grid search approach (BELETE; HUCHAIAH, 2022) time was not considered due to the varying number of hyper-parameter combinations among algorithms. Thus, those with smaller length combinations would benefit, e.g., RF. For both measures, lower values mean a better model.

4.3.4 Statistical Analysis

The non-parametric Friedman Test (FRIEDMAN, 1940) was used to compare the forecast accuracy performance of all algorithms over the series, as done by Mende & Koschke (2009), Cornejo-Bueno et al. (2019) and Mastelini & de Carvalho (2021). The Friedman test ranks the algorithms according to their forecast accuracy performance, i.e., the best-performing algorithm gets 1, the second-best gets 2, and so on. For algorithms which have the same forecasting accuracy performance, the average rank is assigned to them. The average rank of an algorithm is the average of its ranks across all series by metric.

The Nemenyi post-hoc test (NEMENYI, 1963) was applied to evaluate the average algorithm ranks similar to Graczyk et al. (2010). This test provides a pairwise comparison to report any significant difference between individual algorithms. Individual algorithms significantly differ if their average ranks exceed the Critical Difference (CD) defined in Equation 4.4.

$$CD = q\sqrt{\frac{|\Delta|(|\Delta|+1)}{6|\lambda|}}, \tag{4.4}$$

where $|\lambda|$ is the number of series, $|\Delta|$ denotes the number of algorithms, and q is a critical value.

The Nemenyi Test was applied with a 95% confidence level, and the CD diagram (DEMVSAR, 2006) was used to visualise its results. Horizontal lines in the diagram connect algorithms whose average ranks are lower than CD, i.e., no evidence was found to guarantee a significant difference between them.

4.4 RESULTS

This section compares different classes of learning algorithms for forecasting microservices performance metrics. Drawing on insights from Chapter 3, where the multivariate PMA module was introduced, the following analysis evaluated the performance of popular forecasting algorithms to forecast microservices performance metrics. By identifying the most effective predictive algorithms and clarifying their forecasting accuracy and predicting and fitting costs, this section aims to optimise the PMA univariate module. Establishing a robust foundation of predictive modelling techniques aids in refining the univariate forecasting process within the PMA, enabling more accurate and data-driven decision-making in microservice auto-scaling.

Section 4.4.2 assesses the MET for fitting and forecasting the series and correlates it with their accuracy. The comparative analysis source code, figures, datasets and detailed model accuracies by time series are available in a public repository[2].

4.4.1 Models Accuracy

Table 4.2 presents the MSE average and standard deviation accuracy of the models to forecast 40 different series (10 series per microservice metric). SVR obtained the best results (lowest MSE) for CPU usage, memory, and traffic series. DeepAr stands out in the response time series. Nevertheless, both algorithms alternate as best or second-best across all metrics. ARIMA, DARNN, and XGBoost repeatedly alternate as having the worst results.

Table 4.2 – Accuracy of the models by microservice performance metric. Mean and standard deviation results of the MSE competence measure. The best results are highlighted in bold, and the second-best results are underlined per microservice metric. Error-values are in the 10^{-3} scale.

Algorithms	CPU usage	Memory	Response time	Traffic
ARIMA	82.02 (74.74)	14.12 (20.93)	21.42 (11.68)	34.39 (37.51)
DARNN	107.1 (84.23)	57.75 (71.18)	32.42 (21.43)	42.79 (50.71)
DeepAr	<u>5.309</u> (5.920)	<u>0.924</u> (1.666)	**8.891** (5.781)	<u>6.734</u> (6.462)
DeepState	20.39 (40.35)	1.264 (2.197)	10.01 (5.279)	8.329 (6.319)
LSTM	16.83 (13.90)	1.662 (2.024)	15.71 (7.161)	9.904 (6.726)
TFT	5.629 (6.178)	0.928 (1.743)	12.68 (8.702)	9.789 (10.45)
MLP	9.350 (7.097)	2.991 (3.147)	9.936 (5.634)	7.952 (5.773)
RF	20.25 (27.76)	84.90 (101.7)	21.03 (26.17)	17.94 (20.31)
SVR	**3.232** (1.972)	**0.736** (1.661)	<u>9.117</u> (6.496)	**6.701** (6.445)
XGBoost	42.48 (50.75)	92.84 (98.70)	37.01 (32.27)	26.09 (22.89)

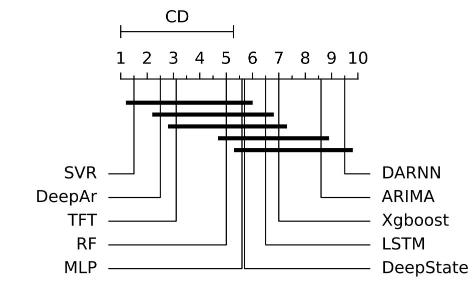
Some algorithms had better results only on specific metrics. For instance, DeepState is better for memory, response time, and traffic, TFT for CPU usage and memory, and LSTM for memory and traffic series. MLP had good results on all metrics but underperformed SVR and DeepAr. RF does not stand out for any metric, having worse results in memory forecasting.

The Friedman test indicated a statistical difference between the accuracy of the models. So, a pairwise comparison was performed using the Nemenyi post-hoc test, whose results are shown in Figure 4.1. The CD diagram shows no statistical difference between SVR, DeepAr, TFT, and DeepState accuracies across all metrics. Likewise, it confirms that ARIMA, XGBoost, and DARNN are statistically worse than SVR and DeepAr in most results except for DeepAr, which

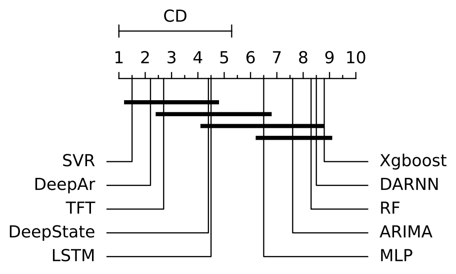
[2] https://github.com/gfads/comparative-analysis-microservices-prediction

is performed equally to ARIMA and XGBoost in traffic. Also, there is no statistical evidence that other algorithms were worse than SVR, DeepAr, TFT, and DeepState on specific metrics such as CPU usage (MLP and RF), memory (LSTM), response time and traffic (LSTM, MLP and RF).

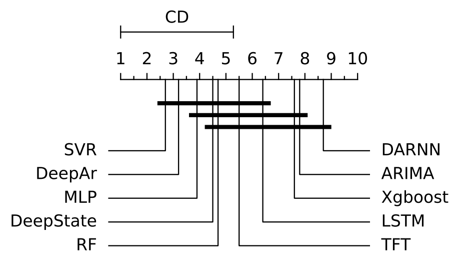
Figure 4.1 – The graphical representation of the Nemenyi post-hoc test results on the CD diagram. Each microservice metric has its CD diagram. Horizontal lines in the diagram connect algorithms whose average ranks are lower than CD, i.e., no evidence was found to guarantee a significant difference between them.



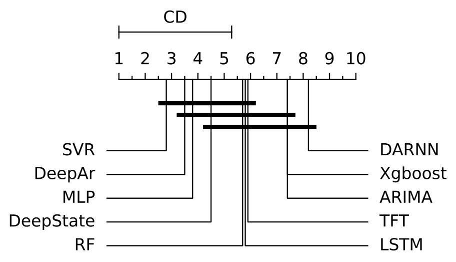
(a) CPU usage.



(b) Memory.



(c) Response time.



(d) Traffic.

Furthermore, these results show that the algorithms of both predicting classes (DL and ML) obtain high (SVR and DeepAR), medium (MLP and LSTM), and low (XGBoost and DARNN) forecast accuracies, i.e., choosing a specific predicting class does not guarantee better forecast accuracy. Hence, the results indicate a metric-dependent problem, and the algorithm choice decision must be carefully addressed during tool design.

Based on the accuracy obtained in the 40 analysed time series, Table 4.3 recommends the best learning algorithms for each microservice performance metric. The ✓ indicates whether the algorithm should be used per microservice series. Otherwise, X is used.

Considering the results, the **most accurate algorithms for forecasting microservices time series are DeepAr, DeepState, TFT and SVR**, answering 4.1. Three algorithms (DeepAR, DeepState, TFT) belong to the DL predicting class, and one, SVR, belongs to the traditional ML class. However, SVR accuracy deserves a highlight since DL algorithms tend to be overly complex compared to traditional techniques. The results also show that MLP is

Table 4.3 – A recommendable summary of the evaluated algorithms considering accuracy only. The choices were based on the statistical Friedman and the Nemenyi post-hoc tests. The ✓ means that the algorithm is recommended to forecast the metric. Otherwise, ✗ is used.

Algorithms	**CPU usage**	**Memory**	**Response time**	**Traffic**
ARIMA	✗	✗	✗	✗
DARNN	✗	✗	✗	✗
DeepAr	✓	✓	✓	✓
DeepState	✓	✓	✓	✓
LSTM	✗	✓	✓	✓
MLP	✓	✗	✓	✓
RF	✓	✗	✓	✓
SVR	✓	✓	✓	✓
TFT	✓	✓	✓	✓
XGBoost	✗	✗	✗	✗

more accurate in response time and traffic series, LSTM in memory and RF in CPU usage. Likewise, even the recommended algorithms are more prominent in specific series, e.g., TFT (CPU usage and memory) and DeepState (memory, response time and traffic). Therefore, it is possible to conclude that the **accuracy of the models depends on the time series to be predicted**, giving a positive answer to 4.2.

4.4.2 Model Effort Time

Although higher forecasting accuracy is essential for proactive auto-scaling tools, other factors can also affect their performance. Forecast time affects the reaction time of the adaptive tool to changes and fitting time affects its operation, e.g., a slow self-adaptive tool startup or unavailability due to lengthy model retraining. Thus, for real applications, both times are crucial in determining whether the adaptation execution takes place on time. Both times were summed and defined for analysis as MET in Equation 4.3.

Table 4.4 presents the average MET results of each algorithm by microservice performance metric. As expected, traditional ML algorithms are faster than ARIMA and DL due to their lower complexity. The MET in DL algorithms is broad, going from faster algorithms than ARIMA (DARNN and LSTM) to the two slowest evaluated (TFT and DeepState). The fastest algorithms per predicting classes are MLP and SVR in ML and DARNN and LSTM in DL. Finally, ARIMA has comparable MET to DeepAr and LSTM.

Considering MET and accuracy, **SVR is the best for forecasting microservice met-**

Table 4.4 – Average MET results of algorithms. The best results are highlighted in bold for each prediction algorithm class and microservice metric, while the second-best results are underlined. The MET value is in seconds.

Predicting Class	Algorithms	CPU usage	Memory	Response time	Traffic
Statistical	ARIMA	7.22	7.51	8.12	8.84
DL	DARNN	**2.56**	**2.66**	**2.79**	**2.92**
	DeepAr	9.23	9.22	9.22	9.22
	DeepState	92.31	92.34	92.38	92.45
	LSTM	<u>6.18</u>	<u>6.50</u>	<u>6.77</u>	<u>7.11</u>
	TFT	17.91	17.91	17.90	17.90
ML	MLP	**0.05**	**0.05**	**0.05**	**0.05**
	RF	0.97	1.06	0.96	0.94
	SVR	<u>0.12</u>	<u>0.13</u>	<u>0.13</u>	<u>0.15</u>
	XGBoost	0.20	0.20	0.22	0.24

rics, answering 4.3. Although SVR, DeepAR, DeepState, and TFT do not have significantly different accuracies, SVR stands out for its lower MET. DeepAr and TFT are also outstanding choices in an environment without strict time constraints. LSTM is another alternative, but only for forecasting specific metrics. DeepState is discouraged due to its lengthy MET and, consequently, its impact on a microservices execution environment. MLP and RF can also be adopted because they have good accuracy in microservices-specific metrics and do not have lengthy MET. Finally, ARIMA, DARNN, and XGBoost have low accuracy and should be avoided in series with behaviour similar to that of the Alibaba database.

Table 4.5 summarises the findings of the comparative study by associating the algorithms and microservice performance metrics considering MET and accuracy. The ✓ indicates that the algorithm accurately forecasts the microservice metric based on the Friedman statistical test. Otherwise, ✗ is used. The '*' indicates whether an algorithm has a MET above the median of those evaluated. Algorithms that were not accurate for any metric were omitted.

Table 4.5 – A final summary of the findings of the comparative analysis correlating the algorithms and microservice performance metrics evaluated. The ✓ indicates that the algorithm accurately forecasts the microservice metric based on the Friedman statistical and the Nemenyi post-hoc tests. Otherwise, ✗ is used. The * indicates whether an algorithm has a MET above the median of those evaluated. Algorithms that were not accurate for any metric were omitted.

Algorithms	**CPU usage**	**Memory**	**Response time**	**Traffic**
DeepAr*	✓	✓	✓	✓
LSTM*	✗	✓	✓	✓
MLP	✓	✗	✓	✓
RF	✓	✗	✓	✓
SVR	✓	✓	✓	✓
TFT*	✓	✓	✓	✓

4.5 DISCUSSION

The findings presented in this chapter contribute to the domain of proactive adaptation of microservices. First, the existing solutions commonly indicate that DL algorithms are more accurate for forecasting time series than traditional ML algorithms (PATERAKIS et al., 2017; MEHTAB; SEN, 2020; RAHIMZAD et al., 2021; KUMAR et al., 2022; BAMISILE et al., 2022). However, as microservices operate in pay-as-you-go environments, the cost and accuracy of forecasting models must be factors to consider. As demonstrated, traditional ML models are as accurate and have lower MET as DL ones for forecasting microservices time series. Therefore, service providers can save costs by adopting accurate traditional ML models in their adaptive proactive tools. This finding is similar to another recently reported by Elsayed et al. (2021), who also points out that traditional ML algorithms outperformed DL ones in time series from other domains, such as electricity, urban air quality, stock exchange and solar energy.

Second, although SVR and DeepAr are models with higher accuracy agnostic to the predicted metric type, most models tend to perform better on specific performance metrics. This trait of learning algorithms is a known forecasting problem (SILVA; NETO; CAVALCANTI, 2021), where it is understood that no forecast model is the best in all possible scenarios (YAO; DAI; SONG, 2019). Therefore, prediction mechanisms for adaptive tools need a solution to the single-model problem. Inspired by the problem described, a Multiple Predictor Systems (MPS) is proposed in Chapter 6 for building microservices auto-scaling tools using a multiple predictor approach.

The results of this analysis have important implications for the PMA, particularly in the strategy of univariate prediction. It shows that traditional ML models can achieve forecast accuracy similar to DL ones but with lower MET, making it a cost-effective strategy for use in PMA. A cost-effective strategy is essential for microservices environments where prediction accuracy and computational efficiency are crucial. Validating traditional ML models as effective predictors ensures that the PMA can maintain high forecast performance while optimising resource usage and reducing operational costs. These findings lay a solid foundation for the PMA univariate predictive strategy proposed in Chapter 3, enhancing its ability to make accurate, timely, and cost-efficient auto-scaling decisions based on individual performance metrics.

4.6 CONCLUDING REMARKS

This chapter presents a comparative analysis of different classes of learning algorithms for forecasting microservices performance metrics. Firstly, the problem solved by the comparative analysis is stated and motivated. Next, the experimental setup used is detailed. Then, an experimental evaluation is presented to assess the performance of the different learning algorithms. Finally, the outcomes were discussed in detail.

5 FORECASTING STRATEGIES COMPARATIVE STUDY

This chapter compares univariate and multivariate strategies for proactive auto-scaling of Microservice-based Applications (MBAs). Section 5.1 explains how this chapter is related to PMA. Section 5.2 provides the context for the problem and objectives of the comparative study. Section 5.3 details the experimental protocol applied. Section 5.4 presents an experimental evaluation to measure the performance of both strategies. Section 5.5 discusses the results.

5.1 CONTEXT

Univariate forecasting is a crucial strategy in time series analysis, which involves predicting future values based solely on past observations of a single performance metric, as demonstrated in Chapter 4. This strategy assumes that the past behaviour of a variable, such as the historical demand for a service, contains enough information to forecast its future behaviour. In microservices, where different components of an application are deployed and scaled independently, univariate forecasting can be used to predict the workload or resource needs of individual services based on their historical usage patterns. Despite its simplicity and ease of implementation, univariate forecasting has limitations, particularly when the interactions between different microservices or external factors significantly impact their performance.

The literature about forecasting methods for microservices also looks into multivariate prediction. This strategy assumes that the past behaviour of a specific performance metric, such as CPU usage, is influenced by the behaviour of other metrics, such as response time and memory. The multivariate approach is better at capturing complex dependencies and interactions among different services and external factors, potentially providing more accurate forecasts. However, existing studies often apply univariate and multivariate forecasting techniques and do not explain why they chose a particular strategy. Therefore, the current literature requires a thorough comparative study to identify when and why one strategy may outperform the other in forecasting the behaviour of microservices.

This chapter addresses this research gap by comparing univariate and multivariate forecasting techniques for microservices. It aims to evaluate the performance of each approach and establish the basis for creating a comprehensive, proactive, and adaptable forecasting tool. The forecasting tool proposed is the Multivariate Forecasting Tool (MFT), essentially the multivariate PMA module discussed in Chapter 3, with a different name.

5.2 PROBLEM STATEMENT

Existing solutions have used univariate and multivariate forecasting strategies for proactive auto-scaling of MBAs (COULSON; SOTIRIADIS; BESSIS, 2020; ROSSI; CARDELLINI; PRESTI, 2020). For example, some approaches use univariate strategies to predict response time (ROSSI; CARDELLINI; PRESTI, 2020), traffic (DANG-QUANG; YOO, 2022; TOKA et al., 2021), and CPU usage (WANG, 2021). Conversely, other approaches employ multivariate strategies to predict traffic (COULSON; SOTIRIADIS; BESSIS, 2020; GOLI. et al., 2021) and response time (MOHAMED; EL-GAYAR, 2021). Univariate strategies are more straightforward to implement than multivariate ones. However, as multivariate forecasting uses different features, it can capture complex dependencies and interactions among services and external factors, potentially providing more accurate forecasts. However, most solutions neither explain why they choose a particular forecasting strategy (ALIPOUR; LIU, 2017; GALANTINO et al., 2021; MOHAMED; EL-GAYAR, 2021; PODOLSKIY et al., 2018) nor consider comparing both alternatives. Also, the approaches proposed are typically evaluated in limited and artificial scenarios (COULSON; SOTIRIADIS; BESSIS, 2020; DANG-QUANG; YOO, 2021; ROSSI; CARDELLINI; PRESTI, 2020), which hampers the applicability of the findings.

Thus, although several approaches have embraced these strategies for adapting MBAs, little effort has been devoted to evaluating and understanding their performance at run-time. A comparative study can help the PMA configurator to choose the best PMA forecasting module according to their purpose and execution environment.

This chapter presents a comparative study evaluating univariate and multivariate proactive scaling of MBAs. The Predict Kube (PK) (see Section 2.2.2.2) is compared to a proposed custom-made proactive auto-scaling multivariate system named Multivariate Forecasting Tool (MFT). MFT is essentially the multivariate PMA module discussed in Chapter 3, with a different name. They were evaluated to adapt four popular open-source benchmark applications, namely Daytrader (DAYTRADER, 2024), Quarkus-HTTP-Demo (QHD) (QUARKUS-HTTP-DEMO, 2024), Online Boutique (OB) (ONLINE-BOUTIQUE, 2024) and Travels (TRAVELS, 2024). The experiments considered three forecasting horizons, namely, 1, 3, and 5 minutes.

The following research questions guide this comparative study:

RQ5.1 - Is an auto-scaler that adopts multivariate forecasting more efficient in managing MBAs than one that employs univariate forecasting?

RQ5.2 - Does the efficiency of the auto-scaler forecast strategy depend on the characteristics/traits of the managed application?

The next section presents details of the comparative study.

5.3 EXPERIMENTAL PROTOCOL

This section introduces the experimental protocol adopted to compare the *Time Series Forecasting* (TSF) strategies univariate and multivariate for the proactive auto-scaling of MBAs.

5.3.1 Setup

All experiments were conducted on a local K8s cluster of six nodes, comprising one master and five workers. Each node, running Ubuntu 22.04 LTS, is a Virtual Machine (VM) equipped with Intel® Xeon® CPU 1220 (four vCpus) and 16GB DDR3 RAM.

This comparative study considers MBAs as applications composed of one or more services running in a container in a K8s cluster. Four heterogeneous applications were used, namely Daytrader (DAYTRADER, 2024), OB (ONLINE-BOUTIQUE, 2024), QHD (QUARKUS-HTTP-DEMO, 2024) and Travels (TRAVELS, 2024). Daytrader and QHD are both Java applications[1]. Daytrader is an online stock trading system where users can log in, view their portfolios, look up stock quotes, and buy or sell stock shares. QHD is a simple CRUD (Create, Read, Update and Delete) REST application. Travels is a marketplace with six microservices written in Go, where users can search for and book flights, hotels, cars, or insurance. OB is an e-commerce platform with ten microservices written in various languages like Go, Java, Node.js, Python, and C#. Finally, each application has a database attached: DB2[2] (Daytrader), Redis[3] (OB), PostgreSQL[4] (QHD), and MySQL (Travels).

[1] https://www.java.com/
[2] https://www.ibm.com/analytics/db2
[3] https://redis.io/
[4] https://www.postgresql.org/

Several companies have adopted these applications to benchmark their operations. For example, IBM (International Business Machines) uses Daytrader to assess its products internally[5], while RedHat uses QHD to evaluate its Quarkus framework. Google showcases technologies such as K8s, gRPC, and Stackdriver with OB (ONLINE-BOUTIQUE, 2024), and Istio uses Travels as a showcase[6] for its service mesh.

5.3.2 Proactive Tools

Next are the details about the two proactive tools subjects to be compared in this section.

5.3.2.1 Predict Kube

PK[7] is a proactive adaptive tool that scales services in a K8s cluster (see Section 2.2.2.2). To activate the auto-scaling, the PK configurator must specify a performance metric, such as response time, and its desired value, i.e., response time must be less than 200ms. Furthermore, the PK configurator must set a forecast horizon and historical time window. The prediction horizon determines how far into the future the tool needs to estimate, e.g., application traffic in *ten minutes*. The historical time window refers to the minimum amount of past data the system needs to train its models, e.g., *three days* of data.

In operation, PK predicts the performance metric, and depending on whether it is above or below the set threshold, pods are added or removed to keep the metric as close as possible to the threshold. This approach is similar to the Horizontal Pod Auto-scaler (HPA)[8] but uses the forecast performance metric instead of its current value for decision-making (see section 2.2.1.2). PK trains a forecasting model for every microservice that needs to be adapted at run-time.

PK is not essentially a self-adaptive proactive univariate system. The developer may propose a score (combination of multiple metrics) and the desired threshold. However, proposing a score that accurately models an application in a dynamic distributed environment is challenging (SAMPAIO et al., 2023). Likewise, finding a score that applies to multiple applications can be challenging.

[5] https://www.ibm.com/docs/en/linux-on-systems?topic=bad-daytrader
[6] https://kiali.io/docs/tutorials/travels/
[7] https://dysnix.com/predictkube
[8] https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

It is worth observing that details about the training procedures of PK, including the data set split, training parameters or time window size, and algorithms used, are not publicly available.

5.3.2.2 Multivariate Forecasting Tool

Nowadays, there are no industry tools for proactively auto-scaling MBAs using a multivariate strategy. Existing literature solutions often do not adhere to proper building standards, such as MAPE-K (COULSON; SOTIRIADIS; BESSIS, 2020; HUANG et al., 2021). Also, some are designed for specific adaptation scenarios, like K8s (DANG-QUANG; YOO, 2022), while others are limited to particular forecasting models, such as Machine Learning (ML) (GOLI. et al., 2021). Motivated by these shortcomings and inspired by their architectures, a generic, holistic and custom-made tool was developed for comparison purposes.

Multivariate Forecasting Tool (MFT) is a proactive auto-scaler written in Python (PYTHON, 2024), integrated with K8s through the K8s API (KUBERNETES, 2024b) and uses Prometheus as a database. An offline-trained multivariate model per microservice is required to deploy the auto-scaler. MFT does not define predefined features to train the model. Each independent variable in the data set can be used as either input or target for the model. MFT models the correlations among features within the time series, resulting in a black box function that captures the behaviour of the system, abstracting the *PK score formulation*. The source code for MFT is available in an open-source repository[9].

Training Features. The evaluated applications provide different metrics (features) to be collected and utilised for model training. For example, Daytrader and QHD provide performance metrics like CPU usage and memory, applications metrics such as response time and traffic, JVM information including as heap memory, and connection pool info such as database connections. Meanwhile, OB and Travels provide only performance metrics, e.g., CPU usage and memory. To broaden the availability of OB and Travels metrics, K8s has been expanded with the Istio service mesh (ISTIO, 2024). Table 5.1 summarises the features used for training the model of each application.

Model Training. The series was scaled to the interval [0, 1] using min-max normalisation (SHALABI; SHAABAN; KASASBEH, 2006), defined in Equation 4.1. The series was divided as follows: 70% points for training and the last 30% for testing, as done by Alipour & Liu (2017) and Yadav, Rohit & Yadav (2021). According to the configured forecast horizon, the series

[9] https://github.com/gfads/univariate-multivariate-prediction

Table 5.1 – Features used for training MFT models per application.

Application	Features	Target
Daytrader	ACP, C, DCP, HM, M, NP, PCGS, RT, T, TP, TCGS, GJVMO	NP
QHD	C, DCP, HM, M, NP, RT, T, PCGS, TCGS, GJVMO	NP
OB	C, M, ERT, ET	NP
Travels	C, M, ERT, ET	NP

Features: Application Connection Pool (ACP), CPU Usage (C), Database Connection Pool (DCP), Envoy Response Time (ERT), Envoy Traffic (ET), Global JVM Operations (GJVMO), Heap Memory (HM), Memory (M), Number of Pods (NP), Per-second GC scavenge (PCGS), Response Time (RT), Thread Pool (TP), Traffic (T), Total GC Scavenge (TCGS).

was also organised into different lag sizes, with 20 lags for 1 minute, 7 for 3 minutes and 4 for 5 minutes. MFT adopts Long Short-Term Memory (LSTM) for multivariate forecasting because it is widely employed to microservices (COULSON; SOTIRIADIS; BESSIS, 2020; MARIE-MAGDELAINE; AHMED, 2020; TOKA et al., 2021; DANG-QUANG; YOO, 2022; WANG, 2021). LSTM were trained using a grid search approach to find the best hyper-parameters per model. Table 5.2 summarises the hyper-parameters used for LSTM training.

Table 5.2 – Hyper-parameters for LSTM training.

Parameters[a]	Values
Batches	64, 128
Epochs	1, 2, 4, 8, 10
Learning Rate	Adam=[0.05, 0.01, 0.001]
Units	50, 75, 125
Hidden layers	2, 3, 4, 5, 6

[a]The hyper-parameters used were slightly adapted from Coulson, Sotiriadis & Bessis (2020).

5.3.3 Metrics

The efficiency of an adaptive MBA is commonly measured by its ability to ensure customer experience, such as meeting Service Level Agreement (SLA) requirements, as well as its cost-effectiveness. Therefore, the quality measures proposed by Straesser et al. (2022), which consider cost-effectiveness and customer experience, were slightly adapted and used in the comparative evaluation of MFT and PK. This thesis adopts response time as a customer experience metric instead of the failures adopted by Straesser et al..

The total cost (C) of an experiment is the sum of all pods allocated during it. C is defined in Equation 5.1.

$$C = \sum_{\mu \in M} \sum_{\nu \in N} AN_{\nu}(\mu), \tag{5.1}$$

where M is a set of all microservices of an application that needs to be scaled, N is a set of all measurement intervals in the experiment, and $AN_{\nu}(\mu)$ is the absolute number of deployed instances of μ in the measurement interval ν.

The response time percentiles (90^{th}, 95^{th} and 99^{th}) are an alternative to measuring customer experience. The 95^{th} and 99^{th} percentiles are commonly used in software applications and Web Services (DING et al., 2019). However, the 99^{th} percentile may not be proper if there are outliers in the response time sample generated by external factors to the application. On the other hand, 95^{th} is more flexible and can include outliers and genuinely slow requests, making it a more reliable measure for assessing customer experience. Therefore, this thesis considered the 95^{th} percentile (RT) to evaluate the customer experience, called hereafter *response time*.

The auto-scaling efficiency (AE_w) metric is defined as a weighted average of response time and cost (STRAESSER et al., 2022). AE_w is defined in Equation 5.2.

$$AE_w = \omega \frac{RT}{RT_{max}} + (1 - \omega) \frac{C}{C_{max}}, \tag{5.2}$$

where ω is an adjustable weight for the desired costs and the response time ratio. RT is the 95^{th} response time, and C is the total cost. C_{max} is a normalisation factor that maps the costs to a scale between 0 and 1. C_{max} is defined in Equation 5.3:

$$C_{max} = M \times N \times AN_{max}, \tag{5.3}$$

where M is the number of microservices of an application, N is the number of measured intervals, and AN_{max} is the maximum number of pods allocated by a microservice considering all its experiments. In short, C_{max} would be equivalent to the cost C if a service M had AN_{max} instances deployed throughout the experiment. RT_{max} is a normalisation factor that maps the response time to a scale between 0 and 1. RT_{max} is defined in Equation 5.4:

$$RT_{max} = \max\{rt_1, rt_2, ..., rt_n\}, \tag{5.4}$$

where rt_1, rt_2, and rt_n are response times of n experiments for a given microservice $\mu \in M$. In short, RT_{max} is the worst response time observed in a given application.

Some scenarios can prioritise the operation costs or response time differently, and ω (adjustable weight) offers the capacity to adjust accordingly. For example, a lower response time is commonly prioritised over reducing the costs of customer-oriented business applications. The comparative evaluation considered $\omega = 0.5$ ($AE_{0.5}$), which weighted cost and response time equally. For all evaluation measures, lower values mean better results.

5.3.4 Workloads

Periodic workloads were used in the experiments. These workloads are commonly observed in Web applications (ABDULLAH et al., 2021; IQBAL; ERRADI; MAHMOOD, 2018). Appendix B provides more details on periodic workloads.

Periodic workloads are modelled using the mathematical sine function, as referenced in Abdullah et al. (2021). The sine function is defined in Equation 5.5:

$$\chi_{\nu_i} = \chi_{\nu_0} - \alpha sin(\frac{2i\pi}{\gamma}) + s_c Rand(\mathbb{N}(0, 1)), \tag{5.5}$$

where χ_{ν_i} is the number of requests in the time interval ν_i, χ_{ν_0} is the number of requests in the time interval ν_0. α and γ are the amplitude and duration of the periodic pattern. Also, $s_c Rand(\mathbb{N}(0, 1))$ is a scale factor that injects normal random noise with mean zero and unit variance in the workload.

In all experiments, the periodic arrival rate behaviour consists of 180-time intervals of 1 minute each, i.e., three hours. The value of χ_{ν_0} was set to 1, and γ was set to 60. The amplitude α has different values depending on the target application, as follows: Daytrader (850), QHD (1000), OB (100) and Travels (150). The parameter values for χ_{ν_0}, γ, and α were adapted from those used by Abdullah et al. (2021). The amplitude α refers to the highest throughput an application can handle, similar to Abdullah et al. (2021). The parameters used in the experiments are summarised in Table 5.3.

Table 5.3 – Parameters adopted to generate the periodic workload per application.

Application	Parameter	Value
All	χ_{ν_0}	1
	γ	60
	s_c	4
	ν_i (time intervals)	180
Daytrader	α	850
QHD	α	1000
OB	α	100
Travels	α	150

5.3.5 Auto-scaling Rules

The experiments considered horizontal auto-scaling, a widely-used approach to auto-scaling cloud applications (SINGH et al., 2019). The scaling mechanism triggers once every 15 seconds with a 5-minute stabilisation cool-down on both tools as done by the HPA. Due to technical setup restrictions, the maximum number of replicas per application has been limited to ten.

Both proactive tools were evaluated on three different forecasting horizons: 1 minute (1M), 3 minutes (3M) and 5 minutes (5M), similar to Alipour & Liu (2017), Dang-Quang & Yoo (2021), Galantino et al. (2021) and Toka et al. (2021). PK was configured to predict CPU usage, and the 80% threshold has been set to trigger adaptation, as done by Lanciano et al. (2021). MFT was configured to predict the number of pods, as done by Coulson, Sotiriadis & Bessis (2020).

5.3.6 Database

Both evaluated tools require previous data to operate proactively, i.e., time series. According to PK documentation[10], a historical time window of at least seven days is suggested. However, most papers use smaller data sets considering the previously defined forecasting horizons (GOLI. et al., 2021; MARIE-MAGDELAINE; AHMED, 2020; MOHAMED; EL-GAYAR, 2021; TOKA et al., 2021; YADAV; Rohit; YADAV, 2021). Table 5.4 shows the results of experiments carried out to compare the strategies in a shorter historical time window (12 hours) and the recommended one (7

[10] https://keda.sh/docs/2.6/scalers/predictkube/#trigger-specification

days).

The results indicate that even though the PK with the long historical time window, referred to as PKL, has lower costs in the 3M and 5M forecast horizons, these values are similar to those achieved for PK with a short historical time window. PKL was the most efficient strategy on the 5M horizon but did not have the lowest response time in any forecast horizon. Furthermore, the strategies were expected to have a significant discrepancy in efficiency, as PKL uses 14 times more data than PK and MFT for training, something not observed in the experiments.

Table 5.4 – Efficiency, total cost and response time results of auto-scaling tools to adapt Daytrader using univariate and multivariate strategies. The best results for each metric by each application are in bold, while the best results by forecast horizon are underlined.

Forecasting Horizon	Strategy	C	RT	$AE_{0.5}$
1-minute	PK	<u>540</u>	<u>**121.138**</u>	<u>**0.569**</u>
	PKL	604	137.578	0.643
	MFT	581	127.769	0.604
3-minute	PK	500	<u>125.506</u>	<u>0.569</u>
	PKL	<u>**495**</u>	126.902	0.571
	MFT	705	158.403	0.743
5-minute	PK	554	148.340	0.660
	PKL	<u>546</u>	142.106	<u>0.637</u>
	MFT	669	<u>131.982</u>	0.648

Thus, considering these empirical results and that both strategies use the same historical window size for training, it was decided to use the shorter 12-hour version on the comparative analysis, similar to what was done by Mohamed & El-Gayar (2021) and Yadav, Rohit & Yadav (2021). The time series are generated before the experiments and follow the parameters defined in Section 5.3.4, except for t_i, which is extended to 720 minutes (12 hours). A self-adaptive tool for managing microservices must be employed throughout the experiment to generate time series data for model training. Therefore, the widely used reactive tool HPA was adopted. More details about HPA are provided in Section 2.2.2.1.

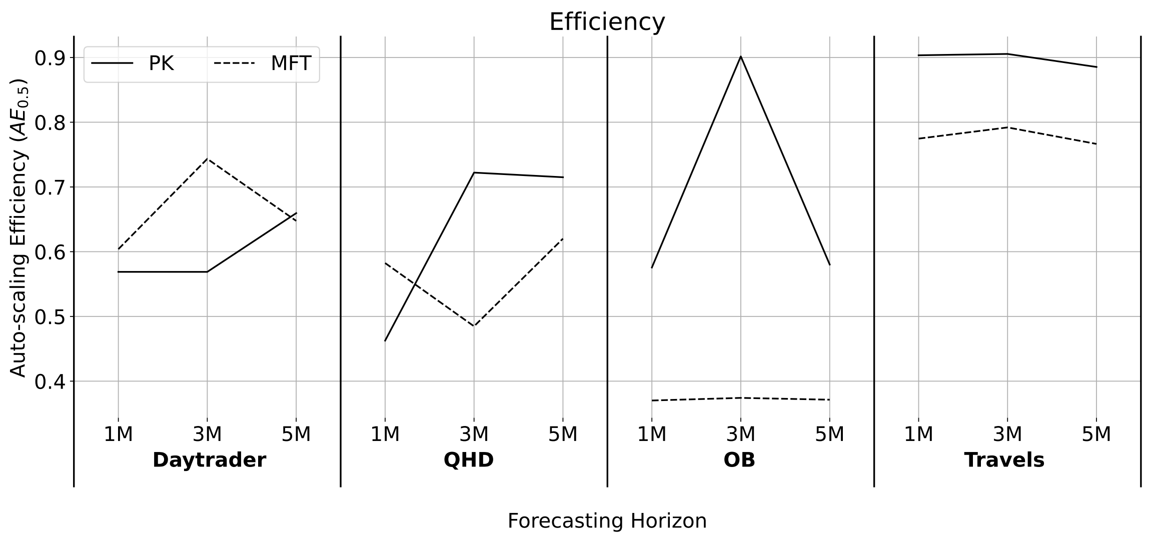
5.4 RESULTS

This section presents the comparative results of the univariate and multivariate strategies for proactive auto-scaling of MBAs. Drawing on the knowledge from Chapter 4, which focused on univariate forecasting, and Chapter 3, where the multivariate PMA module was introduced, this analysis delves into how these forecasting strategies can enhance the PMA tool to provide a more effective proactive auto-scaling. The research questions presented in Section 5.2 help guide the proposed comparative analysis. The comparative study source code, figures, and datasets are publicly available[11].

Figures 5.1, 5.2, and 5.3 compare the efficiency, response time, and cost of univariate (PK) and multivariate (MFT) strategies for managing the evaluated MBAs (see Section 7.2.1) across different forecast horizons. MFT is more efficient than PK in most applications, except for Daytrader. MFT reduced application response time compared to PK in 75% of the experiments, except for Daytrader 1M and 3M and QHD 1M. Regarding cost, PK incurred lower costs in Daytrader and QHD, while MFT was more cost-effective in OB and Travels.

The comparative analysis leads to the following conclusions:

Figure 5.1 – Auto-scaling efficiency ($AE_{0.5}$) results of univariate and multivariate strategies managing MBAs. For efficiency, lower values mean better results.



Daytrader. PK was more efficient than MFT in managing Daytrader in the 1M (-5.81%) and 3M (-23.49%) forecast horizons but slightly less efficient in the 5M (+1.84%) one. MFT decreased response time by -12.40% on the 5M horizon but increased by +5.19% on 1M and +20.77% on the 3M horizons. MFT also incurred higher costs than PK in all forecast horizons (+7.06%, +29.08%, and +17.19%, respectively). PK configured with the 3M horizon was the one that incurred lower costs, while the one with the 1M horizon had the lowest response time.

QHD. MFT was more efficient in managing QHD than PK in the 3M (-48.97%) and

[11] https://github.com/gfads/univariate-multivariate-prediction

Figure 5.2 – P^{95} percentile response time of MBAs managed by univariate and multivariate strategies. For response time, lower values mean better results.

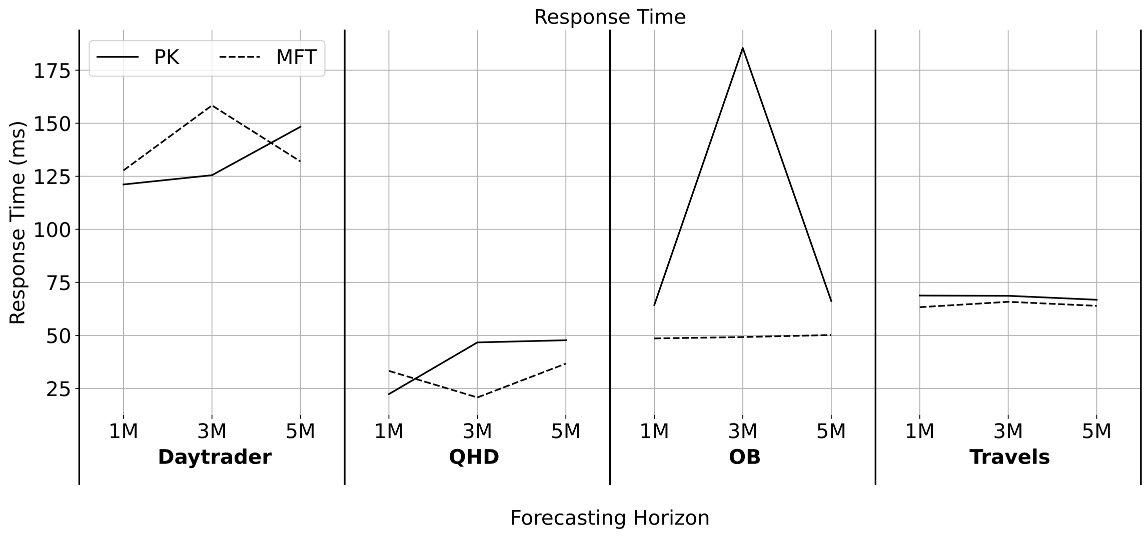
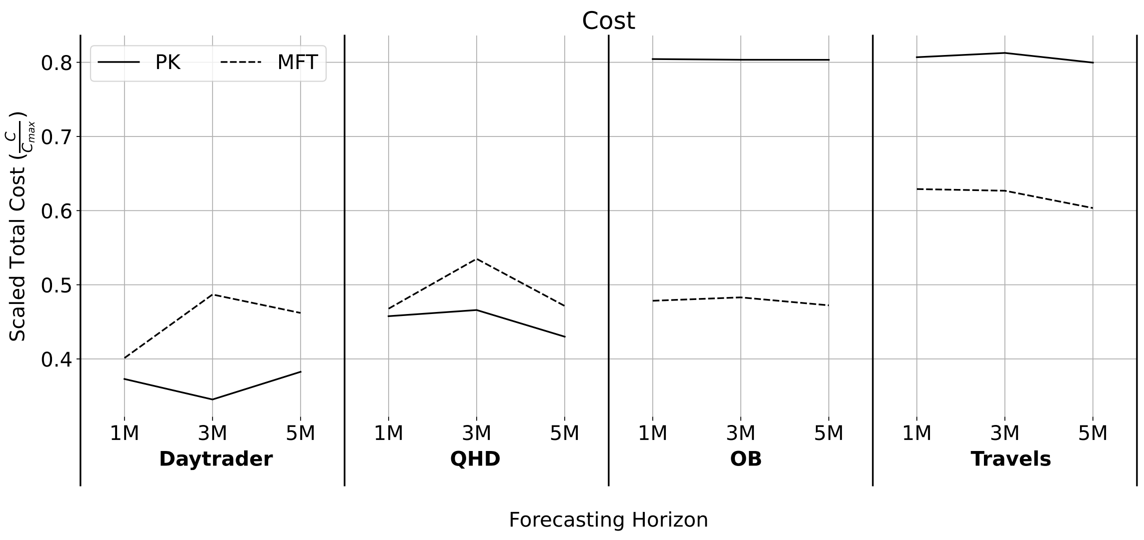


Figure 5.3 – Scaled total cost ($\frac{C}{C_{max}}$) of MBAs managed by univariate and multivariate strategies. For scaled total cost, lower values mean better results.



5M (-15.29%) forecast horizons, but it was less efficient in the 1M horizon (+20.57%). MFT decreased the response time of QHD by -125.15% for 3M and -30.06% for 5M forecasting horizons, and increased by +32.92% for the 1M one. Similar to experiments with Daytrader, MFT incurred higher costs than PK in all forecast horizons (+2.17%, +12.91%, and +8.79%, respectively). The PK configured with the 5M horizon was the one that incurred lower costs, while the MFT configured with the 3M horizon had the lowest response time.

OB. Unlike previous results, MFT is more efficient in managing OB when compared to PK in all forecast horizons (-55.51%, -141% and -56.22%). MFT also incurred lower costs (-68.13%, -66.34%, and -70.10%) and decreased response time (-32.44%, -276.92%, and -31.99%). MFT configured with the 5M horizon was the one that incurred lower costs, while the 1M horizon one had the lowest response time.

Travels. Similarly to OB, MFT is more efficient in managing OB when compared to PK in all forecast horizons (-16.62%, -14.33% and -15.51%). MFT incurred lower costs than PK (-28.26%, -29.65%, and -32.50%) and decreased response time (-8.66%, -4.30%, and -4.47%). MFT configured with the 1M forecast horizon had the lowest response time, while the 5M one also incurred the lowest costs.

5.5 DISCUSSION

The experiments showed that using a multivariate forecasting strategy (MFT) instead of a univariate one (PK) maintains or decreases the response time in most evaluated applications, except for Daytrader. The results indicate a correlation between the efficiency of the forecasting strategy and the application to be managed.

For example, the forecasting strategies are similar in efficiency in applications with a single container, e.g., Daytrader and QHD. MFT was more efficient on average for QHD (+15.15%) but was outperformed by PK for Daytrader (-8.81%). Also, MFT efficiency is guaranteed at a higher average cost than PK (+17.77% and +7.95%, respectively). On the other hand, for multi-container applications, such as OB and Travels, MFT is considerably more effective than PK (+85.59% and +13.51%, respectively), incurring lower costs (-68.19% and -24.79%).

Other interesting findings are that higher incurred costs do not result in lower response time, i.e., resource allocation has to be efficient, as experiments with lower response times commonly also had lower costs. Experiments with a forecast horizon of 1M had the lowest response times in all applications. On the other hand, the 5M forecast horizon tends to incur lower costs and is generally more efficient than the 3M horizon, except for Daytrader.

The results also indicate that as the application size increases, more than a single feature may be required to estimate the behaviour of the system. As a result, auto-scaling tools that rely solely on forecasts from univariate models to perform actions may fail as they do not consider the complete state of the system. On the other hand, in low-complexity applications, adopting a multivariate strategy has little or no advantage. Likewise, as demonstrated, the multivariate strategy commonly incurs more costs to maintain the efficiency of the managed system.

Based on the findings, **the efficiency of the forecasting strategy is impacted by the application to be managed**, mainly due to its complexity, confirming a positive answer to RQ5.1. RQ5.2 is also answered positively, but with reservations, since the multivariate strategy is more effective than the univariate one for complex applications. Thus, **adopting a multivariate approach becomes even more advantageous than a univariate strategy as the application complexity increases**.

The experiments conducted lay the groundwork for PMA, particularly the multivariate module presented in Chapter 3. The results emphasise the value of multivariate strategies for complex microservice systems, demonstrating significant advantages in multi-container environments. Consequently, they directly impact the PMA design and implementation, validating the need to incorporate multivariate models to capture the comprehensive state of the system. PMA is improved by multivariate forecasting, which makes it more versatile and generic. As a result, it becomes more effective in executing accurate and efficient auto-scaling actions across different scenarios.

5.6 CONCLUDING REMARKS

This chapter compared univariate and multivariate strategies for proactive auto-scaling of MBAs. Initially, the problem statement highlighted and discussed the research gap in the literature. Then, a detailed experimental protocol was provided for the comparative study. Next, an experimental evaluation was presented to assess the performance of the univariate and multivariate strategies. Finally, the results were deliberated.

6 MULTIPLE PREDICTOR SYSTEMS

This chapter introduces a Multiple Predictor Systems (MPS) for forecasting microservices time series, selecting the most suitable forecasters from a pool of models for each new test pattern. Section 6.1 explains how this chapter is related to PMA. Section 6.2 contextualises the problem and objectives of this chapter. Section 6.3 presents the proposed method. Section 6.4 details the experimental protocol applied. Subsequently, Section 6.5 shows an experimental evaluation to assess the performance of what is being proposed. After this, Section 6.6 discusses the results.

6.1 CONTEXT

The PMA tool was designed to optimise auto-scaling in microservices environments. This tool integrates three distinct forecasting modules: univariate, multivariate, and MPS. Each module offers a unique approach to predicting Microservice-based Applications (MBAs) performance, enhancing the PMA adaptability and accuracy in diverse scenarios.

This chapter focused on the MPS module and highlighted its essential role in enhancing system performance and resource management in complex microservices environments. The discussed results offer real-world evidence supporting the development and implementation of the MPS module, underlining its crucial role within the PMA tool. The multi-predictor or ensemble method improves forecasting accuracy and reliability by leveraging the strengths of different algorithms to create a robust forecasting system. This chapter builds upon the foundational concepts introduced in Chapter 3 and presents practical validations.

6.2 PROBLEM STATEMENT

As mentioned in Chapter 4, *Time Series Forecasting* (TSF) has been widely adopted for building proactive auto-scaling microservices systems (KANG; LAMA, 2020; YADAV; Rohit; YADAV, 2021; KAKADE et al., 2023). Nevertheless, existing solutions apply the Classical Forecasting Approach (CFA) (SILVA et al., 2020). CFA evaluates a set of learning algorithms and selects only the one with the highest forecast accuracy. However, the free lunch theorem (YAO; DAI; SONG, 2019) demonstrates that no single model can be optimal for all scenarios. Therefore, using only one predictor increases the risk of inaccurate estimates. The inaccurate estimates can conduct unsuitable interventions that may harm the customer experience.

A multi-predictor approach can mitigate the CFA problem by delegating the forecast task to multiple models. MPS, also called an ensemble, is an alternative for building systems with multiple predictors. The basic idea of the ensemble is to combine the strengths of different learning algorithms to build a more accurate and reliable forecasting system (QIU et al., 2017). MPS obtained better accuracy than approaches based on only one predictor, as reported by Widodo & Budi (2011), Kourentzes, Barrow & Crone (2014), Adhikari, Verma & Khandelwal (2015) and Moura, Cavalcanti & Oliveira (2021).

MPS encompass three phases: Generation, Selection and Integration. In the Generation phase, a pool of forecasting models is trained. The pool is called homogeneous when a single learning algorithm trains all models; otherwise, it is heterogeneous. In the second phase, one or more models from the pool are selected. Lastly, the Integration phase provides the final system forecast by combining forecasts from the different models chosen earlier.

This chapter proposes an MPS for forecasting microservices performance metrics. In the proposal, the Generation phase can generate homogeneous pools (Bagging method (BREIMAN, 1996)) or heterogeneous pools (combining different learning algorithms). In the Selection phase, the generated pool can be chosen through dynamic selection algorithms (Dynamic Selection (DS), Dynamic Weighting (DW) and Dynamic Weighting With Selection (DWS)) or statically combined (Mean and Median). The Integration phase can differ depending on the selection approach taken. It may not be required (DS), or it can involve using a simple mean (Mean and Median) or a weighted mean (DW and DWS) to combine pool predictions.

The following research questions guide the proposal presented:

- **RQ6.1** - How can MPS be used to forecast microservices performance metrics?
- **RQ6.2** - How effective are homogeneous and heterogeneous pools in increasing the accuracy of forecasting microservices performance metrics?
- **RQ6.3** - How do dynamic and static selection impact the accuracy of homogeneous and heterogeneous pools for predicting microservices performance metrics?
- **RQ6.4** - How effective are MPS and CFA for forecasting microservices performance metrics?

The following sections detail the MPS.

6.3 OVERVIEW

Figure 6.1 overviews the proposed MPS. MPS encompasses three phases: generation, selection and integration. The generation phase uses a time series database (λ) and a list of learning algorithms (Δ) to train a pool of forecasting models (P). Next, the selection phase chooses the most suitable models in P to forecast a given test pattern (x_j), considering the D_{sel} time series. D_{sel} is a validation dataset that contains a set of time series. It can be just λ or has additional time series. The selected models $\hat{\rho}_i \in \hat{P}$ forecast x_j. The integration phase combines the forecast of the $\hat{\rho}_i$ models to generate the system forecast ($\hat{P}(x_j)$).

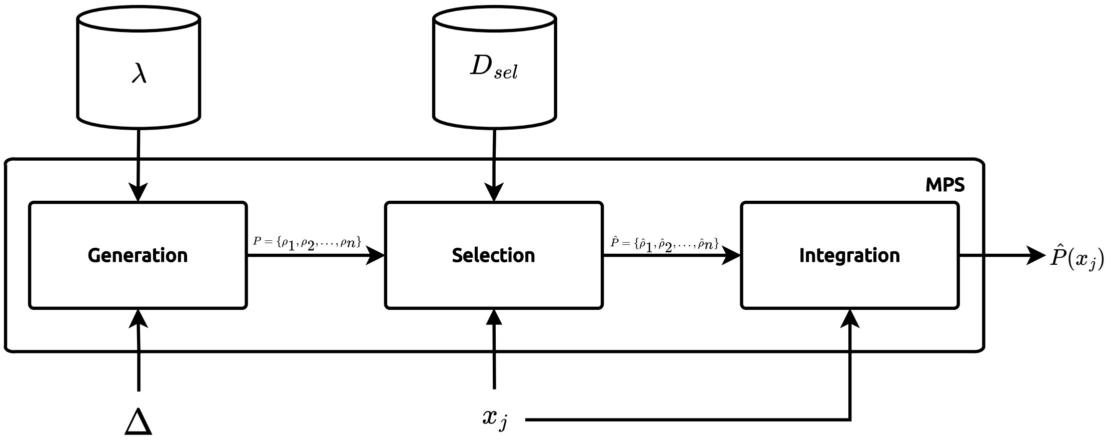

λ
D_{sel}
MPS
Generation
$P = \{\rho_1, \rho_2, ..., \rho n\}$
Selection
$\hat{P} = \{\hat{\rho}_1, \hat{\rho}_2, ..., \hat{\rho} n\}$
Integration
$\hat{P}(x_j)$
Δ
x_j

Figure 6.1 – MPS overview. λ is a time series database; Δ is a list of learning algorithms; P is a pool of forecasting models; D_{sel} is the validation dataset; x_j is a new test pattern; $\hat{\rho}_1(x_j), \hat{\rho}_2(x_j), ..., \hat{\rho}_n(x_j)$ are the forecasts of each model $\hat{p}_i \in \hat{P}$ selected by the selection phase; $\hat{P}(x_j)$ is the final forecast of the MPS.

The following sections detail the MPS phases.

6.3.1 Generation

Figure 6.2 presents an overview of the generation phase that aims to train a diverse pool of models capable of modelling different time series behaviours. The generation phase has two modules: Preprocessor and Pool Generator.

The *Preprocessor* module preprocesses the λ time series for training the *learning algorithms* (Δ). First, a scaling algorithm modifies the λ time-series observations in a standardised way for smaller ranges. Scaling can facilitate model training and improve forecast accuracy.

Next, the time series λ must be restructured as a supervised learning problem. An alterna-

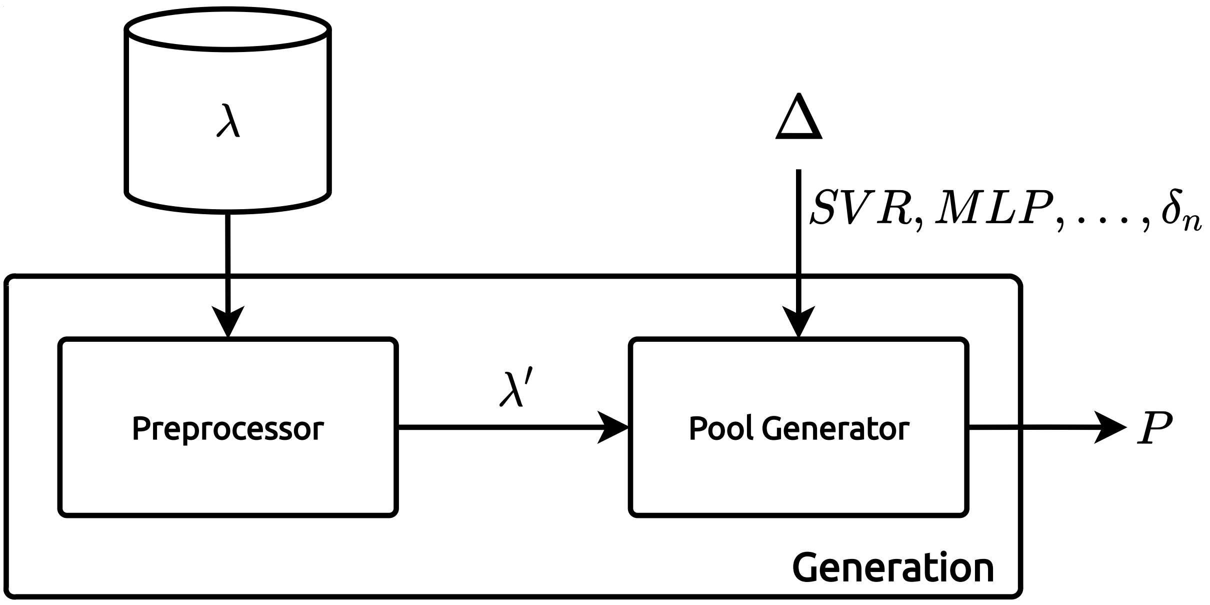


Figure 6.2 – Overview of the MPS generation phase. λ is a time series database; λ' is the preprocessed time series database; Δ is a list of learning algorithms; P is a pool of forecasting models.

tive for restructuring is the sliding window method (YU et al., 2014), which transforms the time series into fixed-size sliding windows. The fixed size is equivalent to the time series lags. Lags are past observations of the time series that most impact the training of the models. Autocorrelation function or optimisation algorithms (SCARGLE, 1989) are alternatives for defining lags. Finally, a preprocessed λ' is sent to the next module.

The *Pool Generator* module uses λ' and a list of learning algorithms (Δ) to generate a pool of models P. The generated pool can be either homogeneous, which means all models were trained using the same learning algorithm, or heterogeneous, which means different learning algorithms were used to train the models (MOURA; CAVALCANTI; OLIVEIRA, 2021).

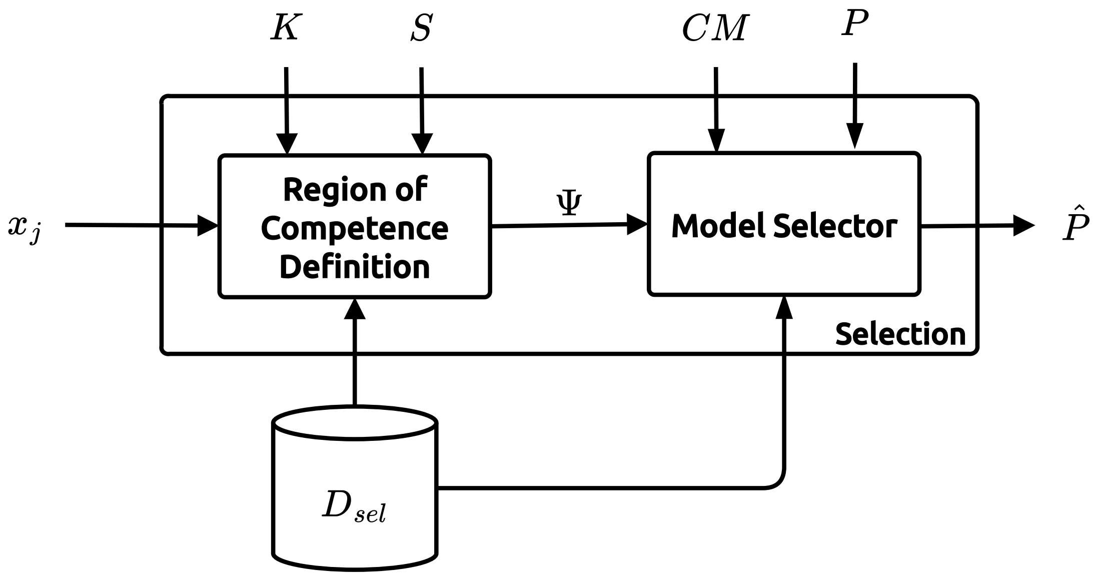
The heterogeneous pools comprise a model for each learning algorithm. For example, if Δ has two algorithms and is trained on a λ' structured into two distinct sliding window sizes (e.g., $L = \{20, 40\}$), four different models are trained ($\Upsilon = \{\upsilon_1^{20}, \upsilon_1^{40}, \delta_2^{20}$, and $\delta_2^{40}\}$). Then, the υ_i^l model that composes the heterogeneous pool has the highest accuracy among its variations, e.g., assuming that υ_1^{20} and υ_2^{40} were more accurate than υ_1^{40} and υ_2^{20}, these models must be integrated into the pool as algorithms 1 and 2, respectively.

Homogeneous pools, by contrast, are composed of models of a single learning algorithm. The selected algorithm used to generate the homogeneous pool is the one that trained the $\upsilon_i^l \in \Upsilon$ with the highest forecast accuracy. The homogeneous pool can be created by methods such as Adaboosting (BARROW; CRONE, 2016) or Bagging (BREIMAN, 1996) that use the parameters (learning algorithm and sliding window size) of the most accurate υ_i^l model as input.

6.3.2 Selection

Figure 6.3 presents the overview of the selection phase that chooses one or a subset of the most suitable models in P to forecast a given test pattern (x_j). The selection phase performs a dynamic or static selection of models and comprises two modules: Region of Competence Definition and Model Selector.

Figure 6.3 – Overview of the MPS selection phase. x_j is a new test pattern; K is the size of the Region of Competence (RoC); S is a similarity measure; D_{sel} is the validation dataset; Ψ is an RoC; CM is a competence measure; P is a pool of forecasting models; $\hat{P}$ is the pool of models selected.



For dynamic selection, a new $\hat{P}$ (pool) is chosen for each test pattern x_j. The first step is defining a Region of Competence (RoC) (Ψ) composed of the *K* time windows most similar to test pattern x_j in *Dsel*. The correlation between two-time windows is calculated using *similarity measures* (S) – for example, Euclidean distance (SILVA et al., 2018).

The *Model Selector* selects the P models most suitable for forecasting x_j. First, it calculates the accuracy using a *competence measure* (CM) of each model $p_i \in P$ to forecast patterns in Ψ. Subsequently, according to accuracy, P models are ranked from best to worst. Finally, $\hat{P}$ models are selected according to the dynamic selection algorithm employed. For example, DS selects only the best model. In contrast, others, such as DW, select a subset of the best models from P, having a size greater than or equal to one (ROONEY et al., 2004).

All models are used for static selection, i.e., $\hat{P} = P$. Therefore, the pool is the same for any test pattern (x_j).

6.3.3 Integration

Figure 6.4 presents an overview of the integration phase that generates the final system forecast $(\hat{P}(x_j))$. The integration phase comprises two modules: Forecaster and Integrator.

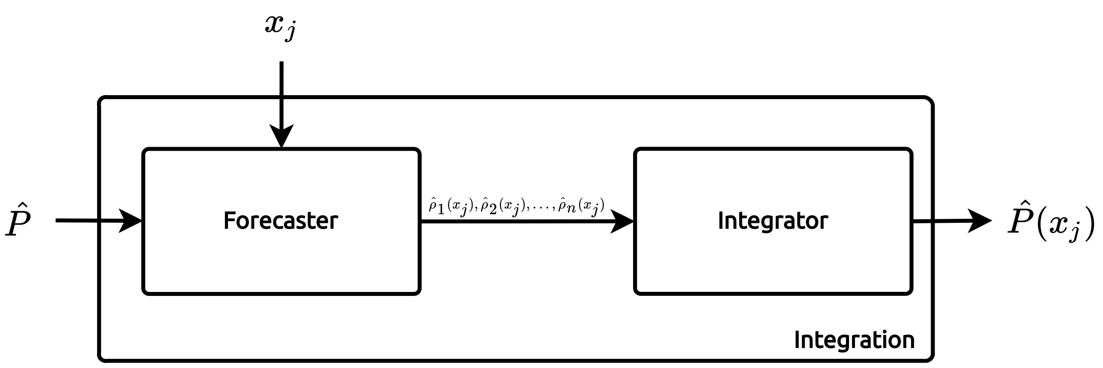

x_j
$\hat{P}$
Forecaster
$\hat{\rho}_1(x_j), \hat{\rho}_2(x_j), \ldots, \hat{\rho}_n(x_j)$
Integrator
$\hat{P}(x_j)$
Integration

Figure 6.4 – Overview of the MPS integration phase. $\hat{P}$ is a pool of models selected; x_j is a new test pattern; $\hat{\rho}_1(x_j), \hat{\rho}_2(x_j), ..., \hat{\rho}_n(x_j)$ are the forecasts of each model $\hat{p}_i \in \hat{P}$ selected by the selection phase; $\hat{P}(x_j)$ is the final forecasting of the MPS.

In the *Forecaster* module, each $\hat{\rho}_i \in \hat{P}$ model forecasts test pattern x_j. Afterwards, the Integrator module must merge the forecasts to compute the final system forecast $\hat{P}(x_j)$. Forecasts are combined using approaches such as linear combination (MENDES-MOREIRA et al., 2012) or simple procedures such as mean and median (MOURA; CAVALCANTI; OLIVEIRA, 2021). If $\hat{P}$ results in a single model, the integration becomes unnecessary, i.e., $\rho_1(x_j) = \hat{P}(x_j)$.

6.4 EXPERIMENTAL PROTOCOL

This section presents the experimental protocol for validating the MPS proposed in Section 6.3, including the datasets and techniques employed.

6.4.1 Datasets

Sixteen time series were used for the experiments, four for each performance metric: CPU usage, memory, response time and traffic. They are a random sub-sample of those used in

Section 4.3. These real-world series were collected from Alibaba production clusters[1] (LUO et al., 2021). The Alibaba dataset includes performance metrics such as CPU usage, memory, response time and traffic.

The selected series (see Table 6.1) comprises different communication mechanisms among microservices: inter-process communication (IPC) (STEEN; TANENBAUM, 2017), remote invocation (RI) (SRIRAMAN; WENISCH, 2018) and indirect communication (IC) (GAN et al., 2019a). CPU usage and memory series consist of 30-second records taken over twelve hours, producing an average of 1440 observations. On the other hand, response time and traffic series are composed of 1-minute records over the same twelve-hour period, resulting in an average of 720 observations. Figure 6.5 shows all real-world time series.

Table 6.1 – Description of the real-world datasets.

Metric	Series	Trend	Stationary	Frequency	Mean	Median	Std	Size	Communication
CPU usage	1	✗	✗	Seconds	0.34	0.33	0.05	1,426	RI, IC, IPC
	2	✗	✓	Seconds	0.34	0.40	0.11	1,427	RI
	3	✗	✗	Seconds	0.18	0.15	0.07	1,420	RI, IC, IPC
	4	✗	✗	Seconds	0.32	0.31	0.04	1,421	RI, IC
Memory	1	✓	✗	Seconds	0.53	0.52	0.04	1,427	RI, IC
	2	✗	✓	Seconds	0.51	0.50	0.03	1,426	RI
	3	✓	✓	Seconds	0.52	0.52	0.00	1,426	RI, IPC
	4	✗	✓	Seconds	0.45	0.45	0.02	1,424	RI, IC
Response time	1	✗	✗	Minutes	1.00	1.02	0.20	720	RI
	2	✗	✗	Minutes	23.75	23.95	3.06	720	RI
	3	✓	✗	Minutes	59.94	58.24	14.79	721	IC
	4	✗	✗	Minutes	470.38	371.96	255.56	715	IC
Traffic	1	✓	✗	Minutes	222.39	220.58	12.42	721	RI
	2	✗	✗	Minutes	50.75	54.37	11.69	721	RI
	3	✗	✗	Minutes	111.60	44.29	87.34	713	IC
	4	✗	✗	Minutes	258.10	255.12	40.43	721	IPC

All series are referred to hereafter as λ. More descriptive details, plots, and time series files can be found in a public repository[2]. λ time series were used in an experimental study considering a one-step-ahead forecasting scenario. Each time series was divided as follows: the first 60% for training, 20% for validation and the last 20% for testing, as adopted by Wong (2018). The following sections detail the experimental protocol according to the MPS phases presented in Section 6.3.

[1] More details about the entire database are available in the Alibaba public repository: https://github.com/alibaba/clusterdata/tree/master/cluster-trace-microservices-v2021.

[2] https://github.com/gfads/mps-methodology/

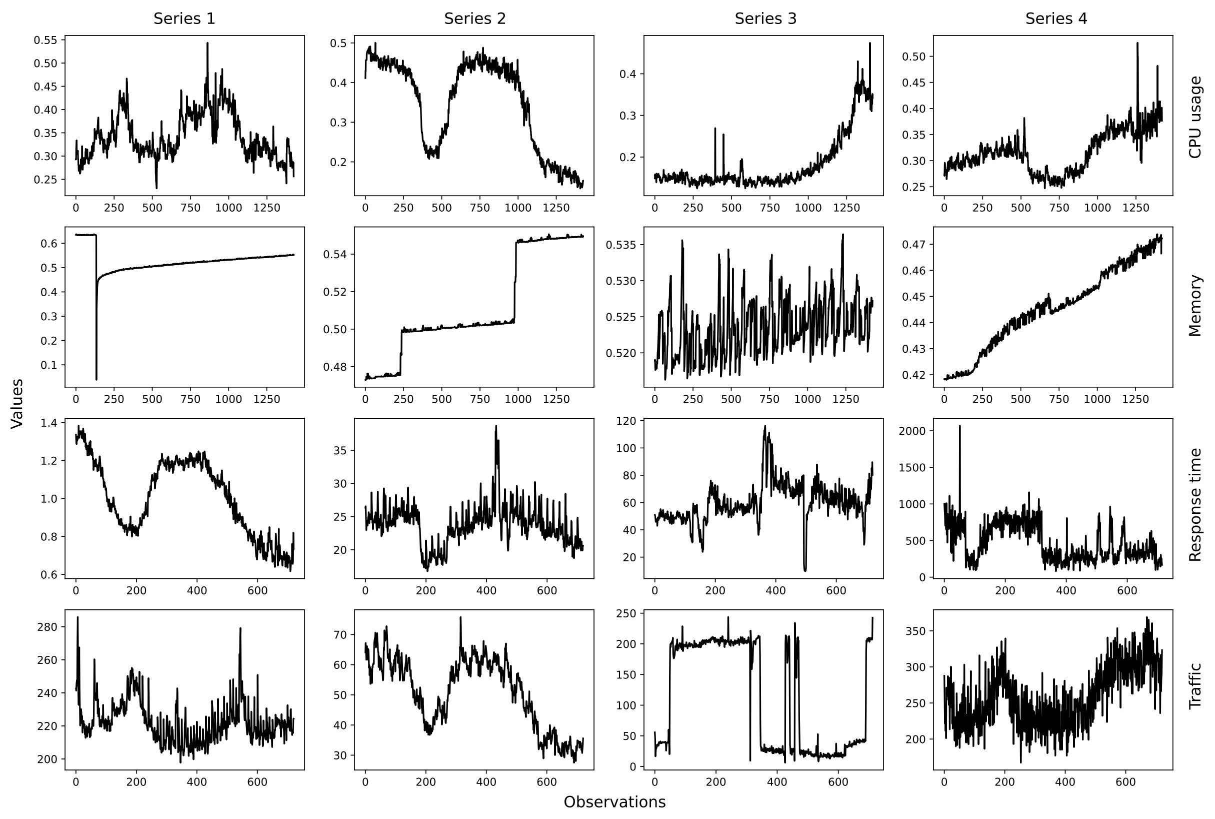


Figure 6.5 – Real-world time series.

6.4.2 Generation

The experimental protocol used in the Preprocessor and Pool Generator modules includes:

Preprocessor. Min-max normalisation (SHALABI; SHAABAN; KASASBEH, 2006) was adopted to scale the λ time series to the interval [0, 1]. This normalisation method was defined in Section 4.1. λ time series was also organised into sliding windows, as done in the comparative analysis (see Chapter 4), having six distinct maximum sizes for lags (L = $\{$10, 20, 30, 40, 50 and 60$\}$) selected using the auto-correlation function (BOX et al., 2015), producing a preprocessed $\lambda' = \{\lambda'_1, \lambda'_2, ..., \lambda'_6\}$.

Pool Generator. The experimental study is performed using different learning algorithms. The list of algorithms Δ includes four Machine Learning (ML) algorithms (Multilayer Perceptron (MLP) (HAYKIN, 2001), Support Vector Regressor (SVR) (DRUCKER et al., 1997), Random Forest (RF) (BREIMAN, 1996), and eXtreme Gradient Boosting (XGBoost) (CHEN; GUESTRIN, 2016)), one statistical algorithm (AutoRegressive Integrated Moving Average (ARIMA) (PRYBUTOK; YI; MITCHELL, 2000)) and one Deep Learning (DL) algorithm (Long Short-Term Memory (LSTM) (MA et al., 2015)).

The list of algorithms used for evaluation in this chapter is smaller than the one used in

Chapter 4. This is because it was shown in Chapter 4 that traditional ML models were either equally or more accurate than DL models. As this evaluation aims to assess the accuracy of the MPS, any algorithms could be included in the list. Thus, a subset favouring them was selected as ML algorithms train and predict faster than DL ones. LSTM has been selected as an instance of DL algorithms due to its popularity in microservices forecasting (COULSON; SOTIRIADIS; BESSIS, 2020; GOLI. et al., 2021; MOHAMED; EL-GAYAR, 2021) and the performance it showcased in Chapter 4. All algorithms were trained using a grid search approach to find the best hyper-parameters per model. Table 6.2 shows the hyper-parameters used for training the models. The validation set was used to select the best hyper-parameters per model and the time-sliding window size.

Table 6.2 – Hyper-parameters used for training the models.

Algorithm	**Hyper-parameters**
ARIMA	*Autoarima library*
LSTM	'batch_size': [64, 128], 'epochs': [1, 2, 4, 8, 10], 'hidden_layers': [2, 3, 4, 5, 6], 'learning_rate': [0.05, 0.01, 0.001] (COULSON; SOTIRIADIS; BESSIS, 2020)
MLP	'hidden_layer_sizes': [5, 10, 15, 20], 'activation': ['tanh', 'relu', 'logistic'], 'solver': ['lbfgs', 'sgd', 'adam'], 'max_iter': [100, 500, 1000, 2000, 3000], 'learning_rate': ['constant', 'adaptive'] (RUBAK, 2023)
RF	'min_samples_leaf': [1, 5, 10], 'min_samples_split': [2, 5, 10, 15], 'n_estimators': [100, 500, 1,000] (ESPINOSA et al., 2021)
SVR	'gamma': [0.001, 0.01, 0.1, 1] 'kernel': ['rbf', 'sigmoid'] 'epsilon': [0.1, 0.001, 0.0001] 'C': [0.1, 1, 10, 100, 1,000, 10,000] (OLIVEIRA; SILVA; NETO, 2022)
XGBoost	'col_sample_by_tree': [0.4, 0.6, 0.8], 'gamma': [1, 5, 10], 'learning_rate': [0.01, 0.1, 1], 'max_depth': [3, 6, 10], 'n_estimators': [100, 150, 200], 'reg_alpha': [0.01, 0.1, 10], 'reg_lambda': [0.01, 0.1, 10], 'subsample': [0.4, 0.6, 0.8] (MOHAMED; EL-GAYAR, 2021)

The heterogeneous pool has six models, one for each learning algorithm in the Δ list. However, adopting six lags generates six variations of a single model type. Therefore, the one chosen to compose the heterogeneous pool by type had the best accuracy considering its variations (see Section 6.3.1).

The homogeneous pool uses only a single learning algorithm in the generation process. The selected learning algorithm has higher accuracy considering the 36 model variations: six learning algorithms at six different lags. Next, the selected parameters train the new models with the Bagging method (BREIMAN, 1996). The homogeneous pool size varies from 10 to 150 (by steps of 10). The final size adopted follows the same procedure for selecting the homogeneous pool algorithm.

The adopted protocol includes several lags and homogeneous pool sizes to cover a broader range of experimental scenarios. The objective was to investigate the influence of these parameters on microservices' time series. The individual performance of each variation and the analysis of the homogeneous pool sizes are available in the supplementary material[3].

The trained accuracy of the models is evaluated using the RMSE, as done by Kumar & Singh (2018), Moreno-Vozmediano et al. (2019), Moura, Cavalcanti & Oliveira (2021). The Root Mean Square Error (RMSE) is defined in Equation 6.1:

$$RMSE = \sqrt{\frac{1}{|T|}\sum_{t=1}^{T}(t_i - p(t_i))^2} \tag{6.1}$$

where t_i is the actual value of the time series; $p(t_i)$ is the value forecasted by the model; $|T|$ is the number of observations in each λ'.

6.4.3 Selection

The experimental protocol used in the Region of Competence Definition and Model Selector modules includes:

Region of Competence Definition. The Region of Competence (RoC) size (K) was defined as ten, and the similarity measure (S) adopted was the Euclidean Distance, as done by Moura, Cavalcanti & Oliveira (2021), Mendes-Moreira et al. (2009).

Model Selector. The models were selected using three dynamic selection algorithms: Dynamic Selection (DS), Dynamic Weighting (DW) and Dynamic Weighting With Selection (DWS) (MENDES-MOREIRA et al., 2009; MOURA; CAVALCANTI; OLIVEIRA, 2021). The competence measure (CM) adopted to assess the forecast accuracy of the models Ψ is RMSE, the same used for model training (see Section 6.4.2).

All dynamic algorithms adopted need the distance vector d_k, which is the distance between RoC $(\psi_1, \psi_2, ..., \psi_K)$ and test pattern x_j, defined in Equation 6.2.

$$d_k = \frac{\frac{1}{dist_k}}{\sum_{i=1}^{K}\left(\frac{1}{dist_j}\right)}, \tag{6.2}$$

where $dist_k$ is the distance between a pattern $\psi_k \in \Psi$ and test pattern x_j; d_k is the weight vector.

[3] https://github.com/gfads/mps-methodology/tree/main/results

DS selects only the ρ_i model with the lowest error in *CM*. d_k provides weighting for the errors. The DS forecast integration is unnecessary as only one model is selected. DS selection is defined in Equation 6.3.

$$\hat{P} = \underset{1 \leq k \leq K}{min} \{CM \times d_k\} \tag{6.3}$$

where CM is an competence measure and d_k is a distance vector defined in Equation 6.2.

DW selects and gives weighting to all *P* pool models. Weight βi referring to the model $\hat{\rho}_i$ is calculated as defined in Equation 6.4.

$$\beta_i = \frac{\frac{1}{\sqrt{\sum_{k=1}^{K}(d_k \times CM_{k,i})}}}{\sum_{n=1}^{N}\left(\frac{1}{\sqrt{\sum_{k=1}^{K}(d_k \times CM_{k,n})}}\right)}, \tag{6.4}$$

where *N* is the P pool size; k represents the index of the neighbour; $CM_{k,i}$ is the error of model ρ_i to forecast $\psi_k \in \Psi$.

DWS selects and weights only a subset of *P* pool models. Models with an error greater than half of the error range are discarded, i.e., $E_i > (E_{max} - E_{min})/2$, where E_{max} and E_{min} are the highest and lowest error of any $\rho_i \in P$, respectively. After that, the subset is submitted to the same process adopted by DW.

6.4.4 Integration

The experimental protocol used in the Integrator module includes:

Integrator. The integrator varies according to the selection algorithm adopted. DS does not need integration because it selects a single model. DW and DWS use weighted averages in their integrations.

Two integration strategies were adopted in which selection algorithms were not applied, namely Mean and Median, as done by Moura, Cavalcanti & Oliveira (2021). The integration strategies calculate the mean and median of forecasts from pool $\hat{P}$, respectively. The objective was to contrast popularly used dynamic selection algorithms against simple integration approaches.

6.4.5 Evaluation Measures

The RMSE was adopted to evaluate the accuracy of the approaches. It is also applied to the *model training and selection* (see Sections 6.4.2 and 6.4.3).

The Diebold-Mariano Statistical test (DIEBOLD; MARIANO, 2002) was used to check whether the proposed MPS performs as well as the CFA, as done by Neto et al. (2022). The test aims to determine whether the two forecasts are significantly different. If the *p*-value is greater than 0.05, the accuracy of the MPS approach is deemed statistically equivalent to that of CFA. However, if the *p*-value is less than or equal to 0.05, the more accurate approach is the one with the lowest RMSE value.

The Accuracy Percentage Difference (APD) was adopted to compare the approaches: MPS and CFA, as done by Wang et al. (2018), Silva, Neto & Cavalcanti (2021). The APD is defined in Equation 6.5:

$$APD = \frac{(\epsilon_\alpha - \epsilon_\beta)}{\epsilon_\alpha} \times 100, \tag{6.5}$$

where ϵ_α is the baseline accuracy (i.e., CFA), and ϵ_β is the accuracy of the proposed approach, i.e., MPS. The higher the percentage difference (positive), the better the proposed approach outperforms the baseline.

6.5 RESULTS

This section evaluates the MPS as an alternative to improve the forecasting accuracy of microservices time series. The proposed MPS is the one used in the PMA MPS module. The evaluation compares the CFA, which uses only a single forecasting model, against the proposed MPS, which uses a pool of predictive models. The results of experiments on the real-world series are presented, followed by a statistical evaluation of the findings. Detailed results on the accuracy of the approaches, including additional metrics, can be found in a public repository[4].

Table 6.3 shows the RMSE of the CFA models, considering the 16 real-world time series described in Section 6.4.1. The SVR had better accuracy on eight series (50%), MLP on four (25%), RF on two (12.5%), and XGBoost and LSTM on one each (7.5%).

[4] https://github.com/gfads/mps-methodology/

Table 6.3 – RMSE results of the CFA models for real-world series. The best result per time series is in bold. Error values are in 10^{-3} scale.

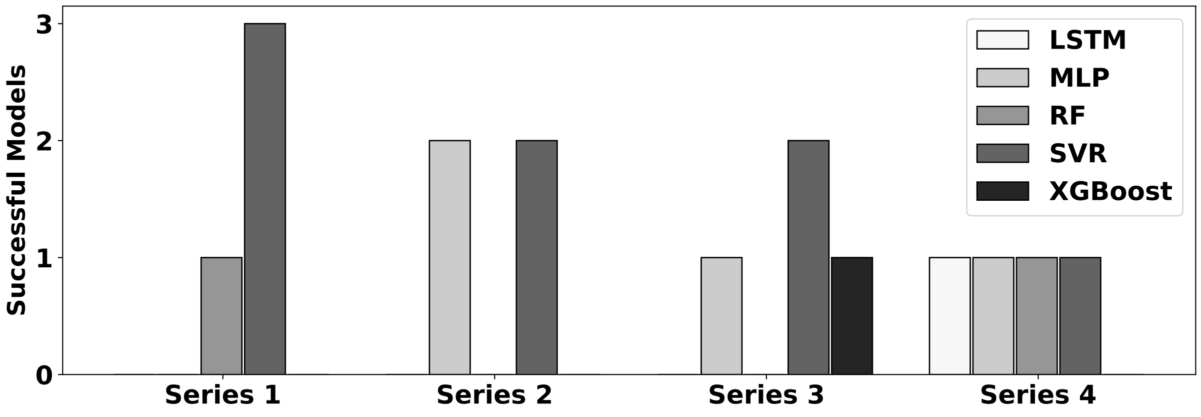
		Models					
Metric	**Series**	**ARIMA**	**LSTM**	**MLP**	**RF**	**SVR**	**XGBoost**
CPU usage	1	76.42	53.76	36.23	38.35	**35.56**	37.54
	2	149.33	38.94	32.08	133.73	**25.89**	182.84
	3	348.04	434.01	61.81	311.33	**41.99**	230.30
	4	110.16	86.53	**64.95**	121.04	65.73	86.13
Memory	1	12.37	2.52	1.38	33.47	**1.36**	13.93
	2	114.68	6.24	4.25	580.65	**3.83**	544.53
	3	181.06	151.42	**73.29**	80.65	75.44	86.59
	4	30.48	32.41	24.19	316.97	**23.34**	358.88
Response time	1	96.07	103.16	56.10	163.33	**53.51**	210.35
	2	155.71	83.22	**51.23**	60.98	51.70	60.46
	3	93.15	87.68	64.33	61.13	63.90	**59.40**
	4	67.93	56.52	51.50	**50.90**	51.97	52.70
Traffic	1	67.62	69.89	65.90	**62.96**	66.40	66.01
	2	54.90	87.28	**38.09**	109.37	39.20	152.84
	3	328.30	197.74	51.03	77.72	**45.21**	130.78
	4	123.07	**122.38**	128.44	143.07	128.11	177.10

Figures 6.6a and 6.6b summarise the successful model (i.e., the most accurate model compared to other CFA models) for each time series and microservice performance metric. Regarding the series (Figure 6.6a), SVR and RF were the most accurate for forecast Series 1, and SVR and MLP were the most accurate for Series 2. More models stand out in Series 3 and 4, with MLP, SVR and RF being more accurate in Series 3 and LSTM, MLP, SVR and RF in Series 4. Based on the metrics analysis (Figure 6.6b), SVR was more accurate for CPU usage and memory series, MLP achieved a good result in at least one of the metrics, and LSTM, RF and XGBoost achieved better results in predicting the response time and traffic metrics. It is worth highlighting the performance of the SVR model in the different series.

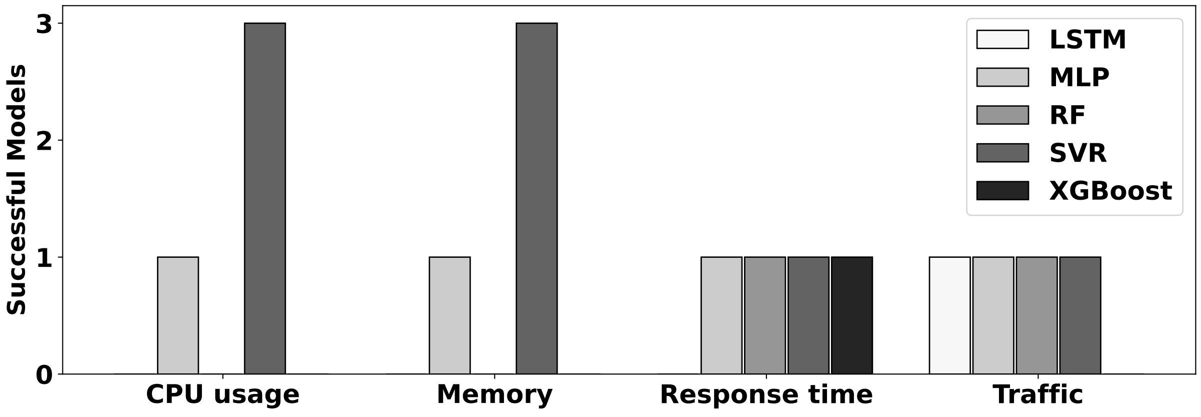
Although the SVR was the best model compared to other monolithic models, it was not better in all series. Thus, choosing only a single model can be a risky strategy due to the limitation of CFA. Hence, as mentioned in Section 5.5, MPS emerges as an alternative to this problem since it uses a pool of forecasting models.

Table 6.4 compares the RMSE results of the MPS approach and CFA on the real-world series. CFA results are those achieved by SVR, the model with the highest accuracy in the real-world time series. The MPS achieved better results than CFA in 87.5% (14 out of 16

Figure 6.6 – Successful CFA models of real-world series evaluated from the point of view of time series (a) and metrics (b).



(a) Time Series



(b) Metrics

datasets). The homogeneous attained better accuracy at 62.5%, the heterogeneous at 25% and the CFA at 12.5%. CFA outperformed MPS only in Series 1 CPU usage and Series 2 memory. Dynamic algorithms (DS, DW and DWS) stood out in homogeneous pool results, while the Median static combination and DS excelled in the heterogeneous pool.

It is worth noting that some of the results are the same in different selection strategies. For example, DW and DWS for CPU usage Series 1 and 3 or Traffic Series 3 and 4. As the selection mechanisms of DW and DWS algorithms are similar, they can select the same models to forecast all test patterns, resulting in the same final accuracy.

Regarding the CPU usage metric, the MPS algorithm outperformed the CFA in 75% of the series. The SVR algorithm was the most accurate for Series 1. The DS homogeneous algorithm was the best for Series 2 and 4, and the Mean homogeneous algorithm for Series 3. Similarly, the MPS algorithm was more accurate than CFA in 75% of the series for the memory metric. The DS homogeneous achieved the highest accuracy for Series 1, SVR for Series 2, DWS

Table 6.4 – Accuracy (RMSE) of MPS and CFA for real-world time series. The best results are in bold, and the second-best ones are underlined per time series. Error values are in 10^{-3} scale.

		CFA	**Homogeneous**					**Heterogeneous**				
Metric	**Series**	**SVR**	**MEAN**	**MEDIAN**	**DS**	**DW**	**DWS**	**MEAN**	**MEDIAN**	**DS**	**DW**	**DWS**
CPU usage	1	**35.56**	37.30	37.26	37.85	37.34	37.34	39.16	35.90	38.00	68.83	74.21
	2	25.89	25.93	25.87	**25.56**	26.13	26.14	80.58	79.30	42.81	48.08	45.10
	3	41.99	**41.01**	41.39	41.79	41.05	41.05	217.48	268.05	42.02	154.53	152.75
	4	65.73	63.15	63.88	**62.10**	62.76	62.75	75.81	76.29	71.59	113.36	113.94
Memory	1	1.36	1.34	1.35	**1.33**	1.34	1.34	10.24	7.28	12.37	5.29	5.27
	2	**3.83**	4.17	4.26	3.87	4.11	4.11	171.89	4.37	7.94	8.41	8.40
	3	75.44	68.50	69.58	73.43	67.71	**67.52**	89.85	80.95	83.37	79.08	78.90
	4	23.34	23.41	23.43	23.59	23.40	23.40	115.83	27.40	**23.24**	24.92	24.69
Response time	1	53.51	52.91	52.70	**51.68**	52.85	52.85	83.95	71.44	159.46	139.09	150.06
	2	51.70	49.73	49.76	50.83	49.49	**49.48**	60.65	52.57	60.86	72.82	72.19
	3	63.90	58.70	58.63	60.14	**58.48**	58.50	62.87	59.91	64.43	90.51	93.02
	4	51.97	51.43	51.52	54.32	51.06	51.64	49.77	**49.47**	49.78	56.72	56.73
Traffic	1	66.40	62.41	62.66	63.26	62.59	62.91	**58.18**	60.07	62.03	74.68	75.20
	2	39.20	36.75	36.95	42.17	**36.57**	**36.57**	63.64	60.87	52.85	66.22	72.17
	3	45.21	49.71	49.80	45.78	48.47	**42.59**	105.53	62.11	70.22	190.75	190.97
	4	128.11	176.31	177.50	133.10	169.86	169.86	126.15	**124.29**	130.39	125.22	125.22

homogeneous for Series 3 and DS heterogeneous for Series 4.

MPS outperformed CFA in all response times and traffic series. Regarding response time, DS homogeneous was more accurate for Series 1, DWS homogeneous for Series 2, DW homogeneous for Series 3 and Median heterogeneous for Series 4. Regarding traffic series, Mean heterogeneous achieved better results for Series 1, DW and DWS homogeneous for Series 2, DWS homogeneous for Series 3 and Median heterogeneous for Series 4.

Although the CFA was more accurate than the MPS in some series (2 out of 16), all homogeneous algorithms achieved similar accuracies. For most heterogeneous pool results, only the Median and DS algorithms achieved CFA-like accuracy. The results also show that the Median was more accurate than the Mean in most heterogeneous MPS results. Static combination pools comprise all trained models, including less accurate ones, such as ARIMA, RF and XGBoost. Therefore, any outlier (i.e., inaccurate forecast) can decrease the accuracy of the Mean selection algorithm. The Median mitigates the impact of inaccurate models by using the median forecast of the pool, which does not consider outliers commonly found at the extremes. Such a statement is also supported by the fact that static combinations achieve similar accuracy in homogeneous pools. As these pools are comprised of a single algorithm, their forecasts are less diverse, mitigating the existence of outliers. Furthermore, when dynamic algorithms have the same accuracy as static ones, static integration approaches should be prioritised for system building, as they do not perform the selection phase, which speeds up the forecast.

Statistical Analysis. Table 6.5 shows the percentage accuracy difference between the best homogeneous and heterogeneous algorithms compared to the best, intermediate and worst CFA models in the real-world series. The algorithms are classified based on their overall accuracy across all series. The highest overall accuracy determines the best algorithm. CFA Intermediate Model (IM) and Worst Model (WM) are the third and last algorithms considering overall accuracy. Based on this criterion, the best algorithm for homogeneous and heterogeneous pools in real-world series is DS. SVR is the Best Model (BM), LSTM is the IM, and XGBoost is the WM.

The symbols $+$, $—$, and $\sim$ in Table 6.5 mean that the proposed homogeneous and heterogeneous algorithms attained better, worse, or equal statistical accuracy than the CFA models (see Section 6.4.5). The final three rows in Table 6.5 summarise the results. *Wins* are computed as the percentage of series where the proposed algorithm (i.e., homogeneous or heterogeneous) achieves statistically significant improved accuracy over CFA. *Loss* means the opposite scenario to that described previously. *Tie* denotes the percentage of series where the proposed algorithm and CFA accuracy were not statistically different.

Considering the 48 comparisons between the MPS and CFA (i.e., 16 datasets with three CFA variations), the best homogeneous approach obtained 25 wins (52.1%), 22 ties (45.8%) and one loss (2.10%). On the other hand, the best heterogeneous approach obtained 18 wins (37.5%), 21 ties (43.75%) and nine losses (18.75%). These results indicate that the homogeneous approach is more accurate than the heterogeneous approach for forecasting microservices time series.

When focusing solely on BM results, the best homogeneous approach was equal to or better in 100% of cases, while the best heterogeneous approach only achieved this in 62.5% of cases. Therefore, since the best CFA model was unknown before analysis, these results suggest that using a homogeneous approach can enhance the accuracy of the forecast adaptive system and mitigate the CFA problem. At the same time, opting for a heterogeneous approach aiming for the highest accuracies is not recommended.

Table 6.5 – Percentage (%) difference between the best homogeneous and heterogeneous algorithms compared to BM, IM, and WM in real-world series computed using Equation 6.5. The statistical result was obtained using the Diebold Mariano statistical test (see Section 6.4.5). DS is the best homogeneous and heterogeneous algorithm, SVR is the BM, LSTM is the IM, and XGBoost is the WM

		Homogeneous			**Heterogeneous**		
Metric	**Series**	**BM**	**IA**	**WM**	**BM**	**IM**	**WM**
CPU usage	1	-6.45 (∼)	29.59 (+)	-0.83 (∼)	-0.39 (−)	29.31 (+)	-1.22 (∼)
	2	1.26 (∼)	34.35 (+)	86.02 (+)	-65.37 (−)	-9.96 (∼)	76.58 (+)
	3	0.49 (∼)	90.37 (+)	81.86 (+)	-0.08 (∼)	90.32 (+)	81.75 (+)
	4	5.53 (∼)	28.24 (+)	27.90 (+)	-8.91 (∼)	17.27 (+)	16.88 (∼)
Memory	1	2.87 (∼)	47.49 (+)	90.49 (+)	-806.9 (−)	-390.3 (−)	11.22 (+)
	2	-0.85 (∼)	38.05 (+)	99.29 (+)	-107 (−)	-27.19 (−)	98.54 (+)
	3	2.67 (∼)	51.51 (+)	15.20 (+)	-10.51 (∼)	44.94 (+)	3.71 (∼)
	4	-1.07 (∼)	27.19 (+)	93.43 (+)	0.45 (∼)	28.29 (+)	93.52 (+)
Response time	1	3.42 (∼)	49.90 (+)	75.43 (+)	-198 (−)	-54.57 (−)	24.19 (+)
	2	1.69 (∼)	38.92 (+)	15.93 (+)	-17.70 (∼)	26.87 (+)	-0.66 (∼)
	3	5.88 (∼)	31.41 (+)	-1.24 (∼)	-0.83 (∼)	26.52 (+)	-8.46 (∼)
	4	-4.52 (∼)	3.89 (∼)	-3.09 (∼)	4.22 (∼)	11.93 (∼)	5.54 (∼)
Traffic	1	4.73 (∼)	9.49 (∼)	4.16 (∼)	6.58 (∼)	11.25 (∼)	6.02 (∼)
	2	-7.57 (∼)	51.69 (+)	72.41 (+)	-34.82 (−)	39.45 (+)	65.42 (+)
	3	-1.27 (∼)	76.85 (+)	65.00 (+)	-55.34 (∼)	64.49 (+)	46.30 (+)
	4	-3.89 (∼)	-8.75 (−)	24.85 (+)	-1.78 (∼)	-6.55 (∼)	26.37 (+)
Wins		0.0%	81.3%	75.0%	0.0%	56.3%	56.3%
Ties		100.0%	12.5%	25.0%	62.5%	25.0%	43.8%
Loss		0.0%	6.3%	0.0%	37.5%	18.8%	0.0%

6.6 DISCUSSION

Section 6.2 raised research questions that were implicitly discussed throughout all sections. More straightforward answers to these questions are presented in the following:

1. **RQ6.1 - How can MPS be used to forecast microservices performance metrics?** A set of phases demonstrated how to apply MPS to forecast microservices time series (see Sections 6.3 and 6.4). The phases include time series generation and preprocessing, model training, and the generation of model pools, selection, and integration.

2. **RQ6.2 - How effective are homogeneous and heterogeneous pools in increasing the accuracy of forecasting microservices performance metrics?** MPS homogeneous pools are more accurate than heterogeneous ones for forecasting microservices performance metrics. However, heterogeneous pools are formed from a smaller number of models (6) compared to homogeneous ones (10-150). Likewise, the poor accuracy of

particular models (e.g., ARIMA, RF and XGBoost) contributed to decreased heterogeneous pool accuracy, especially in static combination algorithms. Therefore, removing low-accurate forecasting models is highlighted as an extra step for heterogeneous pools before the selection phase (see Section 6.3).

3. **RQ6.3 - How do dynamic and static selection impact the accuracy of homogeneous and heterogeneous pools for predicting microservices performance metrics?** All selection algorithms achieved high accuracy for homogeneous pools, while only Median and DS stand out for heterogeneous pools. The static combination accuracy varied depending on the pool. The Mean was more effective for homogeneous pools, while the Median was more accurate for heterogeneous pools. Nonetheless, it is worth noting that the difference in accuracy between Mean and Median is more significant in heterogeneous pools. For instance, the Median is more accurate for heterogeneous pools than the Mean in 81.25% of the experiments, i.e., 13 out of 16. DS was the selection algorithm most accurate for forecasting real-world series.

4. **RQ6.4 - How effective are MPS and CFA for forecasting microservices performance metrics?** The best homogeneous approach was equal to or more accurate than the BM of the CFA in 100% of the results, i.e., 16 out of 16. On the other hand, compared to the best heterogeneous approach, it was equal to or more accurate than the BM of the CFA in only 62.5% of the cases, i.e., 10 of the 16. Therefore, since the BM of the CFA was unknown before analysis, these results suggest that using a homogeneous MPS approach can enhance the accuracy of the forecast adaptive system and mitigate the CFA problem. At the same time, opting for the heterogeneous one aiming for the highest accuracies is not recommended. Finally, it is essential to highlight the high accuracy achieved by the SVR, which stood out in the study presented.

These findings significantly impact the overall implementation of the PMA tool, especially the MPS module discussed in Chapter 3. They offer valuable insights into the practical applications to be applied and the effectiveness to be expected of the MPS module. The detailed exploration of how MPS can be used to forecast microservices performance metrics (RQ6.1) sheds light on the practical implementation phases involved in applying MPS. This information directly informs the design and implementation of the MPS module within the PMA tool, guiding the development of forecasting methodologies explicitly tailored for microservices environments. Also, comparing homogeneous and heterogeneous pools regarding forecasting accuracy (RQ6.2) highlights the importance of pool composition in achieving accurate predictions. This insight directly influences the selection and integration processes within the MPS module, informing decisions regarding the inclusion or exclusion of specific models based on their performance.

Furthermore, examining the influence of dynamic and static selection algorithms on forecasting accuracy (RQ6.3) helps determine the most appropriate selection algorithms for various pool compositions. This knowledge assists in improving the selection process within the MPS module, ensuring that the most precise forecasting models are chosen for implementation. Lastly, the comparison between the MPS and CFA approaches (RQ6.4) highlights the outperformance of the MPS. This finding emphasises integrating forecasting strategies like MPS into the PMA tool to improve the accuracy and reliability of their predictions. These findings provide valuable insights into the MPS approach and its selection processes, highlighting the potential to improve the performance and efficiency of MBA auto-scaling when integrating the PMA tool.

6.7 CONCLUDING REMARKS

This chapter introduced the MPS, which uses multiple predictors to forecast microservices time series. Its objective is to overcome the CFA limitation, making auto-scaling systems for microservices more reliable and accurate. The chapter begins by discussing the research gap in the literature. After this, the proposed MPS was presented, and the experimental protocol applied was detailed. Finally, an experimental evaluation was presented to assess the performance of the proposed MPS, followed by a discussion of the findings.

7 PMA EXPERIMENTAL EVALUATION

This chapter presents an experimental evaluation of PMA. Section 7.1 outlines the performance evaluation objectives. Next, Section 7.2 details the experimental protocol used in the evaluation. Sections 7.3 and 7.4 present and discuss the evaluation results, respectively. Finally, Section 7.5 discusses how the PMA results correlate to those presented in Chapters 4, 5 and 6.

7.1 OBJECTIVES

The chapter evaluates the effectiveness of PMA proactive strategies for auto-scaling Microservice-based Applications (MBAs). The PMA experimental evaluation objectives are:

- Evaluate the PMA strategies performance to adapt MBAs proactively; and
- Compare and analyse the results obtained in this chapter with those presented in Chapters 4, 5 and 6.

To benchmark the performance of the PMA tool, its proactive strategies are compared with those of Predict Kube (PK). PK (see Section 2.2.2.2) is a widely used industry tool for auto-scaling in K8s environments. The evaluation focuses on assessing the effectiveness of adaptive tools to manage MBAs aiming to maintain customer experience and be cost-effective.

The following research questions guide the PMA experimental evaluation:

- **RQ7.1** - How effective are univariate, multivariate and Multiple Predictor Systems (MPS) strategies for auto-scaling MBAs?
- **RQ7.2** - How effective are PMA and PK self-adaptive tools for auto-scaling MBAs?

7.2 EXPERIMENTAL PROTOCOL

The experimental protocol used to evaluate PMA is similar to the one employed in Chapter 5. However, they have a few differences, such as using real-world workloads and evaluating two new PMA modules: univariate and MPS. Only the crucial details are presented here to prevent redundancy. Nevertheless, the text will guide the reader towards more vital information whenever required.

7.2.1 Setup

All experiments were conducted on a local K8s cluster of six nodes, comprising one master and five workers. Each node, running Ubuntu 22.04 LTS, is a Virtual Machine (VM) equipped with Intel® Xeon® CPU 1220 (four vCpus) and 16GB DDR3 RAM.

The PMA evaluation uses the applications: Daytrader (DAYTRADER, 2024), Online Boutique (OB) (ONLINE-BOUTIQUE, 2024), Quarkus-HTTP-Demo (QHD) (QUARKUS-HTTP-DEMO, 2024), and Travels (TRAVELS, 2024). More details about these applications are presented in Section 5.3.1, as these are the same ones used in Chapter 5.

7.2.2 Proactive Tools

PMA is compared to PK (see Section 2.2.2.2). PMA offers three forecasting strategies: univariate, multivariate, and MPS (see Chapter 3). All of them are considered in this evaluation.

7.2.3 Metrics

The efficiency of an adaptive MBA is commonly measured by its ability to ensure customer experience, such as meeting Service Level Agreement (SLA) requirements, as well as its cost-effectiveness. Therefore, the quality measures proposed by Straesser et al. (2022), which consider cost-effectiveness and customer experience, were used to compare PMA and PK. The metrics adopted for evaluation are Auto-scaling Efficiency (AE_w), Response Time (RT) and Cost (C), defined in Section 5.3.3.

7.2.4 Workloads

The proactive tools are evaluated using four real-world workloads: Alibaba (LUO et al., 2021), ClarkNet (ARLITT; WILLIAMSON, 1996), NASA (ARLITT; WILLIAMSON, 1996) and FIFA WorldCup98 (ARLITT; JIN, 2000). All of them have been used to evaluate self-adaptive auto-scaling tools (TAHIR et al., 2020; ABDULLAH et al., 2021; DANG-QUANG; YOO, 2021; XU et al., 2022; SHIM et al., 2023).

The Alibaba database comprises time series extracted from nearly twenty thousand microservices running over ten thousand bare-metal nodes during twelve hours in 2021. The

Traffic series 2, presented in Section 6.4, was adopted for evaluation and will be referred to as Alibaba.

The ClarkNet (ARLITT; WILLIAMSON, 1996) dataset includes two traces containing all HTTP requests made to the ClarkNet WWW server over two weeks. The first trace was collected from August 28, 1995, at midnight to September 3, 1995, at 11:59 pm, totalling seven days. The second trace was collected from September 4, 1995, at midnight to September 10, 1995, at 11:59 pm, also totalling seven days. The two-week traces account for 3,328,587 requests.

The NASA (ARLITT; WILLIAMSON, 1996) dataset includes two traces containing all HTTP requests made to the NASA Kennedy Space Center WWW server in Florida over two months. The first trace was collected from July 1, 1995, at midnight to July 31, 1995, at 11:59 pm, totalling 31 days. The second trace was collected from August 1, 1995, at midnight to August 31, 1995, at 11:59 pm, also totalling 31 days. The two-month traces account for 3,461,612 requests.

The FIFA WorldCup 98 (ARLITT; JIN, 2000) includes one trace containing all HTTP requests made to the 1998 World Cup website over 88 days. The trace was collected from April 30, 1998 to July 26, 1998. The trace contains 1,352,804,107 requests.

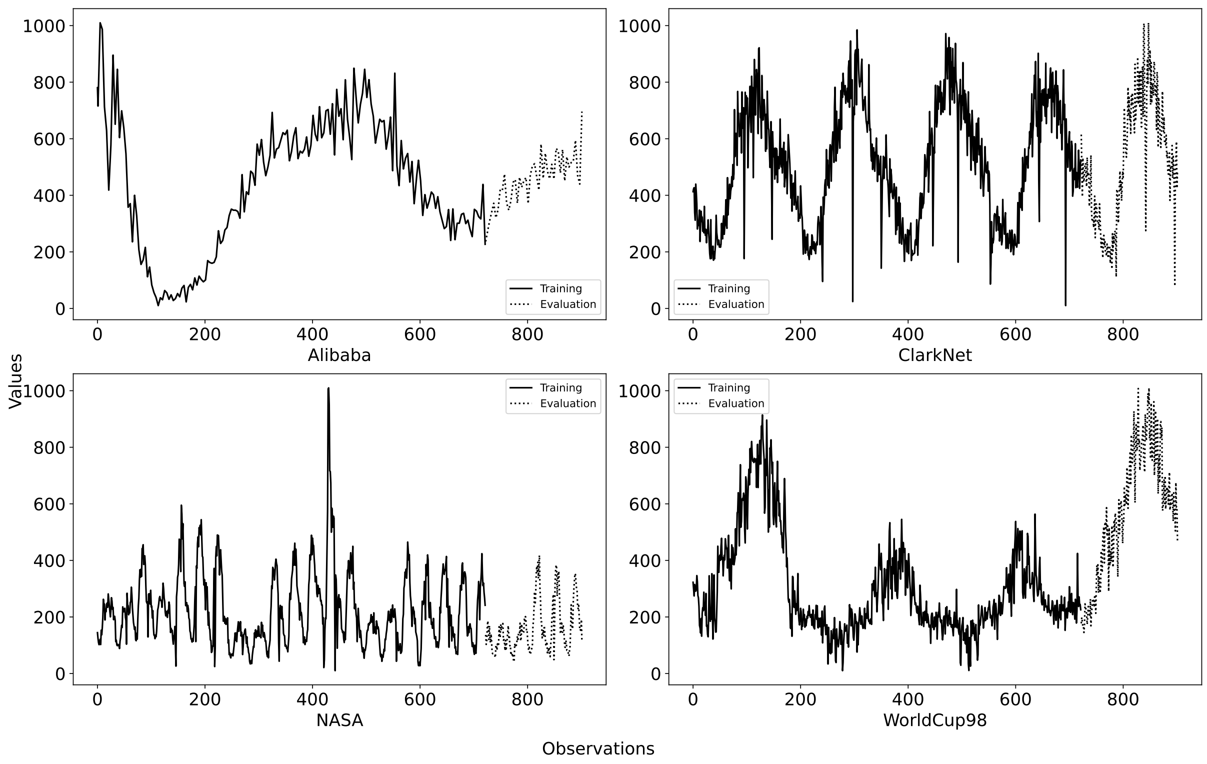
PMA and PK require previous data to operate proactively. Hence, all the traces were processed and transformed into a time series of 15 hours, comprising 900 observations of 1 minute each. The first 12 hours were utilised to generate data to train the forecasting models. The final three hours were dedicated to comparing the tools. The reason for training forecast models with 12 hours of data was discussed in Section 5.3.6. The workloads were emulated using Jmeter (JMETER, 2024), following the same approach described in Section 5.3.4. Figure 7.1 shows all real-world time series.

7.2.5 Training of PMA Forecasting Modules

The min-max normalisation was adopted to scale the series to the interval [0, 1] (SHALABI; SHAABAN; KASASBEH, 2006). Min-max normalisation was defined in Equation 4.1. It was also divided as follows: 70% points for training and the last 30% for testing, as done by Alipour & Liu (2017) and Yadav, Rohit & Yadav (2021). The series was restructured as time windows, each consisting of 20 lags. These steps were used to train the models for three strategies.

The univariate and MPS models were trained according to the proposals presented in

Figure 7.1 – Evaluation workloads.



Chapters 4 and 6. However, the model training for MPS employed a homogeneous pool and Dynamic Selection (DS) due to their performance in Chapter 6. Furthermore, the forecasting model pool comprises 100 models, a commonly used size (YAO; DAI; SONG, 2019). The list of algorithms for both components is the same used in Chapter 6 and includes four Machine Learning (ML) algorithms (Multilayer Perceptron (MLP) (HAYKIN, 2001), Support Vector Regressor (SVR) (DRUCKER et al., 1997), Random Forest (RF) (BREIMAN, 1996), and eXtreme Gradient Boosting (XGBoost) (CHEN; GUESTRIN, 2016)), one statistical algorithm (AutoRegressive Integrated Moving Average (ARIMA) (PRYBUTOK; YI; MITCHELL, 2000)) and one Deep Learning (DL) algorithm (Long Short-Term Memory (LSTM) (MA et al., 2015)).

All algorithms were trained using a grid search approach to find the best hyper-parameters per model. Table 6.2 shows the hyper-parameters used for training the univariate and MPS models and their source. The validation set was used to select the best hyper-parameters per model and the time-sliding window size. The multivariate models were trained using the same approach as the Multivariate Forecasting Tool (MFT) models in Chapter 5. A multivariate LSTM algorithm was utilised with a grid search approach (see in Table 5.2). Also, different features for each application were selected (see Table 5.1). Appendix D summarises the best algorithms for univariate and MPS strategies for each application and workload.

7.2.5.1 Auto-scaling Rules

The experiments were limited to horizontal auto-scaling, a widely used approach for microservices auto-scaling research (SINGH et al., 2019). The scaling mechanism triggers once every 15 seconds with a 5-minute stabilisation cooldown on both tools as done by the Horizontal Pod Auto-scaler (HPA) (HPA, 2024). Due to technical setup restrictions, the maximum number of replicas per containerised application has been limited to ten.

All proactive strategies were configured to predict one step ahead. The forecasting horizon adopted was 1 minute. PK, univariate and MPS were configured to predict CPU usage, and the 80% threshold has been set to trigger adaptation, as done by Lanciano et al. (2021). Multivariate was configured to predict the number of pods, as done by Coulson, Sotiriadis & Bessis (2020).

7.3 RESULTS

This section compares PMA and PK for proactive auto-scaling of MBAs. The research questions presented in Section 7.1 help guide the proposed evaluation analysis. Tables 7.1, 7.2, and 7.3 compare the efficiency ($AE_{0.5}$), cost (C) and response time (P^{95}) of PMA strategies and PK for managing MBAs. Source code, figures, and datasets of this experimental evaluation are available in a public repository[1].

7.3.1 Auto-scaling Efficiency

PMA was more efficient than PK in managing MBAs in almost all experiments (93.75%), except for the QHD application in the ClarkNet workload. PMA MPS was the more efficient approach in 50% of the experiments, i.e., 8 out of 16. Meanwhile, PMA univariate was the best in 43.75% of the experiments, i.e., 7 out of 16. In contrast to the experiments conducted in Chapter 5, the PMA multivariate did not perform well in complex applications, e.g., OB and Travels. Furthermore, it was not the most efficient tool in any experiment.

PMA MPS had the lowest average efficiency (77.08%), followed by PMA univariate (78.49%), PK (80.22%) and PMA multivariate (80.82%). Although the PMA multivariate had the worst auto-scaling efficiency, it was the best for managing the QHD application, with an average

[1] https://github.com/gfads/PMA

Table 7.1 – Auto-scaling efficiency ($AE_{0.5}$) results of PMA and PK managing MBAs. For efficiency, lower values mean better results. The best result for each application and workload is in bold, while the second best is in italics.

Application	Workload	PK	Univariate	Multivariate	MPS
Daytrader	Alibaba	97.13	98.99	*86.40*	**82.07**
	ClarkNet	82.82	**73.84**	*75.56*	79.28
	NASA	*58.47*	**53.48**	69.16	64.53
	WorldCup98	*92.82*	93.41	93.51	**79.96**
QHD	Alibaba	99.59	95.58	*88.91*	**85.97**
	ClarkNet	**44.73**	74.95	49.91	*47.99*
	NASA	80.80	**55.62**	67.82	*63.99*
	WorldCup98	*63.49*	**62.58**	65.08	81.31
OB	Alibaba	*91.65*	92.77	94.55	**91.32**
	ClarkNet	91.85	*87.24*	88.90	**87.09**
	NASA	*85.09*	86.04	87.08	**84.64**
	WorldCup98	88.52	*86.66*	90.25	**84.74**
Travels	Alibaba	92.98	**87.46**	*92.84*	96.46
	ClarkNet	57.76	**51.89**	70.00	*52.51*
	NASA	88.46	88.52	*88.10*	**84.12**
	WorldCup98	*67.38*	**66.89**	85.16	67.41
Average		80.22	*78.50*	80.83	**77.09**

efficiency of 67.93%. PMA MPS had the second-highest efficiency score of 69.82%, while PK got the third position with 72.15%, followed closely by the PMA univariate with 72.18%. PMA MPS was also the best for managing OB and Daytrader, while PMA univariate performed better for Travels. PK was not the most efficient tool in any application. Finally, PMA MPS was more efficient on Alibaba and ClarkNet workloads, while PMA univariate did the same on NASA and WorldCup98.

Costs (C). PMA incurs lower costs than PK in managing MBAs in all experiments. PMA MPS incurs lower costs in 56.25% of the experiments, i.e., 9 out of 16. Meanwhile, the PMA univariate was less costly in 31.25% of the experiments (5 out of 16) and the PMA multivariate one in 12.5%, i.e., 2 out of 16.

PMA MPS had the best cost average (68.94%), followed by PMA univariate (73.74%), PMA multivariate (74.46%), and PK (77.77%). PMA MPS incurred lower costs for Daytrader, Travels, and OB, while PMA multivariate was the best for QHD. Likewise, PMA MPS incurred lower costs for Alibaba, NASA and WorldCup98, while PMA multivariate was the best for ClarkNet.

Table 7.2 – Scaled total cost ($\frac{C}{C_{max}}$) of MBAs managed by PMA and PK. For scaled total cost, lower values mean better results. The best result for each application and workload is in bold, while the second best is in italics.

Application	Workload	PK	Univariate	Multivariate	MPS
Daytrader	Alibaba	98.53	97.97	*74.77*	**66.11**
	ClarkNet	68.51	**52.35**	*53.04*	58.56
	NASA	37.90	**26.41**	64.86	*29.06*
	WorldCup98	*85.64*	91.71	91.53	**66.11**
QHD	Alibaba	*99.17*	**98.90**	*99.17*	*99.17*
	ClarkNet	59.67	*49.91*	61.33	**42.17**
	NASA	61.60	*58.84*	59.12	**54.14**
	WorldCup98	75.51	*71.82*	72.56	**62.62**
OB	Alibaba	89.21	91.02	*89.09*	**87.68**
	ClarkNet	83.70	82.62	*82.53*	**82.07**
	NASA	75.43	*72.08*	77.52	**69.99**
	WorldCup98	79.95	*78.55*	80.51	**76.54**
Travels	Alibaba	93.07	**88.90**	97.04	*92.92*
	ClarkNet	77.75	67.33	**40.00**	*67.22*
	NASA	81.09	*77.04*	77.96	**74.16**
	WorldCup98	77.64	*74.36*	**70.31**	74.44
Average		77.77	*73.74*	74.46	**68.94**

Response Time (RT). PMA decreased response time compared to PK in 68.75% of the experiments, i.e., 11 out of 16. The PK had the lowest response time in 37.25% of the experiments, i.e., 6 out of 16, followed by univariate with the lowest response time in 31.25%, i.e., 5 out of 16. Meanwhile, the PMA MPS had the best response time in 25% of the experiments (i.e., 4 out of 16), and PMA multivariate had the best response time in 6.25%, i.e., 1 out of 16.

PK had the lowest response time average (82.67%), followed by PMA univariate (83.25%), PMA MPS (85.24%), and PMA multivariate (87.19%). PMA MPS had the lowest response time for OB and Daytrader, while PMA univariate and PMA multivariate did the same for Travels and QHD. PMA univariate had the lowest response time for NASA and WorldCup98 workloads. Meanwhile, PMA MPS had the lowest response time for Alibaba, and PK had the lowest for ClarkNet.

Table 7.3 – Scaled normalised response time ($\frac{RT}{RT_{max}}$) of MBAs managed by PMA and PK. For scaled response time, lower values mean better results. The best result for each application and workload is in bold, while the second best is in italics.

Application	Workload	PK	Univariate	Multivariate	MPS
Daytrader	Alibaba	**95.73**	100.00	*98.02*	98.03
	ClarkNet	*97.13*	**95.33**	98.08	100.00
	NASA	*79.03*	80.54	**73.45**	100.00
	WorldCup98	100.00	*95.11*	95.49	**93.81**
QHD	Alibaba	100.00	92.26	*78.66*	**72.77**
	ClarkNet	**29.80**	100.00	*38.49*	53.81
	NASA	100.00	**52.41**	76.53	*73.83*
	WorldCup98	**51.47**	*53.33*	57.60	100.00
OB	Alibaba	**94.09**	*94.52*	100.00	94.96
	ClarkNet	100.00	**91.86**	95.27	*92.12*
	NASA	**94.75**	100.00	*96.64*	99.29
	WorldCup98	97.09	*94.77*	100.00	**92.95**
Travels	Alibaba	92.88	**86.02**	*88.63*	100.00
	ClarkNet	*37.76*	**36.46**	100.00	37.81
	NASA	*95.83*	100.00	98.24	**94.09**
	WorldCup98	**57.12**	*59.41*	100.00	60.38
Average		**82.67**	*83.25*	87.19	85.24

7.4 DISCUSSION

The experiments compared PMA and PK using efficiency, cost, and response time metrics for different applications and workloads. The investigation demonstrated that PMA univariate and PMA MPS were a more efficient proactive system than PK in managing MBAs. PMA univariate and PMA MPS allocates lower resources while maintaining response times similar to PK.

The experiments also showed that each PMA tool has advantages and disadvantages depending on the evaluation measures and applications to be managed. For example, PMA univariate and PMA MPS outperformed PK in efficiency and cost. However, the same is not valid for response time. PMA MPS and PMA univariate are +3.13% and +1.73% more efficient on average than PK, respectively, while PMA multivariate is -0.61% less efficient. Likewise, PMA MPS, PMA univariate and PMA multivariate incur -8.84%, -4.03% and -3.31% lower costs on average than PK. On the other hand, compared to PMA univariate, PMA multivariate and PMA MPS, PK offers a lower average response time of -0.58%, -4.53%, and -2.57%, respectively.

The results also revealed some characteristics of PMA and PK. Firstly, the PK yields a lower response time but has a high allocation cost. PMA univariate provides a similar response time to PK (+0.58% on average) but with a lower allocation cost, i.e., -4.03% on average. Correspondingly, the PMA MPS delivers a higher response time than PK (+2.57% on average) but with a significantly lower allocation cost, i.e., -8.84% on average. Conversely, the PMA multivariate tool provides a higher response time to PK (+4.53% on average) and a lower allocation cost, i.e., -3.31% on average. These results made the PMA MPS the best tool based on efficiency measures.

Also, the effectiveness of PMA MPS and PMA univariate differs depending on the application to be managed. MPS performs better for Daytrader and OB applications, while univariate stands out with Travels. Likewise, even though the PMA multivariate is the worst tool in terms of efficiency, it was the most efficient for managing the QHD application. Thus, it is crucial to tailor auto-scaling strategies to the unique characteristics of each application and workload to optimise efficiency, reduce costs, and improve response times.

Section 7.1 raised research questions that were implicitly discussed throughout this chapter. More straightforward answers to these questions are presented in the following:

- **RQ7.1 - How effective are univariate, multivariate and MPS strategies for auto-scaling MBAs?** The experiments reveal that the most efficient PMA strategies are MPS, univariate, and multivariate, in descending order. MPS is the best strategy because it offers a response time similar to other mechanisms at a significantly lower cost. The univariate strategy is less efficient than MPS but provides better response time at a higher cost. The PMA multivariate was less costly than the PMA univariate but had the worst response time. Even so, the PMA multivariate was the most effective for the QHD application. Thus, depending on self-adaptive system objectives, both PMA MPS and PMA univariate are primary choices due to their higher prediction accuracy and efficiency. However, the results from the QHD application showed that there is at least one scenario where the multivariate strategy proves to be effective. It is up to the PMA configurator to determine which tool best suits their needs.

- **RQ7.2 - How effective are PMA and PK self-adaptive tools for auto-scaling MBAs?** The evaluation showed that PMA was more efficient than PK in managing MBAs in 15 out of 16 experiments (93.75%), except for the QHD application in the

ClarkNet workload. Furthermore, PMA incurs lower costs than PK in all experiments and decreased response time compared to PK in 11 out of 16 (68.75%). Nevertheless, when only the response time is considered, PK performance compared to PMA stands out. PK had a lower average response time than PMA univariate by 0.58%, PMA multivariate by 4.53%, and PMA MPS by 2.57%. However, as mentioned earlier, cost is also critical for adaptive tools. In this regard, PMA univariate outperformed PK by 4.03%, PMA multivariate by 3.31% and PMA MPS by 8.84%. Therefore, it is possible to state that the PMA is an advance in the literature compared to the industry solution (PK), especially when using univariate and MPS strategies.

7.5 GENERAL DISCUSSION

This section analyses and compares all the results presented in this thesis. Specifically, it will examine the findings from Chapters 4, 5, and 5 with those presented in this chapter.

Chapter 4 compared popular ML, DL and statistical algorithms for univariate forecasting of microservices time series. The evaluation results indicated that DL algorithms are not necessarily more accurate than ML algorithms to forecast microservices time series, contrary to common expectations in the literature (ELSAYED et al., 2021). The algorithms with the highest prediction accuracy were SVR and DeepAr, with SVR being chosen as more suitable due to its lower cost, i.e., shorter prediction and training time.

Chapter 5 performed a comparative study comparing univariate and multivariate strategies to proactive auto-scaling MBAs. The study aimed to determine whether the efficiency of the forecasting strategy was correlated with the complexity of the managed application. The results demonstrated that the multivariate strategy was more effective for controlling complex applications like OB and Travels, while the univariate strategy performed better for less complex applications. Furthermore, the results indicated that as the complexity of the environment increases, more than a single feature may be required to estimate the behaviour of the system.

Chapter 6 introduced a MPS to forecast microservices time series using multiple models. The Classical Forecasting Approach (CFA) often falls short as no model can be considered the best for all possible scenarios, as stated in the free lunch theory (YAO; DAI; SONG, 2019). The results revealed that using multiple models to forecast microservice time series significantly increased the accuracy and reliability of the predictor system compared to the CFA.

This chapter compared the proposed proactive, adaptive PMA system to PK, a leading adaptive industry tool. According to the findings, the most effective way to manage MBAs is through the PMA MPS, followed by the PMA univariate and PMA multivariate. Furthermore, PMA MPS and PMA univariate have proven more efficient in managing MBAs than PK.

When analysing the outcomes of all the chapters, it is important to highlight specific points:

- The findings from the univariate evaluation (Chapter 4) are consistent with the results obtained in the MPS evaluation (Chapter 6). Both chapters identified the SVR model as the most effective CFA model. Furthermore, the XGBoost and ARIMA algorithms have exhibited low accuracy.
- The results observed in the MPS evaluation (Chapter 6) are corroborated by the PMA experimental evaluation (Chapter 7). Both chapters indicate that adopting MPS improved the forecast system to auto-scaling MBAs. The proposed MPS was evaluated at project time and then corroborated at runtime. This finding demonstrates that MPS can improve proactive auto-scaling tools for microservices.
- The results observed in the MFT assessment (Chapter 6) were not corroborated by the PMA experimental evaluation (Chapter 7). This chapter evaluation has shown that a multivariate strategy may not always be the optimal solution for managing complex applications. However, it is worth mentioning that the PMA evaluation used different workloads. Furthermore, the comparative study has known limitations described in Section 9.4, which were partially overcome in this evaluation. Based on the PMA evaluation, it was found that the univariate PK strategy performed slightly better than the multivariate PMA strategy. The difference in performance between them is only 0.60%. Thus, a simple univariate strategy can be as effective as a complex multivariate one for the evaluated applications and workloads.

7.6 CONCLUDING REMARKS

This chapter presented the experimental evaluation of the proactive, adaptive system proposed in this thesis. The chapter begins by discussing the experimental evaluation objectives. After this, the experimental protocol used was detailed. Next, the experimental evaluation outcomes were presented and discussed. Finally, the PMA results were contrasted to those in Chapters 4, 5 and 6.

8 RELATED WORK

This chapter provides an overview of the current literature related to the PMA. Section 8.1 describes reactive tools. Section 8.2 covers proactive auto-scaling tools designed for microservices, while Section 8.3 shows proactive works for cloud applications. Section 8.4 presents a comparative analysis of all related work.

8.1 REACTIVE APPROACHES

CAUS (KLINAKU; FRANK; BECKER, 2018) is a heuristic reactive auto-scaling tool for Microservice-based Application (MBA). CAUS regularly checks the microservice workload to identify managed system violations and trigger horizontal auto-scaling adaptations. The heuristic auto-scaling tool was structured as a centralised MAPE-K feedback loop. PMA and CAUS perform a horizontal adaptation of microservices and structures their tool using centralised MAPE-K.

Zhang et al. (ZHANG et al., 2018) presented an adaptive reference model for microservices, which was used as a foundation for building a self-adaptive framework called MSSAF. MSSAF comprises an auto-scaling tool, a translation system, and a managed system. The auto-scaling tool operates over a centralised MAPE-K feedback loop. The translation system interprets adaptation strategies. The managed system is a MBA. PMA and MSSAF are approaches for auto-scaling microservices that aim to maintain Quality of Service (QoS) based on performance.

REMaP (SAMPAIO et al., 2019) was built on a centralised MAPE-K feedback loop and applied concepts from Models@run.time. REMaP uses affinity and resource usage to analyse microservices. *Affinity* is a metric proposed by the author that is defined as the relationship between two microservices given by the number and size of messages exchanged over time. REMaP can identify high-affinity microservices and group them on the same physical server, which reduces the number of servers. REMaP auto-scaling migrates microservices (and their replicas) between cluster nodes, while PMA adapts microservices by horizontally scaling in/out the replicas.

As expected, all reactive approaches differ from PMA as they only act after a problem has occurred.

8.2 MICROSERVICE PROACTIVE APPROACHES

Proactive approaches centred on adapting MBAs are divided according to their prediction strategy: univariate and multivariate.

8.2.1 Univariate

Alipour & Liu (2017) proposed a microservice architecture that combined monitoring, *Time Series Forecasting* (TSF) and auto-scaling. Two models were employed for TSF: Multinomial Logistic Regression (MLR) and Linear Regression (LR). A synthetic dataset generated by ND-Bench[1] was used in the experiments. The dataset corresponds to records of CPU usage every minute for two weeks, i.e., 20160 observations. LR forecasts the CPU usage, and MLR classifies it, e.g., high, medium or low CPU usage. Both forecast and classification are sent to an adaptive logic process which adjusts the MBA, executing horizontal scaling in-out actions. The authors pioneered the integration of TSF and auto-scaling to build a self-adaptive microservice tool. However, they restrict their approach to Machine Learning (ML) algorithms, while PMA also considers Deep Learning (DL) and statistical ones.

Podolskiy et al. (2018) compared several predictive models for microservices time series. The models investigated were Exponential Smoothing (ES), Seasonal Autoregressive Fractionally Integrated Moving Average (SARFIMA), Seasonal Autoregressive Integrated Moving Average (SARIMA), LR, Singular Spectrum Analysis (SSA), and Support Vector Regressor (SVR). Also, models based on AutoRegressive Integrated Moving Average (ARIMA) are extended with Generalised Autoregressive Conditional Heteroskedasticity (GARCH). A real-world time series, available per request and provided by Instana, was adopted for evaluation. The dataset corresponds to response time records every hour for four weeks, i.e., 673 observations. The paper focused on determining the best models that accurately forecast the time series while minimising model training time. The authors showed that SSA and SARIMA models achieved fast and accurate forecasting for predicting microservices time series. They focus on evaluating several forecasting models. Therefore, microservice auto-scaling is out of their scope.

Rossi, Cardellini & Presti (2020) proposed Me-Kube, a K8s extension that introduced a hybrid hierarchical architecture for auto-scaling microservices. The hybrid architecture addressed

1 https://github.com/Netflix/ndbench

the shortcomings of K8s' decentralised auto-scaling tool, i.e., Horizontal Pod Auto-scaler (HPA). The lower level performs decentralised adaptations considering only the state of each element (microservices), while the upper-level coordinates and performs adaptations considering the overall purpose of the application. Me-Kube adaptations can be both reactive and proactive. The ARIMA model was used to forecast the response time of the microservices so that they could adapt proactively. However, no information is provided on the datasets used for model training. The authors showed that Me-kube decreased response time and Service Level Objective (SLO) violations but allocated more pods than the centralised approach of HPA. PMA has three different forecasting strategies evaluated on various workloads; it is implemented over a MAPE-K feedback loop and provides mechanisms to prevent unnecessary adaptations. On the other hand, they implement Me-kube on a hybrid MAPE-K feedback loop, using only the ARIMA. Although they were proactive and achieved better results than HPA, their evaluation was conducted in an experiment with a specific workload, so it may not be generalisable to other scenarios.

Applying a strategy different from the others, RScale (KANG; LAMA, 2020) is a resource auto-scaling tool that provides end-to-end performance guarantees for containerised microservices deployed in the cloud. RScale employs the Gaussian Process Regression, a probabilistic ML performance model. The performance model quickly adapts to dynamic changes in the cloud environment and provides confidence bounds in its forecasts. It can also learn more from less data. A synthetic dataset was extracted from the Online Boutique (OB) application to evaluate the RScale. However, no information about its features, frequency, or size is available. RScale results indicate that it can handle end-to-end tail latency of microservice workloads, even with multi-tenant performance interference and changes in environment dynamics. PMA does not use models that guarantee bounded confidence in the forecasts. However, its Multiple Predictor Systems (MPS) strategy can increase the robustness and accuracy of the adaptive tool.

Like Alipour, Marie-Magdelaine & Ahmed (2020) proposed a proactive auto-scaling framework for MBAs. The framework employs a Long Short-Term Memory (LSTM) model to forecast microservices metrics, such as traffic. It uses the forecast value to adapt horizontally (scaling in/out) and vertically (scaling up/down). The adaptations aim to maintain the end-to-end latency of the applications to fulfil the agreed QoS. LSTM accuracy was evaluated by a real-world traffic time series comprising 600 observations. Also, the framework was evaluated using synthetic traffic series extracted from OB. The results revealed that the auto-scaling framework reduced application latency and increased throughput under flash crowds compared to HPA. PMA proposes a horizontal auto-scaling tool with three prediction strategies but does not apply vertical auto-scaling.

Fontana de Nardin et al. (2021) proposed Elergy, a lightweight elasticity model that auto-scales MBAs focusing on energy. Elergy uses an ARIMA model to forecast CPU usage. It auto-scales containers vertically or migrates them to prevent violations. Four synthetic CPU usage time series considering four distinct workload patterns (decreasing, increasing, wave, and constant) were generated for evaluation. The use of Elergy reduced energy consumption (1.93% to 27.92%) compared to a non-elastic scenario. PMA and Elergy are two proactive microservices auto-scaling tools. However, PMA focuses on resource management, while Elergy focuses on energy consumption.

Toka et al. (2021) introduced a new Kubernetes container engine for auto-scaling microservices. The engine operates either reactively, based on the CPU usage as Kubernetes, or proactively, predicting future traffic through four different models: LSTM, *Auto-Regressive* (AR), Hierarchical Temporal Memory (HTM) and Reinforcement Learning (RL). The engine switches to reactive adaptation if no model reaches a minimum pre-establish accuracy. A traffic time series containing the number of hits to a Facebook page measured every second for two weeks including 604800 observations was adopted in the evaluation. The authors significantly decreased the number of rejected requests of the application (22% to 72%) and had slightly higher resource usage (2% to 9%) compared to HPA. They have enhanced their adaptive tool by utilising three models in their prediction engine. However, since each model is applied separately, the tool is still vulnerable to the Classical Forecasting Approach (CFA) problem.

Kakade et al. (2023) propose a proactive tool for auto-scaling containers. Similar to other works, the forecasted value anticipates future demands and proposed adaptive actions. Two DL models (Bi-directional Long Short-term Memory (Bi-LSTM) and LSTM) were investigated. Both models were evaluated using synthetic CPU usage series. The proposed auto-scaling tool decreased response time by 50% than HPA. They compare only DL models for forecasting microservices series, while PMA also uses statistical and ML models.

8.2.2 Multivariate

Advancing on Alipour and Liu's approach, Coulson, Sotiriadis & Bessis (2020) proposed a self-adaptive pipeline for auto-scaling microservices. The pipeline employed a stacked LSTM model that forecasted future application requests and suggested adaptive recommendations actions. The recommendations were sorted according to their effectiveness by dealing with the forecasted demand. A synthetic time series containing nine features, including request response time, bytes sent, response code, and others, was extracted from a prototype system to validate the pipeline. The authors pioneered integrating multi-step-ahead and multivariate forecasting into a microservice self-adaptive tool. Their work provided a helpful road map for developing, tuning, and evaluating microservice auto-scaling solutions. They focus on long-term multivariate forecasting, while PMA focuses on short-term forecasting.

To address the bottleneck problem in microservices, Huang et al. (2021) proposed a workload prediction approach. The approach is implemented through a Gate Recurrent Unit (GRU), a variation of LSTM. GRU was compared to other popular statistical (ARIMA and ES), ML (K-modes) and DL (Deep Belief Networks (DBN)) models. A multivariate time series with five features (CPU usage, memory, disk usage, network usage and traffic) was synthetically generated for the evaluation. The authors revealed that GRU has lower accuracy than all other models, indicating a promising candidate for the bottleneck problem. They compared different models for bottleneck detection, while PMA is a self-adaptive tool focusing on resource management.

Goli. et al. (2021) developed a new proactive auto-scaling tool for containerised applications. Their tool analyses the forecast CPU usage to trigger horizontal auto-scaling adaptations. However, they also estimate how this adaptation can impact other services. The impact is estimated using a multivariate model that uses the current and forthcoming number of containers, CPU usage and traffic as input. The tool is built based on the MAPE-K framework and incorporates various ML models, such as LR, Random Forest (RF) and SVR. The authors demonstrated that awareness of how a service can impact other services during auto-scaling avoids cascading bottlenecks, improving the overall performance of the application. PMA has a proactive multivariate strategy, allowing the use of different metrics in its prediction. It also has mechanisms to prevent successive adaptations that may harm the stability of the MBA.

Like Podolskiy, Dang-Quang & Yoo (2021) evaluated the forecast accuracy of statistical and DL models to predict short- and long-term traffic time series. The forecast component assists the K8s framework decision-making, allowing proactively horizontally auto-scaling containers. Two DL models (Bi-LSTM and LSTM) and a statistical model (ARIMA) were investigated. The evaluation included two commonly adopted datasets: NASA (ARLITT; WILLIAMSON, 1996) and FIFA World Cup 98 (ARLITT; JIN, 2000). The NASA and FIFA World Cup 98 datasets contain records of HTTP requests over two and three months, respectively. The authors revealed that Bi-LSTM outperformed HPA in managing containers and had a prediction time of 530 to 600 times faster than the ARIMA. They proposed an adaptive tool that includes DL and statistical algorithms. PMA extends this approach by considering traditional ML models.

Qassem et al. (2023) proposed a proactive auto-scaling tool for microservices. The auto-scaler is built on a centralised MAPE-K feedback loop and uses an RF model to forecast CPU usage and memory. The auto-scaler uses the forecast value to adapt horizontally (scaling in/out) and vertically (scaling up/down). They evaluated the RF using the FastStorage (FASTSTORAGE, 2024) dataset containing workload traces of 1,250 Virtual Machines (VMs). The auto-scaler was evaluated using a proposed synthetic experiment with the Sock-Shop[2] application. Their results show an improvement in application response time (60%) and tail latency (19.6×). PMA performs only horizontal auto-scaling but has three distinct forecasting strategies.

8.3 CLOUD PROACTIVE APPROACHES

Nikravesh, Ajila & Lung (2017) used TSF to minimise Service Level Agreement (SLA) violations in cloud applications. Their work is an alternative to the reactive adaptation, whose deficiency is neglecting the startup time of VM. The autoscaling strategy uses workload forecasting for decision-making. They use SVR or Multilayer Perceptron (MLP) models for prediction. A decision fusion technique selects the most appropriate model during run-time based on the current workload pattern. The main finding of the research is that the prediction accuracy of the models is positively impacted by using different prediction algorithms for the different cloud workload patterns. This thesis has also corroborated this finding for predicting microservices time series. PMA uses two other classes of forecasting models (DL and statistical algorithms) and focuses on microservices (not on cloud applications). Also, it implements the MAPE-K feedback loop.

Palmerino et al. (2019) focused on the evolution of corrective actions applied to cloud applications over time. Adaptive solutions usually consider that the cost and time required to take corrective action are always static. However, these factors can fluctuate continuously depending on the managed system environment. For instance, network congestion could cause data transmission to take longer than anticipated. The authors proposed two forecasting models

[2] https://github.com/microservices-demo/microservices-demo

(ARIMA and Multiple Regression Analysis (MRA)) to predict the future state of the application. ARIMA predicts when an application objective defined in the SLA can be violated, while MRA predicts the cost and time of each corrective action. PMA uses additional forecasting models and focuses on adapting microservices. They do not use the MAPE-K feedback loop. However, their two-model approach enables the best corrective action to be chosen based on the future application state.

Shim et al. (2023) proposed a proactive auto-scaling framework based on MAPE-K for cloud applications. It uses the forecast value to propose adaptive actions. Two DL models (Bi-LSTM and LSTM), a statistical model (ARIMA) and a Transformer (Informer) were investigated. The evaluation included two datasets: NASA (ARLITT; WILLIAMSON, 1996) and FIFA World Cup 98 (ARLITT; JIN, 2000), as done by Dang-Quang & Yoo (2021). The authors demonstrated that Transformers perform well when forecasting time series of cloud applications. PMA also demonstrated that transformers perform well in forecasting microservices series. However, PMA focuses on microservices rather than on cloud applications.

8.4 SUMMARY

The literature highlights several approaches to auto-scaling in microservices and cloud applications, focusing on maintaining QoS through reactive and proactive strategies. Common characteristics include:

The need to adapt microservices. The dynamic nature of microservices and their execution environment demands adaptation. Microservices are complex distributed applications that require monitoring and adaptation. Both reactive and proactive auto-scaling have been applied. Reactive auto-scaling has been implemented to optimise deployment, enhance performance, and prevent cascading failure (ANGELOPOULOS et al., 2016; FLORIO; NITTO, 2016; KLINAKU; FRANK; BECKER, 2018; SAMPAIO et al., 2019). Proactive auto-scaling has been applied to improve resource and energy management for microservices and VMs, bottleneck detection, and mitigation of SLO violations (AJILA; BANKOLE, 2016; ALIPOUR; LIU, 2017; NIKRAVESH; AJILA; LUNG, 2017; HUANG et al., 2021; TOKA et al., 2021). Maintaining application performance at run-time is a common goal in these adaptive tools.

Proactive adaptation features. Most papers focus on monitoring/predicting performance metrics such as CPU usage, memory, response time, and traffic metrics. Also, most proactive adaptive microservices tools use DL (COULSON; SOTIRIADIS; BESSIS, 2020; MARIE-

MAGDELAINE; AHMED, 2020), ML ((ALIPOUR; LIU, 2017; GOLI. et al., 2021)), and statistical learning algorithms ((PODOLSKIY et al., 2018; ROSSI; CARDELLINI; PRESTI, 2020)) for TSF. Furthermore, applying TSF algorithms is the most popular method for forecasting microservices.

Emphasis on QoS. Many studies focus on maintaining or enhancing QoS. Proactive approaches aim to predict changes in demand to prevent any possible QoS decline before it happens. On the other hand, reactive approaches are based on current workload data and aim to respond quickly to sudden spikes in demand to maintain service quality.

Use of MAPE-K Feedback Loops. Many discussed tools, such as CAUS (KLINAKU; FRANK; BECKER, 2018), REMaP (SAMPAIO et al., 2019) and Me-kube (ROSSI; CARDELLINI; PRESTI, 2020), rely on the MAPE-K (Monitor, Analyse, Plan, Execute over shared Knowledge) as a reference model for building their adaptive solutions.

Datasets for training. The effectiveness of a proactive auto-scaling strategy heavily relies on the quality and characteristics of the datasets used for training prediction models. These datasets range from synthetic benchmarks to real-world time series.

The current trend in related work is focused on developing more intelligent and predictive scaling mechanisms that can adjust to changing workloads with minimal human intervention. This trend aims to ensure high levels of service quality in cloud and microservices dynamic environments. The PMA was developed to help advance this objective. Table 8.1 summarises all the discussed related works by specific characteristics.

Table 8.1 – Related work summary.

Author	Performance Metrics	Forecasting Classes	TSF Strategy	Validation	Competences Evaluated	MAPE-K
Alipour et al. (2017)	C	ML	U	S	A, FIT	✗
Nikravesh et al. (2017)	T	ML	U	S	A	✗
Podolskiy et al. (2018)	T	S, ML	U	R	A, FIT	✗
Klinaku et al. (2018)	MQ	-	-	-	-	✓
Zhang et al. (2018)	C	-	-	-	-	✓
Sampaio et al. (2019)	C, M, ME	-	-	-	-	✓
Palmerino et al. (2019)	L, CO	S, ML	U	S	A	✗
Coulson et al. (2020)	RT, BS, RC, URL, NP	S, DL	M	S	A	✗
Rossi et al. (2020)	T	S	U	Unknown	A	✓
Kang et al. (2020)	C, T, NU	ML	U	S	A	✗
Marie-Magdelaine et al. (2020)	T	DL	U	S,R	-	✓
Fontana et al. (2021)	C	S	U	S	A	✗
Dang et al. (2021)	T	S,DL	M	R	A, FOT	✓
Huang et al. (2021)	C, M, D, N	S, DL	M	S	A	✗
Toka et al. (2021)	T	S, DL	U	R	A	✗
Goli et al. (2021)	C	ML	M	S	A	✓
Kakade et al. (2023)	C	DL	U	S	A	✗
Shim et al. (2023)	T	DL, S	U	R	A	✓
Al-Qassem et al. (2023)	C, M, D, N	ML	U	R	A	✓
PMA	**ACP, C, DCP, HM, M, NP, PCGS, RT, T, TP, TCGS, GJVMO**	**ML, DL, S**	**U, M, MPS**	**S,R**	**A, FIT, FOT**	✓

Performance Metrics: Application Connection Pool (ACP), Bytes Sent (BS), CPU Usage (C), Disk I/O (D), Database Connection Pool (DCP), Latency (L), Global JVM Operations (GJVMO), Heap Memory (HM), Memory (M), Messages Exchanges (ME), Message Queue (MQ), Network I/O (N), Number of Pods (NP), Per-second GC scavenge (PCGS) Response Code (RC), Response Time (RT), Thread Pool (TP), Traffic (T), and Total GC Scavenge (TCGS).
Forecasting classes: Deep Learning (DL), Machine Learning (ML), Statistical (S).
TSF Strategy: Univariate (U), Multivariate (M), Multiple Predictor Systems (MPS).
Validation: Synthetic Series (S) and Real-world Series (R).
Competences Evaluated: Accuracy (A), Fitting Time (FIT), and Forecasting Time (FOT).

8.5 CONCLUDING REMARKS

This chapter presented existing related works similar to the proposed approach. Initially, several reactive auto-scaling tools were introduced. Next, proactive auto-scaling tools for microservices and cloud applications were examined. Finally, the related works were summarised and deliberated.

9 CONCLUSION AND FUTURE WORK

This chapter presents the conclusion of this thesis. Section 9.1 presents the final remarks. Section 9.2 answers the research questions presented in Chapter 1. Section 9.3 presents the contributions of this thesis. Section 9.4 presents some limitations of the proposed approach. After this, Section 9.5 finishes debating ways to extend future work.

9.1 THESIS REMARKS

Although several studies propose proactive auto-scaling tools for microservices, they have design limitations that may negatively impact the forecast accuracy at run-time. This thesis has delved into current forecasting approaches and strategies to enhance proactive Microservice-based Application (MBA) adaptation. As a result, this thesis introduced PMA (**P**roactive **Mi**croservices **A**uto-scaler), a generic, holistic, adaptive and proactive tool that applies horizontal auto-scaling actions to managing MBA. PMA is a multi-model self-adaptive tool that employs different learning algorithms and forecasting strategies.

PMA is built upon the MAPE-K feedback loop, which consists of four phases: monitoring, analysis, planning and execution. The monitor fetches and processes performance metrics from the microservices and makes them available to other feedback loop components. The Analyser is designed to proactively assess if the resources for each microservice assigned are enough to fulfil their predicted needs. To accomplish this, it predicts the required performance metrics for each service, employing one of three forecasting modules: univariate, multivariate, or Multiple Predictor Systems (MPS). PMA univariate uses one performance metric for prediction, while PMA multivariate uses multiple ones. PMA MPS employs multiple models, which can be univariate or multivariate. PMA MPS main idea is to increase the accuracy and reliability of predictions by combining various forecasting models. These forecasting strategies employed learning algorithms from the three most prominent predicting classes for microservices, i.e., Machine Learning (ML), Deep Learning (DL) and statistical.

This feedback loop comprises four phases that enable the autonomous management of microservices. These phases include **m**onitoring, **a**nalysis, **p**lanning, and **e**xecution. Furthermore, a knowledge base is integrated throughout all phases to store relevant information for the adaptation process.

The planner determines which adaptive actions should be applied to ensure the stability of the managed system. It can propose horizontal auto-scaling actions, e.g., scaling in/out of replicas. These actions address violations found during analysis, such as microservices with too few or too many replicas. After the planner creates adaptation plans, the executor carries them out.

The PMA has the potential to:

- **Improve MBA performance.** Unlike reactive tools that scale resources after performance issues have arisen, PMA can adapt resources in advance by forecasting future resource needs, ensuring that microservices can handle incoming workloads without performance degradation.
- **Improve cost efficiency.** The enhanced forecasting allows PMA to reduce resource wastage, and thus costs, associated with over-provisioning and decrease response times by ensuring that adequate resources are always available.
- **Improve forecasting strategies for self-adaptive tools.** PMA employs various forecasting strategies, such as univariate, multivariate, and MPS. It is a highly flexible self-adaptive tool that harnesses the strengths of each forecasting strategy, whether they focus on a single performance metric, multiple metrics or employ multiple models. As a result, PMA better adapts to changes at runtime.

9.2 RESEARCH QUESTIONS

Chapter 1 raised research questions that were implicitly discussed throughout all chapters. More straightforward answers to these questions are presented in the following:

RQ1.1 - Which algorithms and prediction techniques are the more accurate and cost-effective for microservices forecasting, and how can these findings be applied to enhance proactive tools?

Several findings were learned from evaluating different forecasting algorithms and strategies:

The accuracy of the forecasting models depends on the time series to be predicted. Proactive tools can use dynamic selection mechanisms to improve the accuracy of forecasting algorithms based on the nature of the data to be predicted. The dynamic selection mechanism could act into a preliminary analysis phase where various algorithms are tested on a subset of the data to determine which provides the best accuracy for the specific time series.

ML algorithms have lower predicting and fitting times and are as accurate as complex DL models. Proactive tools, especially those operating under budget constraints or in pay-as-you-go environments like microservices, can prioritise using ML algorithms to optimise accuracy and cost.

Standard algorithms. When building proactive tools for microservices where a thorough analysis is not feasible due to financial or time constraints, algorithms like Support Vector Regressor (SVR) and DeepAR can be suitable due to their high performance. However, the predicted time series should resemble those used in this thesis. These algorithms have demonstrated high accuracy and can assist self-adaptive tools in achieving better resource management.

Univariate vs Multivariate. Proactive tools should consider the complexity of the application before deciding on the forecasting approach. A univariate approach might suffice for more non-complex applications, but shifting to a multivariate approach can yield better forecasting results as the application complexity increases. Thus, proactive tools should be created to adapt their forecasting strategies based on continuous evaluations of application complexity and performance demands. It may involve implementing mechanisms to periodically reassess the characteristics of the application and modify the forecasting approach accordingly.

Not all time series can be accurately modelled using the same algorithm. Proactive tools can improve by incorporating mechanisms to evaluate and switch between algorithms based on ongoing performance assessments. The adaptive approach allows the managed system to maintain high accuracy across distinct scenarios, as is standard in MBAs.

These findings can guide the design and enhancement of proactive tools, making them more accurate, cost-effective, and adaptable to the needs of the applications they manage. A flexible and dynamic forecasting tool can significantly improve the ability of the managed system to respond to unexpected events effectively.

RQ1.2 - Can the proactive auto-scaling of MBAs be improved by delegating the forecasting task to several models? The experiments showed that the proactive auto-scaling of MBAs can be improved by delegating the forecasting task to several models. During the MPS evaluation (Chapter 6), the MPS showed to be as accurate as or more accurate than the Classical Forecasting Approach (CFA) in all the results (16 out of 16). In the PMA assessment, the MPS strategy was more efficient than the Predict Kube (PK) in 75% of the results

(12 out of 16). These findings suggest that adopting MPS significantly enhances the proactive auto-scaling of microservices. The MPS approach can enhance the accuracy and robustness of predictive tools. As a result, these better-managed applications can experience savings in operating costs and high availability.

9.3 SUMMARY OF CONTRIBUTIONS

The main contribution of this thesis is the PMA, a MAPE-K-based self-adaptive tool designed for the proactive adaptation of MBAs. PMA has three forecasting modules: univariate, multivariate, and MPS. These modules can forecast performance metrics and, based on this estimation, decide whether an adaptation is needed. If necessary, the adaptation consists of executing scaling-in/out actions. PMA is the first self-adaptive proactive tool to offer three different forecasting modules. Likewise, it is the first tool to use MPS for microservices time series.

Other contributions of this thesis are as follows:

- **An analysis of popular learning algorithms for forecasting microservices time series.** It compared ten known learning algorithms from the three most prominent predicting classes, i.e., ML, DL and statistical. The analysis provided a comparative insight into the accuracy, fit, and forecasting time of the algorithms. It aimed to guide the selection of learning algorithms for designing new adaptive and proactive microservice tools. The evaluation considers not only accuracy but also the fit and forecasting time of the algorithms. However, since no measure of competence computes model fitting and forecasting times, this thesis proposes a new competence measure called Model Effort Time (MET) to address this gap. This analysis presented the first comparison of well-known learning algorithms from three main predicting classes for forecasting microservices performance metrics.
- **A comparative study focusing on the proactive auto-scaling of microservices, evaluating univariate and multivariate forecasting strategies.** The study compared PK with Multivariate Forecasting Tool (MFT) using four popular open-source applications across different forecasting horizons. The MFT was integrated into the PMA as a module, and it was referred to as the PMA multivariate. This comparative study is the

first to evaluate univariate and multivariate for proactive auto-scaling of microservices at run-time.

- **A MPS for predicting microservices performance metrics** This approach involves selecting the most suitable forecasters from a pool of models to enhance prediction accuracy and reliability. The proposed MPS is the first strategy that uses multiple predictors as an alternative to improve the accuracy and reliability of microservices forecasting tools.

The PMA development and implementation is a step towards the auto-scaling of microservices. Furthermore, the comparative analysis and study and the proposed MPS offer valuable insights into selecting forecasting algorithms and optimising auto-scaling strategies. These contributions paved the way for building more resilient, flexible, accurate, and efficient proactive microservices auto-scaling tools, establishing PMA as a new benchmark for proactive auto-scaling of microservices.

9.4 RESEARCH LIMITATIONS

Some limitations in this thesis are noteworthy and should be highlighted:

Chapter 4. The analysis has not included multivariate time series forecasting, so the results cannot be generalised for them. Although AutoRegressive Integrated Moving Average (ARIMA) is a commonly used statistical algorithm, the evaluation has not considered other algorithms. Also, the evaluation was restricted to time series data belonging to a single database, specifically the Alibaba database.

Chapter 5. The results of comparing two auto-scaling techniques cannot be generalised. PK is a stable proactive auto-scaling K8s tool, and MFT was designed following the pattern of existing approaches (COULSON; SOTIRIADIS; BESSIS, 2020; GOLI. et al., 2021; MOHAMED; EL-GAYAR, 2021). However, a more comprehensive range of alternatives should be considered to draw more generalised findings. Furthermore, the workload used in the assessment is simplified. Modelling a periodic workload may be a simple task for predictive algorithms. Finally, although Long Short-Term Memory (LSTM) is widely used for forecasting microservices performance metrics, many other algorithms are available.

Chapter 6. The outperformance of MPS over CFA have not considered cost. As MPS contains several models, its adoption costs more than the CFA. Therefore, since cost is a critical factor for MBAs, the financial implications of using MPS must be considered. Furthermore, the proposal has not assessed its scalability. Managing multiple models is significantly more complex than managing a single one. Therefore, incorporating MPS poses challenges for PMA configurator, such as determining which microservices to manage, establishing the life cycle of the pool models, and deciding on the appropriate pool size.

Chapter 7. Because the PMA components match those proposed in other chapters, it shares similar limitations as previously described. Additional limitations are also present:

- Although ARIMA is a commonly employed statistical algorithm, the evaluation did not explore other statistical algorithms.
- The LSTM is widely used for multivariate prediction of microservices performance metrics. However, many other multivariate algorithms are available.
- The PK and PMA comparison cannot be generalised. Although PK is a leading adaptive industry tool, more precise conclusions require evaluating PMA against other industrial and academic tools.
- The scalability cost of the proposal has not been evaluated. Managing multiple models in PMA MPS is far more complex than managing a single model, as practised in PMA univariate.
- The cost evaluation only considered the expenses of running the managed system, not those of running PMA and PK. It would be helpful to determine the memory and CPU usage required to run them.

9.5 FUTURE WORKS

Building on the research limitations and findings presented, here are several ideas for future work that could further advance the field of proactive auto-scaling tools for microservices:

PMA as a self-adaptive tool. Adding more capabilities into the PMA, such as self-healing and self-protection, may be worth considering to provide a more comprehensive approach to MBA management. Reinforcement Learning can also be a significant step in enhancing the decision-making effectiveness in the PMA planning phase. Reinforcement Learning can help PMA learn optimal scaling actions through trial and error, resulting in fewer conflicting actions with managed system objectives. Also, incorporating cost metrics into the decision-making process can improve cost efficiency and promote sustainable cloud practices.

PMA as a forecasting tool. Developing techniques for run-time model retraining to handle microservices' life-cycle evolution and changes in time series behaviour may be worth considering to provide a more robust forecasting approach. These techniques should include mechanisms for continuous monitoring and automatic detection of significant shifts in data patterns, triggering retraining processes for forecasting models to maintain or enhance accuracy. Transfer learning could also be beneficial, allowing forecasting models to leverage knowledge from previously encountered scenarios to adapt to new patterns efficiently. Anomaly detection algorithms can also assist in identifying and isolating atypical behaviour in microservices, ensuring the robustness of PMA against unstable data and improving its forecasting accuracy. Another PMA condition that could be enhanced is exploring other statistical, ML, and DL algorithms to improve the accuracy of PMA forecasting and resource utilisation. These forecasting models could be tested under different workloads and operational conditions to determine the most effective strategies for specific scenarios.

PMA is a robust and resilient tool. It would be helpful to evaluate the scalability and performance of the PMA under extreme conditions, such as sudden spikes in demand or hardware failures. Such an evaluation is crucial for several reasons, including identifying scalability limits, ensuring reliability, improving resilience, optimising performance, and planning disaster recovery.

PMA into different domains. It could be helpful to look into integrating PMA with other technologies like serverless and edge computing. Furthermore, it is essential to explore how well it can adapt and perform in different domains, such as healthcare, finance, and the Internet of Things. Such explorations will help to identify domain-specific challenges and opportunities for proactive auto-scaling. It is also necessary to enhance PMA capabilities for multi-cloud and cross-cloud support, develop strategies to manage microservices across multiple cloud providers and optimise resource allocation in a heterogeneous cloud environment.

Improving proactive auto-scaling for microservices is a complex task involving solving technological and application-specific challenges. The journey towards more sophisticated and robust auto-scaling tools is ongoing, and these proposed directions for future work represent promising steps forward in optimising run-time proactive MBA management.

REFERENCES

ABDULLAH, M.; IQBAL, W.; ERRADI, A. Unsupervised learning approach for web application auto-decomposition into microservices. *Journal of Systems and Software*, v. 151, p. 243–257, 2019. ISSN 0164-1212.

ABDULLAH, M.; IQBAL, W.; MAHMOOD, A.; BUKHARI, F.; ERRADI, A. Predictive autoscaling of microservices hosted in fog microdata center. *IEEE Systems Journal*, v. 15, n. 1, p. 1275–1286, 2021.

ADHIKARI, R.; VERMA, G.; KHANDELWAL, I. A model ranking based selective ensemble approach for time series forecasting. *Procedia Computer Science*, v. 48, p. 14–21, 2015. ISSN 1877-0509. International Conference on Computer, Communication and Convergence (ICCC 2015).

AHMED, N. K.; ATIYA, A. F.; GAYAR, N. E.; EL-SHISHINY, H. An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews*, Taylor & Francis, v. 29, n. 5-6, p. 594–621, 2010.

AJILA, S.; BANKOLE, A. Using machine learning algorithms for cloud client prediction models in a web vm resource provisioning environment. *Transactions on Machine Learning and Artificial Intelligence*, v. 4, 02 2016.

AL-MASRI, E. Enhancing the microservices architecture for the internet of things. In: *IEEE International Conference on Big Data (Big Data)*. [S.l.: s.n.], 2018. p. 5119–5125.

ALIPOUR, H.; LIU, Y. Online machine learning for cloud resource provisioning of microservice backend systems. In: *IEEE International Conference on Big Data*. Boston, MA, USA: IEEE, 2017. p. 2433–2441.

ANGELOPOULOS, K.; Papadopoulos, A. V.; Souza, V. E. S.; Mylopoulos, J. Model predictive control for software systems with cobra. In: *2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. Austin, TX, USA: IEEE, 2016. p. 35–46.

ARLITT, M.; JIN, T. A workload characterization study of the 1998 world cup web site. *IEEE Network*, v. 14, n. 3, p. 30–37, 2000.

ARLITT, M. F.; WILLIAMSON, C. L. Web server workload characterization: The search for invariants. In: *Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. New York, NY, USA: Association for Computing Machinery, 1996. (SIGMETRICS '96), p. 126–137. ISBN 0897917936. Disponível em: <https://doi.org/10.1145/233013.233034>.

ARMSTRONG, J.; COLLOPY, F. Error measures for generalizing about forecasting methods: Empirical comparisons. *International Journal of Forecasting*, v. 8, n. 1, p. 69–80, 1992. ISSN 0169-2070.

BAMISILE, O.; OLUWASANMI, A.; EJIYI, C.; YIMEN, N.; OBIORA, S.; HUANG, Q. Comparison of machine learning and deep learning algorithms for hourly global/diffuse solar radiation predictions. *International Journal of Energy Research*, v. 46, n. 8, p. 10052–10073, 2022.

BARROW, D. K.; CRONE, S. F. A comparison of adaboost algorithms for time series forecast combination. *International Journal of Forecasting*, v. 32, n. 4, p. 1103–1119, 2016. ISSN 0169-2070.

BELETE, D. M.; HUCHAIAH, M. D. Grid search in hyperparameter optimization of machine learning models for prediction of hiv/aids test results. *International Journal of Computers and Applications*, Taylor & Francis, v. 44, n. 9, p. 875–886, 2022.

BERNSTEIN, D. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, v. 1, n. 3, p. 81–84, Sep. 2014. ISSN 2372-2568.

BOX, G.; JENKINS, G.; REINSEL, G.; LJUNG, G. *Time Series Analysis: Forecasting and Control*. Hoboken, New Jersey: Wiley, 2015. (Wiley Series in Probability and Statistics). ISBN 9781118674925.

BREIMAN, L. Bagging predictors. *Machine Learning*, v. 24, n. 2, p. 123–140, Aug 1996. ISSN 1573-0565.

CHANG, C.-C.; LIN, C.-J. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, Association for Computing Machinery, New York, NY, USA, v. 2, n. 3, maio 2011. ISSN 2157-6904.

CHEN, T.; GUESTRIN, C. Xgboost: A scalable tree boosting system. In: *The 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. NY, USA: ACM, 2016. p. 785–794. ISBN 978-1-4503-4232-2.

CORNEJO-BUENO, L.; CASANOVA-MATEO, C.; SANZ-JUSTO, J.; SALCEDO-SANZ, S. Machine learning regressors for solar radiation estimation from satellite data. *Solar Energy*, v. 183, p. 768–775, 2019. ISSN 0038-092X.

COULSON, N. C.; SOTIRIADIS, S.; BESSIS, N. Adaptive microservice scaling for elastic applications. *IEEE Internet of Things Journal*, v. 7, n. 5, p. 1–1, 2020.

DA, K.; DALMAU, M.; ROOSE, P. A Survey of adaptation systems. *International Journal on Internet and Distributed Computing Systems*, v. 2, n. 1, p. 1–18, nov. 2011.

DANG-QUANG, N.-M.; YOO, M. Deep learning-based autoscaling using bidirectional long short-term memory for kubernetes. *Applied Sciences*, v. 11, n. 9, 2021. ISSN 2076-3417.

DANG-QUANG, N.-M.; YOO, M. An efficient multivariate autoscaling framework using bi-lstm for cloud computing. *Applied Sciences*, v. 12, n. 7, 2022. ISSN 2076-3417.

DAYTRADER. *Sample.daytrader8 Repository*. 2024. <https://github.com/OpenLiberty/sample.daytrader8>. Online; accessed May 2024.

de Amorim, L. B.; CAVALCANTI, G. D.; CRUZ, R. M. The choice of scaling technique matters for classification performance. *Applied Soft Computing*, v. 133, p. 109924, 2023. ISSN 1568-4946.

de Mattos Neto, P. S.; MADEIRO, F.; FERREIRA, T. A.; CAVALCANTI, G. D. Hybrid intelligent system for air quality forecasting using phase adjustment. *Engineering Applications of Artificial Intelligence*, v. 32, p. 185–191, 2014. ISSN 0952-1976.

de Oliveira, J. F. L.; PACíFICO, L. D. S.; de Mattos Neto, P. S. G.; BARREIROS, E. F. S.; RODRIGUES, C. M. de O.; FILHO, A. T. de A. A hybrid optimized error correction system for time series forecasting. *Applied Soft Computing*, v. 87, p. 105970, 2020. ISSN 1568-4946.

DEMVSAR, J. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, JMLR.org, v. 7, p. 1–30, dec 2006. ISSN 1532-4435.

DESAI, A.; OZA, R.; SHARMA, P.; PATEL, B. Hypervisor: A survey on concepts and taxonomy. *International Journal of Innovative Technology and Exploring Engineering*, Citeseer, v. 2, n. 3, p. 222–225, 2013.

DIEBOLD, F. X.; MARIANO, R. S. Comparing predictive accuracy. *Journal of Business & Economic Statistics*, Taylor & Francis, v. 20, n. 1, p. 134–144, 2002.

DING, J.; CAO, R.; SARAVANAN, I.; MORRIS, N.; STEWART, C. Characterizing service level objectives for cloud services: Realities and myths. In: *International Conference on Autonomic Computing*. Umea, Sweden: IEEE, 2019. p. 200–206.

DRUCKER, H.; BURGES, C. J. C.; KAUFMAN, L.; SMOLA, A. J.; VAPNIK, V. Support vector regression machines. In: MOZER, M. C.; JORDAN, M. I.; PETSCHE, T. (Ed.). *Advances in Neural Information Processing Systems*. Denver, CO, USA: MIT Press, 1997. p. 155–161.

EHLERS, R. S. Análise de séries temporais. *Universidade Federal do Paraná*, 2007.

ELSAYED, S.; THYSSENS, D.; RASHED, A.; SCHMIDT-THIEME, L.; JOMAA, H. S. Do we really need deep learning models for time series forecasting? *CoRR*, abs/2101.02118, 2021.

ESPINOSA, R.; PALMA, J.; JIMéNEZ, F.; KAMIńSKA, J.; SCIAVICCO, G.; LUCENA-SáNCHEZ, E. A time series forecasting based multi-criteria methodology for air quality prediction. *Applied Soft Computing*, v. 113, p. 107850, 2021. ISSN 1568-4946.

FASTSTORAGE. *The Grid Workloads Archive*. 2024. <http://gwa.ewi.tudelft.nl/datasets/gwa-t-12-bitbrains>. Online; accessed May 2024.

FLORIO, L.; NITTO, E. D. Gru: An approach to introduce decentralized autonomic behavior in microservices architectures. In: *2016 IEEE International Conference on Autonomic Computing*. Wurzburg, Germany: IEEE, 2016. p. 357–362.

Fontana de Nardin, I.; da Rosa Righi, R.; Lima Lopes, T. R.; André da Costa, C.; YEOM, H. Y.; KöSTLER, H. On revisiting energy and performance in microservices applications: A cloud elasticity-driven approach. *Parallel Computing*, v. 108, p. 102858, 2021. ISSN 0167-8191.

FRIEDMAN, M. A Comparison of Alternative Tests of Significance for the Problem of m Rankings. *The Annals of Mathematical Statistics*, Institute of Mathematical Statistics, v. 11, n. 1, p. 86 – 92, 1940.

GALANTINO, S.; IORIO, M.; RISSO, F.; MANZALINI, A. Raygo: Reserve as you go. In: *Intl Conf on Dependable, Autonomic and Secure Computing*. AB, Canada: IEEE, 2021. p. 269–275.

GAN, Y.; ZHANG, Y.; CHENG, D.; SHETTY, A.; RATHI, P.; KATARKI, N.; BRUNO, A.; HU, J.; RITCHKEN, B.; JACKSON, B.; HU, K.; PANCHOLI, M.; HE, Y.; CLANCY, B.; COLEN, C.; WEN, F.; LEUNG, C.; WANG, S.; ZARUVINSKY, L.; ESPINOSA, M.; LIN, R.; LIU, Z.; PADILLA, J.; DELIMITROU, C. An open-source benchmark suite for microservices and their hardware-software implications for cloud; edge systems. In: *ASPLOS '19: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA: Association for Computing Machinery, 2019. (ASPLOS '19), p. 3–18. ISBN 9781450362405.

GAN, Y.; ZHANG, Y.; HU, K.; CHENG, D.; HE, Y.; PANCHOLI, M.; DELIMITROU, C. Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices. In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA: Association for Computing Machinery, 2019. (ASPLOS '19), p. 19–33. ISBN 9781450362405.

GOLI., A.; MAHMOUDI., N.; KHAZAEI., H.; ARDAKANIAN., O. A holistic machine learning-based autoscaling approach for microservice applications. In: INSTICC. *11th International Conference on Cloud Computing and Services Science*. Online: SciTePress, 2021. p. 190–198. ISBN 978-989-758-510-4. ISSN 2184-5042.

GRACZYK, M.; LASOTA, T.; TELEC, Z.; TRAWIŃSKI, B. Nonparametric statistical analysis of machine learning algorithms for regression problems. In: SETCHI, R.; JORDANOV, I.; HOWLETT, R. J.; JAIN, L. C. (Ed.). *Knowledge-Based and Intelligent Information and Engineering Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 111–120. ISBN 978-3-642-15387-7.

GRAVES, A.; FERNáNDEZ, S.; GOMEZ, F.; SCHMIDHUBER, J. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In: *Proceedings of the 23rd International Conference on Machine Learning*. New York, NY, USA: Association for Computing Machinery, 2006. (ICML '06), p. 369–376. ISBN 1595933832.

HASSAN, S.; ALI, N.; BAHSOON, R. Microservice ambients: An architectural meta-modelling approach for microservice granularity. In: *IEEE International Conference on Software Architecture (ICSA)*. Gothenburg, Sweden: IEEE, 2017. p. 1–10. ISBN 978-1-5090-5729-0.

HAYKIN, S. *Redes neurais: princípios e prática*. [S.l.]: Bookman Editora, 2001.

HIGHTOWER, K.; BEDA, J.; BURNS, B. *Kubernetes: Up and Running: Dive Into the Future of Infrastructure*. [S.l.]: O'Reilly Media, 2017. ISBN 9781491935675.

HPA. *Horizontal Pod Autoscaling Documentation*. 2024. <https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/>. Online; accessed May 2024.

HUANG, L.; LEE, M.-Y.; CHEN, X.; TSENG, H.-W.; YANG, C.-F.; LEE, S.-F. Using microservice architecture as a load prediction strategy for management system of university public service. *Sensors and Materials*, v. 33, n. 2, p. 805–814, 2021.

HUEBSCHER, M. C.; MCCANN, J. A. A survey of autonomic computing—degrees, models, and applications. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 40, n. 3, ago. 2008. ISSN 0360-0300.

IBM. An architectural blueprint for autonomic computing. *IBM White Paper*, Citeseer, v. 31, n. 2006, p. 1–6, 2006.

IBM. *Microservices in the enterprise, 2021: Real benefits, worth the challenges*. 2021. Disponível em: <https://www.ibm.com/downloads/cas/OQG4AJAM>.

IQBAL, W.; ERRADI, A.; MAHMOOD, A. Dynamic workload patterns prediction for proactive auto-scaling of web applications. *Journal of Network and Computer Applications*, v. 124, p. 94 – 107, 2018. ISSN 1084-8045.

ISTIO. 2024. <https://istio.io/>. Online; accessed May 2024.

JMETER, A. 2024. <https://jmeter.apache.org/>. Online; accessed May 2024.

KAKADE, S.; ABBIGERI, G.; PRABHU, O.; DALWAYI, A.; G, N.; PATIL, S. P.; SUNAG, B. Proactive horizontal pod autoscaling in kubernetes using bi-lstm. In: *2023 IEEE International Conference on Contemporary Computing and Communications (InC4)*. [S.l.: s.n.], 2023. v. 1, p. 1–5.

KANG, P.; LAMA, P. Robust resource scaling of containerized microservices with probabilistic machine learning. In: *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. Leicester, United Kingdom, United Kingdom: IEEE, 2020. p. 122–131.

KEPHART, J.; CHESS, D. The vision of autonomic computing. *Computer*, v. 36, n. 1, p. 41–50, 2003.

KISILEVICH, S.; MANSMANN, F.; NANNI, M.; RINZIVILLO, S. Spatio-temporal clustering. In: ______. *Data Mining and Knowledge Discovery Handbook*. Boston, MA: Springer US, 2010. p. 855–874. ISBN 978-0-387-09823-4.

KLINAKU, F.; FRANK, M.; BECKER, S. Caus: An elasticity controller for a containerized microservice. In: *International Conference on Performance Engineering*. New York, NY, USA: ACM, 2018. p. 93–98.

KOURENTZES, N.; BARROW, D. K.; CRONE, S. F. Neural network ensemble operators for time series forecasting. *Expert Systems with Applications*, v. 41, n. 9, p. 4235–4244, 2014. ISSN 0957-4174.

KUBERNETES. *Production-Grade Container Orchestration*. 2024. <https://kubernetes.io/>. Online; accessed May 2024.

KUBERNETES. *Python Kubernetes API*. 2024. <https://github.com/kubernetes-client/python>. Online; accessed May 2024.

KUMAR, J.; SINGH, A. K. Workload prediction in cloud using artificial neural network and adaptive differential evolution. *Future Generation Computer Systems*, Elsevier, v. 81, p. 41–52, 2018.

KUMAR, Y.; KOUL, A.; KAUR, S.; HU, Y.-C. Machine learning and deep learning based time series prediction and forecasting of ten nations' covid-19 pandemic. *SN Computer Science*, v. 4, n. 1, p. 91, Dec 2022.

LANCIANO, G.; GALLI, F.; CUCINOTTA, T.; BACCIU, D.; PASSARELLA, A. Predictive auto-scaling with openstack monasca. In: *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing*. NY, USA: ACM, 2021. (UCC '21). ISBN 9781450385640.

LARRUCEA, X.; SANTAMARIA, I.; COLOMO-PALACIOS, R.; EBERT, C. Microservices. *IEEE Software*, v. 35, n. 3, p. 96–100, 2018.

LAURENO, M. *Máquinas Virtuais e Emuladores: Conceitos, Técnicas e Aplicaçoes*. Novatec, 2006. ISBN 9788575220986. Disponível em: <https://books.google.com.br/books?id=JQLeQBH-SUYC>.

LEWIS, J.; FOWLER, M. *Microservices: A Definition of This New Architectural Term*. 2014.

LI, H.; WANG, S.; RUAN, C. A fast approach of provisioning virtual machines by using image content similarity in cloud. *IEEE Access*, v. 7, p. 45099–45109, 2019.

LI, X.; CHEN, Y.; LIN, Z. Towards automated inter-service authorization for microservice applications. In: *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*. New York, NY, USA: Association for Computing Machinery, 2019. (SIGCOMM Posters and Demos '19), p. 3–5. ISBN 9781450368865.

LI, Y.; XIA, Y. Auto-scaling web applications in hybrid cloud based on docker. In: *2016 5th International Conference on Computer Science and Network Technology (ICCSNT)*. [S.l.: s.n.], 2016. p. 75–79.

LIM, B.; ARıK, S. Ö.; LOEFF, N.; PFISTER, T. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, v. 37, n. 4, p. 1748–1764, 2021. ISSN 0169-2070.

LORIDO-BOTRAN, T.; MIGUEL-ALONSO, J.; LOZANO, J. A. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing*, v. 12, n. 4, p. 559–592, 2014. ISSN 1572-9184.

LUO, S.; XU, H.; LU, C.; YE, K.; XU, G.; ZHANG, L.; DING, Y.; HE, J.; XU, C. Characterizing microservice dependency and performance: Alibaba trace analysis. In: *Proceedings of the ACM Symposium on Cloud Computing*. New York, NY, USA: Association for Computing Machinery, 2021. (SoCC '21), p. 412–426. ISBN 9781450386388.

MA, M.; JIN, B.; LUO, S.; GUO, S.; HUANG, H. A novel dynamic integration approach for multiple load forecasts based on q-learning algorithm. *International Transactions on Electrical Energy Systems*, v. 30, n. 7, p. e12146, 2020. E12146 ITEES-19-0540.R1.

MA, X.; TAO, Z.; WANG, Y.; YU, H.; WANG, Y. Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. *Transportation Research Part C: Emerging Technologies*, v. 54, p. 187–197, 2015. ISSN 0968-090X.

MALHOTRA, L.; AGARWAL, D.; JAISWAL, A. et al. Virtualization in cloud computing. *J. Inform. Tech. Softw. Eng*, v. 4, n. 2, p. 1–3, 2014.

MARIE-MAGDELAINE, N.; AHMED, T. Proactive autoscaling for cloud-native applications using machine learning. In: *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*. Taipei, Taiwan: IEEE, 2020. p. 1–7.

MARTíNEZ, R. G.; LI, Z.; LOPES, A.; RODRIGUES, L. Augure: Proactive reconfiguration of cloud applications using heterogeneous resources. In: *16th International Symposium on Network Computing and Applications*. MA, USA: IEEE, 2017. p. 1–8. ISSN 978-1-5386-1465-5.

MASTELINI, S. M.; de Carvalho, A. C. P. de L. F. Using dynamical quantization to perform split attempts in online tree regressors. *Pattern Recognition Letters*, v. 145, p. 37–42, 2021. ISSN 0167-8655.

MEHTAB, S.; SEN, J. A time series analysis-based stock price prediction using machine learning and deep learning models. *ArXiv*, abs/2004.11697, 2020.

MENDE, T.; KOSCHKE, R. Revisiting the evaluation of defect prediction models. In: *Proceedings of the 5th International Conference on Predictor Models in Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2009. (PROMISE '09), p. 1–10. ISBN 9781605586342.

MENDES-MOREIRA, J.; JORGE, A. M.; SOARES, C.; SOUSA, J. F. de. Ensemble learning: A study on different variants of the dynamic selection approach. In: PERNER, P. (Ed.). *Machine Learning and Data Mining in Pattern Recognition*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 191–205. ISBN 978-3-642-03070-3.

MENDES-MOREIRA, J. a.; SOARES, C.; JORGE, A. M.; SOUSA, J. F. D. Ensemble approaches for regression: A survey. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 45, n. 1, dez. 2012. ISSN 0360-0300.

MOHAMED, H.; EL-GAYAR, O. End-to-end latency prediction of microservices workflow on kubernetes: A comparative evaluation of machine learning models and resource metrics. In: *Proceedings of the 54th Hawaii International Conference on System Sciences*. [S.l.: s.n.], 2021. p. 1717–1726.

MORENO, G. A.; CáMARA, J.; GARLAN, D.; SCHMERL, B. Proactive self-adaptation under uncertainty: A probabilistic model checking approach. In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2015. (ESEC/FSE 2015), p. 1–12. ISBN 9781450336758.

MORENO-VOZMEDIANO, R.; MONTERO, R. S.; HUEDO, E.; LLORENTE, I. M. Efficient resource provisioning for elastic cloud services based on machine learning techniques. *Journal of Cloud Computing*, v. 8, n. 1, p. 5, Apr 2019. ISSN 2192-113X.

MORI, U.; MENDIBURU, A.; LOZANO, J. A. Similarity measure selection for clustering time series databases. *IEEE Transactions on Knowledge and Data Engineering*, v. 28, n. 1, p. 181–195, 2016.

MOURA, T. J.; CAVALCANTI, G. D.; OLIVEIRA, L. S. Mine: A framework for dynamic regressor selection. *Information Sciences*, v. 543, p. 157–179, 2021. ISSN 0020-0255.

NEMENYI, P. *Distribution-free Multiple Comparisons*. US: Princeton University, 1963.

NETO, P. S. G. de M.; CAVALCANTI, G. D. C.; JÚNIOR, D. S. de O. S.; SILVA, E. G. Hybrid systems using residual modeling for sea surface temperature forecasting. *Scientific Reports*, v. 12, n. 1, p. 487, Jan 2022. ISSN 2045-2322.

NEWMAN, S. *Building Microservices*. Sebastopol, CA: O'Reilly Media, Inc., 2015. ISBN 1491950358, 9781491950357.

NIKRAVESH, A. Y.; AJILA, S. A.; LUNG, C.-H. An autonomic prediction suite for cloud resource provisioning. *Journal of Cloud Computing*, v. 6, n. 1, p. 3, Feb 2017. ISSN 2192-113X.

OLIVEIRA, J. F. L. de; SILVA, E. G.; NETO, P. S. G. de M. A hybrid system based on dynamic selection for time series forecasting. *IEEE Transactions on Neural Networks and Learning Systems*, v. 33, n. 8, p. 3251–3263, 2022.

ONLINE-BOUTIQUE. *Microservice Demo Repository*. 2024. <https://github.com/GoogleCloudPlatform/microservices-demo>. Online; accessed May 2024.

PAHL, C. Containerization and the paas cloud. *IEEE Cloud Computing*, v. 2, n. 3, p. 24–31, May 2015. ISSN 2372-2568.

PALMERINO, J.; YU, Q.; DESELL, T.; KRUTZ, D. Improving the decision-making process of self-adaptive systems by accounting for tactic volatility. In: *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. San Diego, CA, USA, USA: IEEE, 2019. p. 949–961.

PATERAKIS, N. G.; MOCANU, E.; GIBESCU, M.; STAPPERS, B.; ALST, W. van. Deep learning versus traditional machine learning methods for aggregated energy demand prediction. In: *2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*. [S.l.: s.n.], 2017. p. 1–6.

PODOLSKIY, V.; JINDAL, A.; GERNDT, M.; OLEYNIK, Y. Forecasting models for self-adaptive cloud applications: A comparative study. In: *2018 IEEE 12th International Conference on Self-Adaptive and Self-Organizing Systems*. Trento, Italy: IEEE, 2018. p. 40–49. ISSN 1949-3673.

PRACHITMUTITA, I.; AITTINONMONGKOL, W.; POJJANASUKSAKUL, N.; SUPATTATHAM, M.; PADUNGWEANG, P. Auto-scaling microservices on iaas under sla with cost-effective framework. In: *2018 Tenth International Conference on Advanced Computational Intelligence (ICACI)*. [S.l.: s.n.], 2018. p. 583–588.

PRASETYO, H. D.; HOGANTARA, P. A.; ISNAINIYAH, I. N. A web-based diabetes prediction application using xgboost algorithm. *Data Science: Journal of Computing and Applied Informatics*, v. 5, n. 2, p. 49–59, Jul. 2021. Disponível em: <https://talenta.usu.ac.id/JoCAI/article/view/6290>.

PRYBUTOK, V. R.; YI, J.; MITCHELL, D. Comparison of neural network models with arima and regression models for prediction of houston's daily maximum ozone concentrations. *European Journal of Operational Research*, v. 122, n. 1, p. 31–40, 2000. ISSN 0377-2217.

PYTHON. 2024. <https://www.python.org/>. Online; accessed May 2024.

QASSEM, L. M. A.; STOURAITIS, T.; DAMIANI, E.; ELFADEL, I. A. M. Proactive random-forest autoscaler for microservice resource allocation. *IEEE Access*, v. 11, p. 2570–2585, 2023.

QIN, Y.; SONG, D.; CHENG, H.; CHENG, W.; JIANG, G.; COTTRELL, G. W. A dual-stage attention-based recurrent neural network for time series prediction. In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. [S.l.]: AAAI Press, 2017. (IJCAI'17), p. 2627–2633. ISBN 9780999241103.

QIU, X.; REN, Y.; SUGANTHAN, P. N.; AMARATUNGA, G. A. Empirical mode decomposition based ensemble deep learning for load demand time series forecasting. *Applied Soft Computing*, v. 54, p. 246–255, 2017. ISSN 1568-4946.

QUARKUS-HTTP-DEMO. *Quarkus Runtime Performance*. 2024. <https://quarkus.io/blog/runtime-performance/>. Online; accessed May 2024.

RAHIMZAD, M.; NIA, A. M.; ZOLFONOON, H.; SOLTANI, J.; MEHR, A. D.; KWON, H.-H. Performance comparison of an lstm-based deep learning model versus conventional machine learning algorithms for streamflow forecasting. *Water Resources Management*, v. 35, n. 12, p. 4167–4187, Sep 2021. ISSN 1573-1650.

RANGAPURAM, S. S.; SEEGER, M. W.; GASTHAUS, J.; STELLA, L.; WANG, Y.; JANUSCHOWSKI, T. Deep state space models for time series forecasting. In: BENGIO, S.; WALLACH, H.; LAROCHELLE, H.; GRAUMAN, K.; CESA-BIANCHI, N.; GARNETT, R. (Ed.). *Advances in Neural Information Processing Systems*. Montreal, CA: Curran Associates, Inc., 2018. v. 31, p. 1–10.

Ribeiro, G. H. T.; Neto, P. S. G. d. M.; Cavalcanti, G. D. C.; Tsang, I. R. Lag selection for time series forecasting using particle swarm optimization. In: *The 2011 International Joint Conference on Neural Networks*. [S.l.: s.n.], 2011. p. 2437–2444. ISSN 2161-4393.

ROONEY, N.; PATTERSON, D.; ANAND, S.; TSYMBAL, A. Dynamic integration of regression models. In: ROLI, F.; KITTLER, J.; WINDEATT, T. (Ed.). *Multiple Classifier Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. p. 164–173. ISBN 978-3-540-25966-4.

ROSSI, F.; CARDELLINI, V.; PRESTI, F. L. Hierarchical scaling of microservices in kubernetes. In: *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. Washington, DC, USA, USA: IEEE, 2020. p. 28–37.

RUBAK, A. *Dimensioning Microservices on Kubernetes Platforms Using Machine Learning Techniques*. 73 p. Dissertação (Mestrado) — Karlstad University, Department of Mathematics and Computer Science (from 2013), 2023.

SALEHIE, M.; TAHVILDARI, L. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, ACM, New York, NY, USA, v. 4, n. 2, p. 14:1–14:42, 2009. ISSN 1556-4665.

SALINAS, D.; FLUNKERT, V.; GASTHAUS, J.; JANUSCHOWSKI, T. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, v. 36, n. 3, p. 1181–1191, 2020. ISSN 0169-2070.

SAMPAIO, A. R.; BESCHASTNIKH, I.; MAIER, D.; BOURNE, D.; SUNDARESEN, V. Auto-tuning elastic applications in production. In: *45th International Conference on Software Engineering*. Melbourne, Australia: IEEE, 2023. p. 355–367.

SAMPAIO, A. R.; RUBIN, J.; BESCHASTNIKH, I.; ROSA, N. S. Improving microservice-based applications with runtime placement adaptation. *Journal of Internet Services and Applications*, v. 10, n. 1, p. 4, Feb 2019. ISSN 1869-0238.

SCARGLE, J. D. Studies in astronomical time series analysis. iii. fourier transforms, autocorrelation functions, and cross-correlation functions of unevenly spaced data. *The Astrophysical Journal*, v. 343, p. 874, Aug 1989.

SCHULER, L.; JAMIL, S.; KüHL, N. Ai-based resource allocation: Reinforcement learning for adaptive auto-scaling in serverless environments. In: *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. [S.l.: s.n.], 2021. p. 804–811.

SHALABI, L. A.; SHAABAN, Z.; KASASBEH, B. Data mining: A preprocessing engine. *Journal of Computer Science*, v. 2, n. 9, p. 735–739, 2006.

SHIM, S.; DHOKARIYA, A.; DOSHI, D.; UPADHYE, S.; PATWARI, V.; PARK, J.-Y. Predictive auto-scaler for kubernetes cloud. In: *2023 IEEE International Systems Conference (SysCon)*. [S.l.: s.n.], 2023. p. 1–8.

SILVA, E. G.; CAVALCANTI, G. D. C.; OLIVEIRA, J. F. L. de; NETO, P. S. G. de M. On the evaluation of dynamic selection parameters for time series forecasting. In: *2020 International Joint Conference on Neural Networks (IJCNN)*. Glasgow, UK: IEEE, 2020. p. 1–7.

SILVA, E. G.; JúUNIOR, D. S. de O.; CAVALCANTI, G. D. C.; NETO, P. S. G. de M. Improving the accuracy of intelligent forecasting models using the perturbation theory. In: *2018 International Joint Conference on Neural Networks*. RJ, Brazil: IEEE, 2018. p. 1–7.

SILVA, E. G.; NETO, P. S. G. D. M.; CAVALCANTI, G. D. C. A dynamic predictor selection method based on recent temporal windows for time series forecasting. *IEEE Access*, v. 9, p. 108466–108479, 2021.

SINGH, P.; GUPTA, P.; JYOTI, K.; NAYYAR, A. Research on auto-scaling of web applications in cloud: survey, trends and future directions. *Scalable Computing: Practice and Experience*, v. 20, n. 2, p. 399–432, 2019.

SOMPOLINSKY, H.; KANTER, I. Temporal association in asymmetric neural networks. *Phys. Rev. Lett.*, American Physical Society, v. 57, p. 2861–2864, Dec 1986.

SOPPELSA, F.; KAEWKASI, C. *Native Docker Clustering with Swarm*. [S.l.]: Packt Publishing, 2016. ISBN 1786469758.

SRIRAMAN, A.; WENISCH, T. F. µTune: Auto-Tuned threading for OLDI microservices. In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, 2018. p. 177–194. ISBN 978-1-939133-08-3.

STEEN, M. V.; TANENBAUM, A. S. *Distributed systems*. The Netherlands: Maarten van Steen Leiden, 2017.

STRAESSER, M.; GROHMANN, J.; KISTOWSKI, J. von; EISMANN, S.; BAUER, A.; KOUNEV, S. Why is it not solved yet? challenges for production-ready autoscaling. In: *International Conference on Performance Engineering*. NY, USA: ACM, 2022. p. 105–115. ISBN 9781450391436.

TAHERIZADEH, S.; STANKOVSKI, V. Auto-scaling applications in edge computing: Taxonomy and challenges. In: *Proceedings of the International Conference on Big Data and Internet of Thing*. New York, NY, USA: Association for Computing Machinery, 2017. (BDIOT2017), p. 158–163. ISBN 9781450354301.

TAHIR, F.; ABDULLAH, M.; BUKHARI, F.; ALMUSTAFA, K. M.; IQBAL, W. Online workload burst detection for efficient predictive autoscaling of applications. *IEEE Access*, v. 8, p. 73730–73745, 2020.

TAIBI, D.; LENARDUZZI, V.; PAHL, C.; JANES, A. Microservices in agile software development: A workshop-based study into issues, advantages, and disadvantages. In: *Proceedings of the XP2017 Scientific Workshops*. NY, USA: ACM, 2017. p. 23:1–23:5. ISBN 978-1-4503-5264-2.

TOFFETTI, G.; BRUNNER, S.; BLöCHLINGER, M.; DUDOUET, F.; EDMONDS, A. An architecture for self-managing microservices. In: *Proceedings of the 1st International Workshop on Automated Incident Management in Cloud*. NY, USA: ACM, 2015. p. 19–24.

TOKA, L.; DOBREFF, G.; FODOR, B.; SONKOLY, B. Machine learning-based scaling management for kubernetes edge clusters. *IEEE Transactions on Network and Service Management*, v. 18, n. 1, p. 958–972, 2021.

TRAVELS. *Travels Demo*. 2024. <https://github.com/kiali/demos/tree/master/travels>. Online; accessed May 2024.

VAILSHERY, L. S. *Do you utilize microservices within your organization?* 2021. <https://www.statista.com/statistics/1236823/microservices-usage-per-organization-size/#statisticContainer>. Online; accessed March 2024.

VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, L.; POLOSUKHIN, I. Attention is all you need. In: *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2017. (NIPS'17), p. 6000–6010. ISBN 9781510860964.

WANG, L.; WANG, Z.; QU, H.; LIU, S. Optimal forecast combination based on neural networks for time series forecasting. *Applied Soft Computing*, v. 66, p. 1–17, 2018. ISSN 1568-4946.

WANG, T. *Predictive vertical CPU autoscaling in Kubernetes based on time-series forecasting with Holt-Winters exponential smoothing and long short-term memory*. 55 p. Dissertação (Mestrado) — School of Electrical Engineering and Computer Science, 2021.

WEYNS, D.; SCHMERL, B.; GRASSI, V.; MALEK, S.; MIRANDOLA, R.; PREHOFER, C.; WUTTKE, J.; ANDERSSON, J.; GIESE, H.; GOSCHKA, K. M. On patterns for decentralized control in self-adaptive systems. In: ______. *Software Engineering for Self-Adaptive Systems II: International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. cap. On Patterns for Decentralized Control in Self-Adaptive Systems, p. 76–107. ISBN 978-3-642-35813-5.

WIDODO, A.; BUDI, I. Model selection for time series forecasting using similarity measure. In: *2011 International Conference on Advanced Computer Science and Information Systems*. [S.l.: s.n.], 2011. p. 221–226.

WILLMOTT, C. J.; ACKLESON, S. G.; DAVIS, R. E.; FEDDEMA, J. J.; KLINK, K. M.; LEGATES, D. R.; O'DONNELL, J.; ROWE, C. M. Statistics for the evaluation and comparison of models. *Journal of Geophysical Research: Oceans*, v. 90, n. C5, p. 8995–9005, 1985.

WONG, J. P. *HyScale: Hybrid Scaling of Dockerized Microservices Architectures*. 81 p. Dissertação (Mestrado) — University of Toronto, 2018.

XU, M.; SONG, C.; ILAGER, S.; GILL, S. S.; ZHAO, J.; YE, K.; XU, C. Coscal: Multifaceted scaling of microservices with reinforcement learning. *IEEE Transactions on Network and Service Management*, v. 19, n. 4, p. 3995–4009, 2022.

YADAV, M. P.; Rohit; YADAV, D. K. Maintaining container sustainability through machine learning. *Cluster Computing*, v. 24, n. 4, p. 3725–3750, Dec 2021. ISSN 1573-7543.

YAO, C.; DAI, Q.; SONG, G. Several novel dynamic ensemble selection algorithms for time series prediction. *Neural Processing Letters*, v. 50, n. 2, p. 1789–1829, Oct 2019. ISSN 1573-773X.

YOSHIDA, S.; CHAKRABORTY, B. A comparative study of similarity measures for time series classification. In: OTAKE, M.; KURAHASHI, S.; OTA, Y.; SATOH, K.; BEKKI, D. (Ed.). *New Frontiers in Artificial Intelligence*. Cham: Springer International Publishing, 2017. p. 397–408. ISBN 978-3-319-50953-2.

YU, G.; CHEN, P.; ZHENG, Z. Microscaler: Automatic scaling for microservices with an online learning approach. In: *2019 IEEE International Conference on Web Services (ICWS)*. Milan, Italy, Italy: IEEE, 2019. p. 68–75. ISSN null.

YU, Y.; ZHU, Y.; LI, S.; WAN, D. Time series outlier detection based on sliding window prediction. *Mathematical Problems in Engineering*, Hindawi Publishing Corporation, v. 2014, p. 879736, Oct 2014. ISSN 1024-123X.

ZHANG, P.; PATUWO, E.; HU, M. Forecasting with artificial neural networks: The state of the art. *International Journal of Forecasting*, v. 14, p. 35–62, 03 1998.

ZHANG, S.; MAO, X.; LIU, P.; HOU, F. A self-adaptation framework of microservice systems. In: *SEKE*. CA, USA: Software Engineering & Knowledge Engineering, 2018. p. 282–325.

APPENDIX A – DESCRIPTION OF ALIBABA SERIES

Tables A.1 and A.2 aim to provide a comprehensive overview of the Alibaba series (LUO et al., 2021). Table A.1 highlights the distinct features and trends of these series, making it easier to analyse and compare. Table A.2 shows the microservices and containers from which data in the Alibaba dataset was extracted.

Table A.1 – A detailed description of Alibaba time series.

Metric	Series	Seasonal	Stationary	Trend	Frequency	Mean	Median	Std	Size	Communication Mechanisms
C	1	✓			S	0.342	0.330	0.049	1426	RPC HTTP MQ
C	2	✓			S	0.176	0.175	0.026	1428	RPC
C	3	✓		✓	S	0.169	0.163	0.025	1425	RPC
C	4	✓		✓	S	0.344	0.397	0.113	1427	RPC
C	5	✓		✓	S	0.125	0.077	0.082	1423	RPC MQ
C	6	✓			S	0.180	0.151	0.066	1420	RPC HTTP MQ
C	7	✓	✓		S	0.083	0.078	0.019	1424	RPC MQ
C	8	✓		✓	S	0.378	0.380	0.072	1431	RPC MQ
C	9	✓		✓	S	0.097	0.096	0.010	1428	RPC MQ
C	10	✓			S	0.316	0.311	0.040	1421	RPC MQ
M	1	✓	✓		S	0.528	0.524	0.044	1427	RPC MQ
M	2	✓		✓	S	0.611	0.725	0.135	1424	RPC MQ
M	3	✓		✓	S	0.515	0.512	0.014	1427	RPC HTTP
M	4	✓		✓	S	0.511	0.502	0.026	1426	RPC
M	5	✓		✓	S	0.861	0.862	0.010	1431	RPC MQ
M	6	✓	✓	✓	S	0.523	0.523	0.004	1426	RPC HTTP
M	7	✓		✓	S	0.464	0.463	0.007	1422	RPC HTTP MQ
M	8	✓	✓		S	0.279	0.283	0.019	1426	HTTP
M	9	✓		✓	S	0.808	0.806	0.015	1417	RPC MQ
M	10	✓		✓	S	0.445	0.446	0.016	1424	RPC MQ
RT	1	✓			M	1.002	1.016	0.196	720	RPC
RT	2	✓			M	337.680	333.625	38.488	720	RPC
RT	3	✓	✓		M	6.990	7.112	1.068	720	RPC
RT	4	✓			M	23.754	23.950	3.064	720	RPC
RT	5	✓			M	259.589	254.659	41.350	721	HTTP
RT	6		✓		M	12.662	12.250	7.492	227	HTTP
RT	7	✓		✓	M	50.415	52.052	10.224	721	HTTP
RT	8	✓	✓		M	59.940	58.240	14.790	721	MQ
RT	9	✓			M	15.000	15.041	1.090	721	MQ
RT	10	✓			M	470.378	371.962	255.561	715	MQ

T	1		✓		M	222.392	220.583	12.420	721	RPC
T	2	✓			M	139.128	143.117	10.188	721	RPC
T	3	✓		✓	M	74.175	71.992	9.810	720	RPC
T	4	✓			M	50.750	54.367	11.690	721	RPC
T	5	✓	✓		M	219.579	222.430	69.082	719	MQ
T	6	✓			M	111.599	44.290	87.337	713	MQ
T	7	✓			M	4.669	4.677	1.285	721	MQ
T	8		✓		M	151.233	151.442	8.202	721	HTTP
T	9	✓	✓		M	7.664	7.663	0.779	721	HTTP
T	10	✓			M	258.095	255.120	40.429	721	HTTP

Metrics: CPU usage (C), Memory (M), Response Time (RT) and Traffic (T)

Table A.2 – Listing of the microservices and containers from which the Alibaba dataset data was extracted.

Metric	Series	Microservices	Container
C	1	a1e86e4d8e89da93f40ea4d0ff844517	39b74d87e4fa46ef39d9ca578cb71fd4
		f4e6643363f4e0e2575d265febb42c1f	1ff006af9f94e1c54967da6e34ce05be
C	2	b1601b7399bdfe0c5624ae79665315fa	1fa8ea4c2b2132bf01290ffd9bb2d09d
		50a6ff01fb9e08779a3d5689039e229c	c918fa152c663ce5dd295e8f70c89c49
C	3	7d23087eb792acb037003e30b86b836f	c3bcb0cb09fd91008ac84664c238fcd4
		f9e44d4df6179e78b684ce8892aa5eec	235167e811463c05205c6aed54e706c0
C	4	cd15e277536a82697e10dd402532ebec	b9f37b7f0e17af067ddd5293a880c50b
		8082bd865dcb18c45bff323dc154edac	a16768ad4c0e5a35d0c8be41cb272d56
C	5	b8be9ee0bb33cc822722a125fad0c825	a6c14c91631ffc85e876460ed337fe22
		f69a9c373afed2a063996e351a2324e0	f388b9df20e11fd5c50217c70e212c8e
C	6	3ebf2b6b3c81cff20d8280c847226256	9d4f3fb5674cf0c0a53600608f66b533
		5a016feea468a35b5b9402cdb7b5e657	70dd15c97480c21c9729cc18f9db1a49
C	7	4599f7a798d7579464bf803f8e465f1c	74c18a3846730757eec5920726ef0c13
		6e72821a327653ba764a4083203f1b6f	b36ce4376552970e47d1f210fc732230
C	8	cc0b7e3616b733c02dd218fe52c40b91	79c6b8f79044912becec70b8a2963319
		1d98945cf32e1e114cf46692b033cafb	56a6630f51d616e59323638024e6ac68
C	9	85fb296baeb19fc32c446b044b788186	fa00d4626fa2dc8521d74cb1ba36a5ed
		f25c6158f8f4cb1a29547143d69f2089	102f8479b30eee542ecb8c5e96185596
C	10	14b7e680bd155b30d61e395c40ffb9b6	3a768af0d7ae7660faefb2ee2f96daac
		0b013d9a05414635186eeb622590e789	58f1808a8a0871b9d4aa87ea3e640f80
M	1	655e0269bb90b5ffe6a9d7a22bfc21df	2b6f93562a079e6d3b2b1571162469ea
		4cfc0aae9dc186e11d8f5388789653f7	ff3b48a6880307d224b4cab280e31070
M	2	103f50cb873950444c74ab1b92c007cb	ebf49c2ee410b8786f7a1144fdcafeb6
		fe8eced210268cde0eb8c3a059e7f3bb	3fb5395da92f3b3f43af65892921d7a5
M	3	f1e9137370b218b2428565c0ad8e1aa1	bf3f28a85cf4e023fb65e5bd0a9d238c
		2a4c48deb54a4dc2429191f32f30f998	c2ae56cadf934d16af3253b75a1ff977

M	4	2d37b4120ad5a4f3e8f5e746a5fe83f2 569ec2bde0dcb3cf4147c25bdaa7f13c	4dfffe64135d1375aed0a933e4744614 3dc83b9b7b50001a618c415bfd666961
M	5	e4ed7cd27c4cc804a416aa347b47eea6 eae054714d8c0ec2645cc4f610c487ca	541cafe2de1a9bc4f63019c635c7c266 dee072dd6b2b7512ba5b9d4f7e94a7eb
M	6	54cbf33f387c96eba6a0b3d83ba1796a 85f70c47f53891f02682b0ef8edccb62	348c43aae7f0b6921adc746760804ca9 31de217babb3346e7ff8327f67395e05
M	7	b708a6b01a261167011b934178bcd031 d92197602da0c7ec2b32e36795f90991	a0df7d75d9bcd7a80ef06e0c9443aff8 3fbff710489a84d58a3cca27c83f920e
M	8	7695b43b41732a0f15d3799c8eed2852 665fe8da29fd700c383550fc16e521a3	4fdc0bf40114dc4c4a77a85bcd63e37e 6cc7ec554394f65fd6120a3df97e4927
M	9	ec4b52c7e74ef232643dc244c3bac86e d5a55de1a3cec705914e3b7cea632f0f	a74862e810a2c3aada8bb44f8f8c633e dd0ca4706faddc5a78280687386a796b
M	10	dbcbac8c618d9d04d39fdd41759a1ba2 51e5ad1eae1ebdd98255c36942ba3d93	09702f93678ffe7e5ba455621047cc67 779e3940f1b67e14426e4c06f869397c
RT	1	e5b95cb4a294ca8b852a217ac602f69c 266e6b0799d399a2fe9dbc386cc845b4	78811eca8e788dd03f6637684eda4d86 998db1a7641e113b9320f6173b742cb7
RT	2	d1cfb72cb3209836a6c2efce9d1b4b7c e29fcf7dd521d6265cd29ab41f66301b	81ea9777c3b89089e53bf2f6c4816a0f 5deefff0c6898478021f8a2bc0bd1dfa
RT	3	273ab562e5b930887de6eeb565910412 067edcd76558268defe4beea89201d42	61c838ff9bce9171fd12691ff483115a c255f44120c79e4ab9cc3f02003ee022
RT	4	02bb03429fd7f79d5d8acb3b7f3f82e0 f0ce949cc8f812638c90470c7ff957f4	3540ae72b44e2a2095ada05641008e52 206826d0145532939330bfa02389753a
RT	5	d1cfb72cb3209836a6c2efce9d1b4b7c e29fcf7dd521d6265cd29ab41f66301b	97063bb98b67eb10e64839414cb2fec3 08a5a0209a22eed97de61d8d2fd8e2c1
RT	6	1cd9d5ae0fcdc971f9a2dbdd426f51c7 a85a1e9f06f1f4ddea48df1562b88e52	81c5418ca73828c45221e7e576904084 fe74c5300b76a3ba679377b295f28abb
RT	7	73dc0b88853241955161056b00351811 2b11b6648ed1fedeb0f39567a0e29346	f013c9ed2a459cc066b227512e796a85 69fa69e8a6362de27ce2689b16e396a2
RT	8	788af055ddb4322a7082f5b3a03511e7 d7c0dab5d7b175d13e83008369246e3e	fcbccd4c317e6004acdf116a58f9bbd0 5a574a2871258d439ba3f1a27c5c90ef
RT	9	9c5a60d5b8b67b6e4860979bc485e60c 0314d218a1f8bc0541a1c09942b6f8c2	d4a93bf231fd8ba2f988bcc0d642baa6 6df2745134b146bca42483d5b6acf7c7
RT	10	a02426f289a5396ce87d0ab1ff795b40 d57b266e1571513ea1fe9ef6eab8c772	f3f2377e29277d5a212f6d2c1a620aed 25e149d7c450cb3e3ac82482e7be88f7
T	1	2bcc2832736be67623ce452166a42e2e deefb2f5c8afb3677b52fd095dea7957	08bf9215a57c0c2af15c616ea374cd84 ddb2aa8de2e4360dcaf59186af21c346
T	2	7b3b9cb2faed94c69ae18e8275ec186a 4ffe6b2e2588782db56b1481ecf13ffc	828ce50573245ffaa5bb611a0fae1db0 d540ebf5d6e7a073ae9d4037acadf1e7
T	3	8e0c180ac6ac8dde938ba41545771120 0c59d5d7389d3ae579dbcef1172e9931	642489fa34e850b405900f51b0399c22 c447a8a62fafea5dbe4a16a88e147e24

T	4	a705a0a47ebd57460258a7fe818f67cf 4e0898fdd9bdde946697594c415d4079	2de26f4893dfec3d98ccf50b51af3b7c 4f3bcdd16c19c7a78ff3ab258ff5eb2d
T	5	40f23d72d4d74a9663705cac9d435e6b 40f64b5c94e7b0b05f2978afd15365e5	d55d50b42e40dfbee8e9905e8d62ac91 46955b1be18590224af8281998d493c3
T	6	9c5a60d5b8b67b6e4860979bc485e60c 0314d218a1f8bc0541a1c09942b6f8c2	d4a93bf231fd8ba2f988bcc0d642baa6 6df2745134b146bca42483d5b6acf7c7
T	7	a8e72f7a9b6188b25a130b386821ac57 40c763386e9b8b72addcaf38218ded85	f13a7e649eefd0032fd8f7ab9ad4ddfd 3dd08b3d27ec49578dc72b0df480f02c
T	8	b70842c1b31dc88da528ba646e3a6876 cdce7a5425337dbeefeab70d033556ee	b3f46a1c327a6ee8d201ec050088b6cd 0386b577121ba71c84eb59c63796a55a
T	9	e982765b911e7a7e625252f69980362c 3b1792f3487a1a39bd314351e1eb6fc3	3e2e936bbcecf6f91cc2e5e1e57d28ad 50dc7a9f2f7dda34605a32d63526ff4b
T	10	d1cfb72cb3209836a6c2efce9d1b4b7c e29fcf7dd521d6265cd29ab41f66301b	97063bb98b67eb10e64839414cb2fec3 08a5a0209a22eed97de61d8d2fd8e2c1

Metrics: CPU usage (C), Memory (M), Response Time (RT) and Traffic (T)

APPENDIX B – GENERATING SYNTHETIC TIME SERIES

Synthetic time series can be generated by requesting a MBA with a specific workload over time (ABDULLAH; IQBAL; ERRADI, 2019; ABDULLAH et al., 2021; Fontana de Nardin et al., 2021). This is the same approach used for generating the workloads in Chapter 5.

Generated Workloads. A workload is a set of incoming requests to a target application over time. Web applications such as microservices are commonly submitted to four common workload patterns: increasing, decreasing, periodic, and random (ABDULLAH; IQBAL; ERRADI, 2019; ABDULLAH et al., 2021). An increasing workload is observed when incoming requests to the application increase at a specific rate over time, while a decreasing workload is the inverse of an increasing one. A periodic workload is observed when the incoming requests to the application have a particular pattern that repeats periodically. Finally, a random workload is observed when there is no pattern in the incoming requests.

Tools like httpperf[1], Jmeter[2] and Locust[3] are commonly employed to generate a workload pattern for a target application. The workload pattern is created by adjusting the number of active users in the tool during the experiment. The equations proposed by (IQBAL; ERRADI; MAHMOOD, 2018) were used to calculate these adjustments.

The number of active users (χ_{nu_i}) in the i-th interval in an increasing workload is given by:

$$\chi_{nu_i} = \chi_{nu_0} + (i-1)B + s_c Rand(\mathbb{N}(0,1)), \tag{B.1}$$

where χ_{nu_0} is the number of active users requesting the target application at the time interval ν_0; B is a constant that defines the number of active users added to each new interval ν; S_c is a scale factor used to inject normal random noise with mean zero and unit variance in the workload. In all experiments, 4320-time intervals were considered, each consisting of one minute; B and S_c were defined as 0.15. For the increasing workload, χ_{nu_0} was set as 1.

The number of active users (χ_{nu_i}) in the i-th interval in a decreasing workload is defined as:

$$\chi_{nu_i} = \chi_{nu_0} - (i-1)B + s_c Rand(\mathbb{N}(0,1)), \tag{B.2}$$

1 <https://github.com/httperf/httperf>
2 <https://jmeter.apache.org>
3 <https://locust.io>

where χ_{nu_0} was set as 600.

The number of active users (χ_{nu_i}) in the i-th interval in a periodic workload can be computed as follows:

$$\chi_{nu_i} = \chi_{nu_0} - \alpha sin(\frac{2i\pi}{\gamma}) + s_c Rand(\mathbb{N}(0,1)), \tag{B.3}$$

where γ is the duration of the periodic pattern; α is its amplitude. χ_{nu_0}, γ and α were set as 300;

The number of active users (χ_{nu_i}) in the i-th interval in a random workload is given by:

$$r_{sign} = Rand(0,1),$$

$$\chi_{nu_i} = \begin{cases} \chi_{nu}(i-1) \times (1 + r_c) & if \;\; r_{sign} > 0.50, \\ \chi_{nu}(i-1) \times (1 - r_c) & otherwise. \end{cases} \tag{B.4}$$

where r_{sign}, r_c are random numbers, and $\chi_{nu}(i-1)$ is number of active users in previous interval. The draw of r_{sign} ranges between 0 and 1; r_c ranges between 0 and c. χ_{nu_0} was set as 300; c was define as 2%.

Generated Time Series. Locust[4] (workload generation tool) and Online Boutique (OB)[5] (target application) were adopted for workload generation, as done by Yu, Chen & Zheng (2019), Li, Chen & Lin (2019) and Marie-Magdelaine & Ahmed (2020).

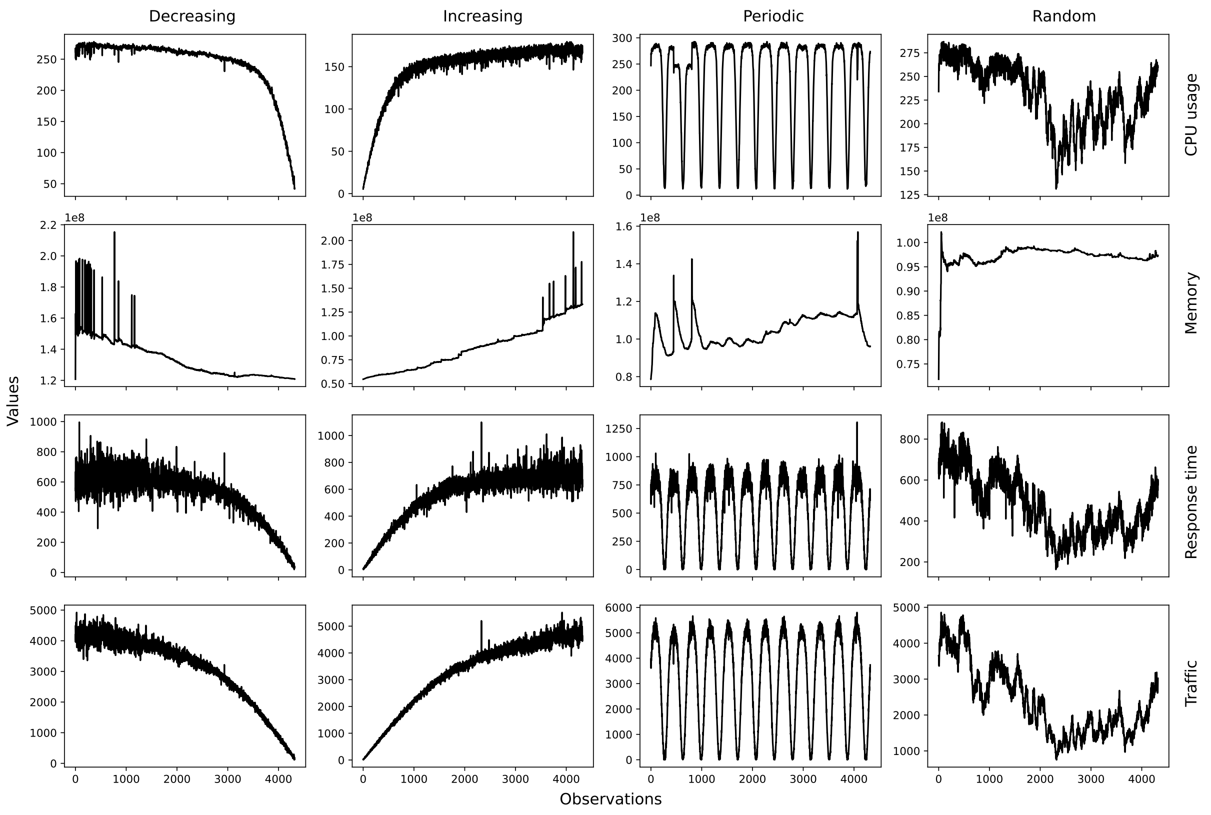
Four OB application performance metrics (CPU usage, memory, response time and traffic) were collected as time series for each workload pattern. These metrics are commonly forecast to anticipate microservices performance degradation for auto-scaling systems (ALIPOUR; LIU, 2017; ROSSI; CARDELLINI; PRESTI, 2020; HUANG et al., 2021; MOHAMED; EL-GAYAR, 2021). The experiments resulted in 16-time series[6] (described in Table B.1) with 4,320 observations per minute of the front-end microservice. Figure B.1 shows all synthetic time series.

4 <https://locust.io/>
5 <https://github.com/GoogleCloudPlatform/microservices-demo>
6 <https://github.com/gfads/mps-methodology/tree/main/time_series>

Table B.1 – Description of the synthetic datasets.

Metric	Series	Trend	Stationary	Frequency	Mean	Median	Std	Size
CPU usage	Decreasing	✓	✗	Minutes	244.28	260.61	44.42	4,320
	Increasing	✓	✓	Minutes	148.47	160.59	33.13	4,320
	Periodic	✗	✓	Minutes	221.67	272.34	89.31	4,320
	Random	✗	✗	Minutes	233.28	237.98	34.27	4,320
Memory	Decreasing	✓	✗	Minutes	134E+6	129E+6	129E+5	4,320
	Increasing	✓	✗	Minutes	875E+5	864E+5	235E+5	4,320
	Periodic	✗	✓	Minutes	104E+6	104E+6	788E+4	4,320
	Random	✗	✓	Minutes	972E+5	974E+5	192+4	4,320
Response Time	Decreasing	✓	✗	Minutes	514.91	561.58	162.41	4,320
	Increasing	✓	✓	Minutes	557.31	624.80	194.04	4,320
	Periodic	✗	✓	Minutes	561.31	691.47	296.56	4,320
	Random	✗	✗	Minutes	476.82	454.16	150.82	4,320
Traffic	Decreasing	✓	✗	Minutes	3,046.08	3,450.78	1,147.25	4,320
	Increasing	✓	✓	Minutes	3,226.51	3,679.96	1,338.79	4,320
	Periodic	✗	✓	Minutes	3,169.47	3,803.33	1,866.10	4,320
	Random	✗	✗	Minutes	2,378.14	2,132.67	968.34	4,320

Figure B.1 – Synthetic time series.



APPENDIX C – MPS SYNTHETIC RESULTS

This chapter presents additional experiments to evaluate the MPS methodology as an alternative to improve the accuracy of forecasting time series from MBAs. The evaluation compares the CFA, which uses only a single forecasting model, against the proposed MPS methodology, which uses a pool of predictive models. The results of experiments on the synthetic series are presented, followed by a statistical evaluation of the findings.

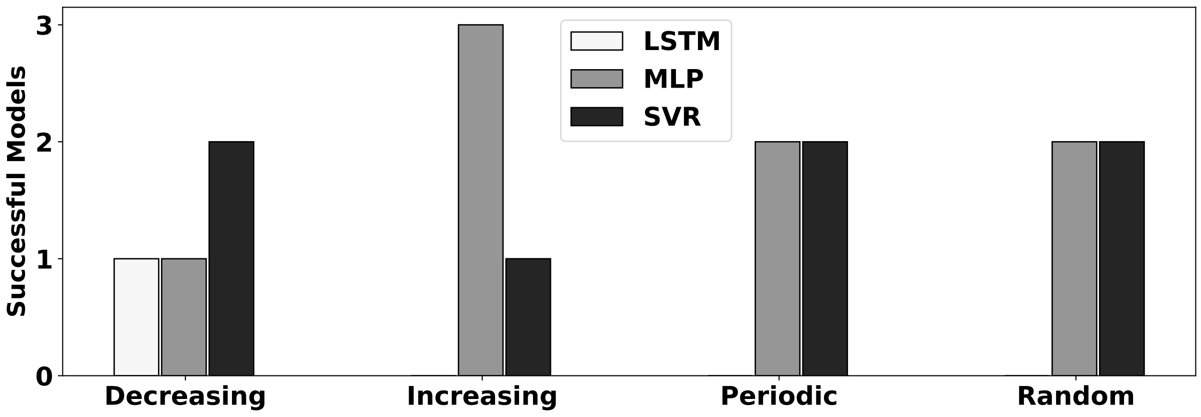
Table C.1 shows the Root Mean Square Error (RMSE) of the CFA models, considering the 16 synthetic series described in Appendix B. The MLP achieved better accuracy on eight series (50%), SVR on seven (43.75%), and LSTM on one (7.5%).

Table C.1 – RMSE results of the CFA models. The best result per time series is in bold. Error values are in 10^{-3} scale.

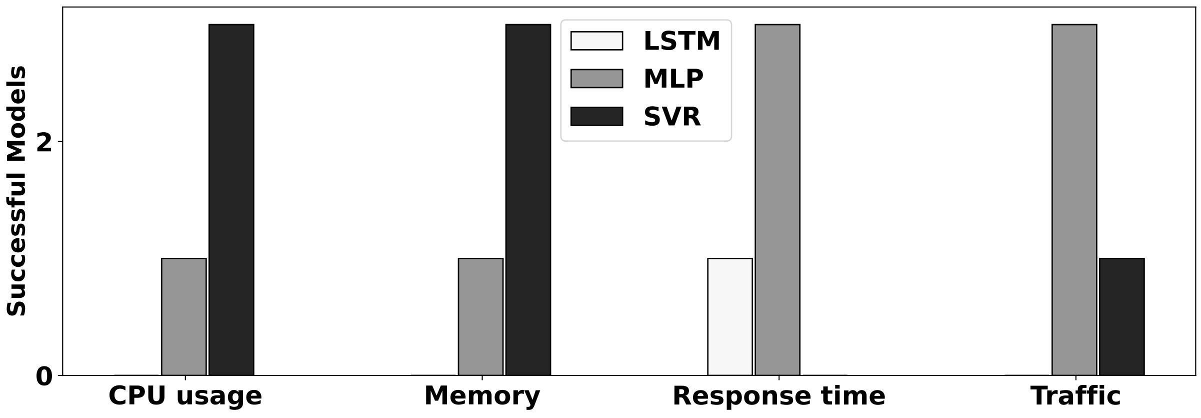
		Models					
Metric	**Series**	**ARIMA**	**LSTM**	**MLP**	**RF**	**SVR**	**XGBoost**
CPU usage	Decreasing	235.60	96.49	11.23	422.89	**7.18**	408.51
	Increasing	30.51	25.34	**24.91**	33.74	24.91	47.92
	Periodic	1,220.88	16.77	32.80	8.66	**7.78**	9.91
	Random	711.42	40.48	33.45	33.09	**32.59**	35.44
Memory	Decreasing	12.57	4.52	0.47	30.86	**0.40**	45.89
	Increasing	101.36	58.83	51.88	218.16	**33.91**	195.08
	Periodic	92.20	40.32	**29.14**	36.27	29.92	42.70
	Random	31.03	6.93	2.79	2.81	**2.59**	3.02
Response time	Decreasing	209.80	**42.21**	49.18	317.32	43.77	298.74
	Increasing	112.57	63.39	**62.61**	65.73	62.64	84.12
	Periodic	3,071.79	38.89	**37.57**	38.91	37.99	41.23
	Random	126.88	42.33	**39.62**	40.91	39.66	40.98
Traffic	Decreasing	154.25	118.86	**9.94**	429.73	10.14	400.68
	Increasing	76.55	34.87	**34.63**	104.03	34.80	137.23
	Periodic	3,494.19	20.83	19.71	20.86	**19.43**	22.78
	Random	125.01	22.99	**19.00**	19.67	19.03	19.75

Figures C.1a and C.1b summarise the successful model (i.e., the most accurate model compared to other CFA models) for each time series and microservice performance metric. Figure C.1a shows that LSTM, MLP and SVR stood out in the decreasing series. MLP and SVR had a similar dominance in periodic and random series. At the same time, MLP was prevailing in increasing series. Figure C.1b shows that MLP was the best model for the response time and traffic series, while SVR was the best for the CPU usage and memory series. ARIMA, RF and XGBoost did not perform exceptionally well in the evaluated series. Nevertheless, these models performed similarly to the most accurate model in some series, such as ARIMA for increasing memory series, SVR for random response time series, and XGBoost for random CPU usage series, among many other examples.

Figure C.1 – Successful CFA models of synthetic series evaluated from the point of view of time series (a) and metrics (b).



(a) Time Series.



(b) Metrics.

However, *none was suitable for forecasting all series*, reaffirming the results observed in the real-world series (see Section 6.5).

Table C.2 compares the RMSE of the MPS approach and CFA on the synthetic series. MPS results include homogeneous and heterogeneous pools using five selection approaches: Dynamic Selection (DS), Dynamic Weighting (DW), Dynamic Weighting With Selection (DWS), Mean and Median (see Section 6.4.3). CFA results are those achieved by MLP, the model with the highest accuracy in the synthetic series, as was done in Section 6.5.

Two peculiarities in the results are worth noting. Firstly, some results are distinct but not noticeable due to numerical approximation. Some examples are Mean and Median ho-

Table C.2 – Accuracy (RMSE) of MPS and CFA for synthetic series. The best results are in bold, and the second-best ones are underlined per time series. Error values are in 10^{-3} scale.

		CFA	Homogeneous					Heterogeneous				
Metric	**Series**	**MLP**	**MEAN**	**MEDIAN**	**DS**	**DW**	**DWS**	**MEAN**	**MEDIAN**	**DS**	**DW**	**DWS**
CPU usage	Decreasing	11.2	18.9	**7.08**	49.4	18.8	18.8	193.5	165.8	7.19	124.6	123.3
	Increasing	**24.9**	24.9	24.9	25.1	24.9	24.9	28.0	26.2	27.1	26.8	26.9
	Periodic	32.8	7.80	7.80	7.99	**7.88**	7.88	204.1	9.38	8.61	83.2	83.6
	Random	33.4	31.7	**31.7**	32.0	31.7	31.7	121.0	32.8	49.4	92.9	94.5
Memory	Decreasing	0.47	0.40	0.40	0.43	**0.40**	0.40	15.2	8.42	3.89	3.23	3.09
	Increasing	51.9	33.1	33.6	33.7	**33.1**	**33.1**	90.8	70.8	33.9	64.3	64.3
	Periodic	29.1	28.8	27.8	30.0	**26.0**	27.8	37.7	36.8	29.7	58.7	58.8
	Random	2.79	2.59	**2.59**	2.60	2.59	2.59	6.03	2.70	7.60	9.02	9.54
Response time	Decreasing	**49.2**	172.0	340.3	107.6	139.7	139.8	123.1	106.8	205.2	81.3	75.5
	Increasing	62.6	62.5	62.5	63.8	62.5	62.5	62.4	**62.4**	72.9	64.6	64.6
	Periodic	37.6	37.5	37.5	38.1	**37.5**	37.5	515.2	38.3	38.9	84.5	84.1
	Random	**39.6**	39.7	39.7	40.3	39.7	39.7	44.6	39.7	41.7	61.2	60.7
Traffic	Decreasing	**9.94**	10.6	10.0	73.1	10.8	10.8	183.2	136.2	10.1	58.2	57.1
	Increasing	**34.6**	34.7	34.7	37.3	34.7	34.7	45.4	34.7	61.2	43.9	43.5
	Periodic	19.7	19.4	**19.4**	19.6	19.4	19.4	583.2	19.8	20.8	43.0	42.5
	Random	19.0	19.0	**18.9**	19.2	19.0	19.0	28.1	19.2	19.9	46.7	47.0

mogeneous in CPU usage (increasing, periodic, and random) and homogeneous algorithms in decreasing memory and increasing traffic series. Secondly, some results are identical. As the selection mechanisms of DW and DWS algorithms are similar, they can select the same models to forecast all test patterns, resulting in the same final accuracy. Some examples are increasing CPU usage and periodic traffic series.

The MPS approach achieved better results than CFA in 68.75% (11 out of 16 datasets). The homogeneous approach attained better accuracy at 62.5%, the heterogeneous approach at 6.25% and the CFA at 31.25%. CFA outperformed MPS in increasing CPU usage, decreasing and random response time series, and decreasing and increasing traffic series. The Median homogeneous was the best algorithm at 31.25% of the series, DW homogeneous at 31.25%, and the Median heterogeneous at 6.25%.

The homogeneous MPS was better than CFA for CPU usage series in 75% of the results. The Median homogeneous was the most accurate approach for decreasing and random, DW homogeneous for periodic and MLP for increasing series. For memory metric, MPS was more accurate than CFA in all series. The DW homogeneous was the best approach for decreasing, increasing and periodic, and the Median homogeneous for increasing series.

The MPS approach and CFA performed similarly for response time and traffic series. The CFA was more accurate for decreasing and random response time series and decreasing and increasing traffic series. Meanwhile, Median heterogeneous was the most accurate for increasing response time, DW homogeneous for periodic response time, and Median homogeneous for periodic and random traffic series.

Although CFA was more accurate than the MPS in some series (5 out of 16), all homogeneous algorithms achieved similar accuracies in these cases, except for decreasing response time series. Only the Median and DS algorithms achieved CFA-like accuracy in most heterogeneous pool results.

It has been observed that many of the findings in the real-world series are also valid in the synthetic ones. For instance, unlike heterogeneous algorithms, all homogeneous algorithms have similar accuracy. Furthermore, in most results, homogeneous algorithms are more accurate than heterogeneous ones. Likewise, the Median is more accurate than the Mean in most heterogeneous results.

Statistical Analysis. Table C.3 shows the percentage accuracy difference between the best homogeneous and heterogeneous algorithms compared to the best, intermediate and worst CFA models in the synthetic series. The algorithms are classified based on their overall accuracy across all series. The highest overall accuracy determines the best algorithm. CFA Intermediate Model (IM) and Worst Model (WM) are the third and last algorithms considering overall accuracy. Based on this criterion, the best algorithms for synthetic series are DW (homogeneous) and DS (heterogeneous), MLP is the Best Model (BM), LSTM is the IM, and ARIMA is the WM.

The symbols $+$, —, and $\sim$ mean that the proposed homogeneous and heterogeneous algorithms attained better, worse, or equal statistical accuracy than the CFA models (see Section 6.4.5). The final three rows of each table summarise the results. Wins are computed as the percentage of series where the proposed algorithm (i.e., homogeneous or heterogeneous) achieves statistically significant improved accuracy over CFA. Loss means the opposite scenario to that described previously. Tie denotes the percentage of series where the proposed algorithm and CFA accuracy were not statistically different.

In the synthetic series, the best homogeneous approach was equal to or more accurate than BM in 62.5% of cases and as accurate as or more accurate than IM and WM in all cases. On the other hand, the best heterogeneous approach achieved equal to or better accuracy than BM in only 31.25% of the series. However, the best heterogeneous approach showed equal or better results in 87.5% and 100% of the cases compared to IM and WM, respectively.

Considering the 48 comparisons between the MPS approaches and CFA (i.e., 16 datasets with three CFA variations), the best homogeneous approach obtained 40 wins (83.33%), 2 ties (4.17%) and six losses (12.5%). On the other hand, the best heterogeneous approach obtained

Table C.3 – Percentage (%) difference between the best homogeneous and heterogeneous algorithms compared to BM, IM, and WM in synthetic series computed using Equation 6.5. The statistical result was obtained using the Diebold Mariano statistical test (see Section 6.4.5). DW is the best homogeneous algorithm, DS is the best heterogeneous algorithm, MLP is the BM, LSTM is the IM, and ARIMA is the WM

		Homogeneous			**Heterogeneous**		
Metric	**Series**	**BM**	**IA**	**WM**	**BM**	**IM**	**WM**
CPU usage	Decreasing	-67.06 (−)	92.78 (+)	92.04 (+)	35.96 (+)	97.23 (+)	96.95 (+)
	Increasing	-0.05 (−)	15.65 (+)	18.32 (+)	-8.93 (−)	8.16 (+)	11.06 (+)
	Periodic	76.30 (+)	38.86 (+)	99.36 (+)	73.75 (+)	32.29 (+)	99.29 (+)
	Random	5.19 (+)	13.78 (+)	95.54 (+)	-47.78 (∼)	-34.39 (∼)	93.05 (+)
Memory	Decreasing	15.26 (+)	97.76 (+)	96.85 (+)	-733.3 (−)	77.99 (+)	69.02 (+)
	Increasing	36.26 (+)	76.12 (+)	67.37 (+)	34.64 (+)	75.52 (+)	66.55 (+)
	Periodic	10.80 (∼)	32.12 (∼)	71.80 (+)	-1.75 (∼)	22.57 (+)	67.84 (+)
	Random	7.11 (+)	46.78 (+)	91.64 (+)	-172.3 (−)	-56.05 (∼)	75.50 (+)
Response time	Decreasing	-184.1 (−)	22.27 (+)	33.40 (+)	-317.3 (−)	-14.17 (−)	2.18 (+)
	Increasing	0.19 (+)	3.21 (+)	44.49 (+)	-16.42 (−)	-12.90 (−)	35.25 (+)
	Periodic	0.20 (+)	3.61 (+)	98.78 (+)	-3.53 (−)	0.01 (∼)	98.73 (+)
	Random	-0.26 (−)	4.57 (+)	68.70 (+)	-5.15 (−)	-0.08 (∼)	67.17 (+)
Traffic	Decreasing	-8.20 (−)	96.08 (+)	93.03 (+)	-2.06 (−)	96.30 (+)	93.42 (+)
	Increasing	-0.27 (−)	50.01 (+)	54.64 (+)	-76.86 (−)	11.83 (+)	20.00 (+)
	Periodic	1.66 (+)	7.02 (+)	99.45 (+)	-5.42 (−)	0.33 (∼)	99.41 (+)
	Random	0.17 (+)	11.05 (+)	84.82 (+)	-4.89 (−)	6.55 (+)	84.06 (+)
Wins		56.25%	93.75%	100%	18.75%	56.25%	100%
Ties		6.25%	6.25%	0%	12.5%	31.25%	0%
Loss		37.5%	0%	0%	68.75%	12.50%	0%

28 wins (58.33%), seven ties (14.57%) and 13 losses (27.1%). These results indicate that the homogeneous approach is more accurate than the heterogeneous approach for forecasting microservices time series. This finding was also demonstrated in the experiments with real-world time series (see Section 6.5).

When focusing solely on BM results, the best homogeneous approach was equal to or better in 62.5% of cases, while the best heterogeneous approach only achieved this in 31.25% of cases. Therefore, since the best CFA model is unknown before analysis, these results suggest that using a homogeneous approach can enhance the accuracy of the forecast adaptive system and mitigate the CFA problem. At the same time, opting for a heterogeneous approach aiming for the highest accuracies is not recommended.

APPENDIX D – PMA FORECASTING STRATEGIES DETAILS

Table D.1 details the accuracy and parameters used by the PMA univariate and MPS strategies for each workload and application in the experimental evaluation of Chapter 7.

Table D.1 – Details of the accuracy and parameters used by the PMA univariate and MPS strategies for each workload and application in the experimental evaluation of Chapter 7.

		Univariate						**MPS**	
App	**Workload**	**ARIMA**	**LSTM**	**MLP**	**RF**	**SVR**	**XGBoost**	**Model**	**Pool (Size)**
Cart	Alibaba	76.27	48.38	**2.88**	5.70	*2.91*	19.38	MLP	Homo (100)
	Clarknet	32.15	39.10	**5.79**	*5.84*	5.87	9.98	MLP	Homo (100)
	NASA	46.02	49.03	**1.33**	2.04	*1.51*	6.62	MLP	Homo (100)
	WorldCup98	40.00	18.96	**4.24**	*4.87*	4.95	6.29	MLP	Homo (100)
Checkout	Alibaba	7.08	5.23	**0.33**	0.44	*0.39*	0.48	MLP	Homo (100)
	Clarknet	38.89	428.83	**1.47**	1.76	*1.51*	4.76	MLP	Homo (100)
	NASA	26.44	29.27	**0.90**	1.44	*0.95*	1.92	MLP	Homo (100)
	WorldCup98	15.45	6.00	*0.58*	0.73	**0.52**	0.71	SVR	Homo (100)
Currency	Alibaba	31.88	48.20	*2.35*	2.94	**2.14**	17.24	SVR	Homo (100)
	Clarknet	16.43	14.19	**4.90**	*4.98*	5.15	8.79	MLP	Homo (100)
	NASA	73.50	49.93	*15.08*	15.54	**13.80**	24.73	SVR	Homo (100)
	WorldCup98	11.66	17.78	**3.38**	*3.43*	3.53	9.30	MLP	Homo (100)
Frontend	Alibaba	1.22	6.26	0.40	*0.40*	**0.39**	0.81	SVR	Homo (100)
	Clarknet	2.34	2.22	**0.48**	0.61	*0.49*	2.01	MLP	Homo (100)
	NASA	21.12	20.62	*5.23*	5.73	**4.49**	11.09	SVR	Homo (100)
	WorldCup98	2.57	1.50	*0.54*	0.55	**0.54**	0.56	SVR	Homo (100)
Product	Alibaba	9.71	206.97	**1.41**	1.55	*1.49*	1.53	MLP	Homo (100)
	Clarknet	16.11	17.01	*3.15*	3.68	**3.04**	7.34	SVR	Homo (100)
	NASA	34.42	43.39	**7.83**	9.82	*8.71*	9.84	MLP	Homo (100)
	WorldCup98	16.56	15.81	**3.77**	*3.78*	4.00	4.22	MLP	Homo (100)
Recommendation	Alibaba	24.25	19.36	**0.29**	0.41	*0.29*	0.45	MLP	Homo (100)
	Clarknet	110.60	25.40	**1.52**	1.97	*1.69*	2.79	MLP	Homo (100)
	NASA	45.48	73.40	**20.29**	*20.88*	26.06	21.58	MLP	Homo (100)
	WorldCup98	224.09	8.74	0.92	*0.70*	**0.61**	2.00	SVR	Homo (100)
Cars	Alibaba	30.27	28.44	14.65	**10.13**	13.85	*12.62*	RF	Homo (100)
	Clarknet	25.68	18.20	102.76	**3.40**	*4.55*	9.38	RF	Homo (100)
	NASA	37.65	34.66	**11.43**	*11.67*	12.08	11.69	MLP	Homo (100)
	WorldCup98	34.48	3.77	*0.44*	0.50	**0.44**	0.51	SVR	Homo (100)
	Alibaba	33.01	6.00	*0.22*	0.28	**0.22**	0.33	SVR	Homo (100)
	Clarknet	58.25	8.00	*0.79*	0.83	**0.79**	0.87	SVR	Homo (100)
	NASA	22.92	28.07	*0.74*	1.02	**0.62**	1.62	SVR	Homo (100)

Discounts	WorldCup98	49.57	106.84	**6.84**	7.35	*7.34*	8.29	MLP	Homo (100)
Flights	Alibaba	23.59	8.48	**1.91**	2.03	*1.97*	2.18	MLP	Homo (100)
	Clarknet	22.18	19.15	2.51	2.62	*2.41*	**2.03**	XGBoost	Homo (100)
	NASA	41.67	58.15	*1.02*	1.62	**0.95**	2.49	SVR	Homo (100)
	WorldCup98	37.45	6.44	**0.49**	0.70	*0.51*	1.42	MLP	Homo (100)
Hotels	Alibaba	10.56	0.85	**0.08**	0.12	*0.08*	0.11	MLP	Homo (100)
	Clarknet	11.49	10.79	1.50	1.25	**1.17**	*1.23*	SVR	Homo (100)
	NASA	50.18	49.79	14.31	*9.05*	**8.91**	11.82	SVR	Homo (100)
	WorldCup98	10.78	8.01	**2.04**	2.17	*2.07*	6.23	MLP	Homo (100)
Insurances	Alibaba	34.34	2.02	*0.34*	0.43	**0.28**	0.45	SVR	Homo (100)
	Clarknet	44.31	15.22	*1.01*	1.23	**0.96**	1.25	SVR	Homo (100)
	NASA	27.04	31.72	**8.42**	*8.59*	9.18	10.83	MLP	Homo (100)
	WorldCup98	31.01	64.36	**2.97**	*3.04*	3.08	7.91	MLP	Homo (100)
Travels	Alibaba	7.23	2.24	**0.21**	0.68	0.64	*0.42*	MLP	Homo (100)
	Clarknet	11.64	10.70	*4.23*	4.28	**4.17**	4.30	SVR	Homo (100)
	NASA	29.83	25.02	**8.68**	*9.56*	9.70	10.05	MLP	Homo (100)
	WorldCup98	26.63	17.72	**3.67**	3.79	*3.71*	4.87	MLP	Homo (100)
Quarkus	Alibaba	50.77	7.24	**2.94**	*3.27*	3.65	3.29	MLP	Homo (100)
	Clarknet	27.66	15.91	5.82	**5.34**	*5.35*	5.84	RF	Homo (100)
	NASA	27.56	29.01	**10.07**	*10.78*	10.99	13.62	MLP	Homo (100)
	WorldCup98	13.26	15.31	**4.07**	*4.11*	4.53	4.30	MLP	Homo (100)
Daytrader	Alibaba	8.90	4.36	**0.81**	1.03	*0.82*	1.34	MLP	Homo (100)
	Clarknet	10.93	9.21	*2.22*	**2.09**	2.32	4.10	RF	Homo (100)
	NASA	22.23	23.99	*8.62*	9.72	**8.47**	12.73	SVR	Homo (100)
	WorldCup98	6.59	5.73	1.54	**1.47**	1.55	*1.52*	RF	Homo (100)