FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Designing and building a microservices-based time series forecasting system

João Ruano Neto Veiga de Macedo

Mestrado em Engenharia Informática e Computação

Supervisors: Jácome Miguel Costa da Cunha (FEUP) and Francisco Amorim (LTPlabs)

July 25, 2022

Designing and building a microservices-based time series forecasting system

João Ruano Neto Veiga de Macedo

Mestrado em Engenharia Informática e Computação

July 25, 2022

Abstract

Time series analysis and forecasting are widely used techniques in business optimization and analytics, as enterprises periodically record data points of an increasing variety of internal metrics. Categorizing and organising a large number of correlated input time series and forecasting these metrics into the future are processes often used to inform an organization's decision making process.

The emerging paradigm of microservices-based architectures aims to decompose systems into decoupled, single-responsibility components analogous to business capabilities. For time series analysis and forecasting the main benefits of such a paradigm would be horizontal scalability through service replication and the organizational impact of maintainability and code reusability in the context of LTPlabs, a Portuguese analytical-driven management consultancy company.

In this work, we propose the applicability of a microservices architecture decomposition strategy to a Time Series Forecasting solution employing univariate, ensemble, and hierarchical forecasting models.

A dataflow driven analysis for service extraction was used to decompose the system. The communication was redesigned to adhere to an event-based paradigm. Additional adjustments were made to avoid documented microservices anti-patterns.

We used a microservices assessment framework supported by execution time, computational cost, and forecasting accuracy metrics for comparative analysis of an implemented prototype with the pre-decomposition system.

The decomposition approach produced the desired outcomes, as the execution time was lowered by over 50% through independent service replication, while the costs and forecasting error metrics remained comparable.

Keywords: Time-series forecasting, Time-series analysis, Microservices, Monolith decomposition

Agradecimentos

Quem me acompanhou nestes anos que me formaram ficará sempre nas minhas memórias. Quer nas maiores decisões como nos mais simples momentos do quotidiano, relembro as palavras de sabedoria, apoio, e carinho, dadas tão generosamente ao longo dos últimos cinco anos. Resta-me escrever os meus breves agradecimentos a algumas dessas pessoas, sempre pequenos ao lado das marcas que deixaram em mim.

Ao Francisco Amorim, por toda a mentoria e amizade, por cada revisão e conversa, e por tudo o que me ensinou, sempre com a calma, confiança e sensatez que o caracterizam;

Ao Professor Jácome Cunha, pelo apoio e disponibilidade ao longo deste percurso, e por sempre me desafiar a ir mais além, nunca esquecendo o pragmatismo e rigor;

Às pessoas incríveis que tive a oportunidade de conhecer nestes seis meses na LTPlabs, com quem já guardo memórias desde a mesa do Avis aos Picos da Europa; ao João Alves, por me lembrar de jogar com as minhas forças sem deixar de trabalhar nas minhas fraquezas;

À minha família, os constantes apoiantes do meu percurso académico que já perfaz duas décadas; à minha irmã Rita, o meu porto seguro, ao meu irmão Miguel, o meu amigo e professor mais antigo, e aos meus pais Teresa e Rui, a quem devo tanto do que sou;

Aos meus amigos, por estes anos de tanto crescimento e vivências partilhadas,

Obrigado.

João Macedo

"La vida no es la que uno vivió sino la que uno recuerda y como la recuerda para contarla"

Gabriel García Márquez

Contents

List of Figures

List of Tables

Abbreviations

TSA	Time Series Analysis
MSA	Microservices-based Architecture
SaaS	Software as a Service
TSF	Time Series Forecasting
SOA	Service Oriented Architecture
API	Application Program Interface
SKU	Stock-Keeping Units
CQRS	Command Query Responsibility Segregation
MA	Moving average
AR	Autoregression
ARIMA	Autoregression Integrated Moving Average
CRISP-DM	Cross Industry Standard Process for Data Mining
TSFM	Pre-decomposition Time-series forecasting module
GBM	Gradient Boosting Machine
DFD	Dataflow Diagram
AMQ	Asynchronous Message Queue
AMQP	Asynchronous Message Queue Protocol

Chapter 1

Introduction

Organizations are collecting increasingly large amounts of time-stamped observations of various internal metrics, structured as Time Series data, such as product sales and production rates. This data structure is characterized by a sequence of values indexed in time. The applicability of Time Series Analysis (TSA) and Forecasting (TSF) [59] is of high interest to a broad range of fields, both in academic and business contexts. For the successful application of TSA and TSF techniques, manual calculations are not suitable, requiring the interaction of an analyst with a computer system [46].

Structuring the multiple techniques in an integrated system is therefore of interest to organizations and the field of software architecture. From an architectural viewpoint, the requirements for these systems are the ability to fulfil different parallel workflows, allow horizontal scalability of individual TSA and TSF techniques, and evolve by including new techniques. The current answer for these requirements is microservices-based architectures (MSA) that emerged from the needs of the industry and just recently are being increasingly studied by academia [9].

MSAs are a recent development on the Service Oriented Architecture (SOA) paradigm, generically consisting of practices aiming to isolate computational logic in independently deployable, single-responsibility contexts. Often framed around business logic, the granularity of the implemented services should be as fine as possible while remaining analogous to business capabilities. Therefore, this paradigm is relevant for the software engineering industry as it motivates code reusability and improves readability, maintainability, and horizontal scalability through service replication [49]. From a technological perspective, recent releases of tooling for containerization, service discovery, monitoring, container orchestration, and other aspects of the current understanding of a mature microservices-based architecture have hastened development and allowed for the management of ever-smaller independent services [32].

To facilitate the use of the evolving techniques included in these areas and maintain competitiveness, organizations are shifting from in-house custom tooling to relying on Software as a Service (SaaS) alternatives, outsourcing the responsibility of the computational logic to third parties [58]. This shift creates the need for SaaS solutions that support the application of a diversity of possibly useful techniques, supporting new feature development and ease of integration.

To respond to these needs, we propose following the principles of microservices-based architecture design to create a flexible set of independently deployable, finely granular TSA and TSF services.

Context

In the current industry landscape, the analysis and forecasting of these time-ordered data structures are highly relevant for Business Intelligence and Business Analytics goals, respectively. The ability to predict the evolution of the time series beyond a forecasting horizon allows organizations to look into the future, informing present decisions and creating value for the organization.

For example, predicting future product sales allows for prevention of stock-off periods, optimization of product allocation and transportation, and the application of Lean Production practices informed by the upstream demand predictions. Even without considering the applications of TSF in other industry contexts, such as service-demand prediction or financial market forecasting, the case of sales data analysis and prediction is in itself context-dependent, as Sales data time-series can be extremely heterogeneous between stock-keeping units (SKU) [8]. Many TSF systems are implemented case by case after a previous step of model fitting, evaluation, and selection for each SKU, or at most standardized through meta-learning and ensemble techniques.

Modern programming language package-management systems include in their listed libraries implementations of some versions of TSA and TSF techniques (e.g., CRAN for R [29], DarTS for Python [27]), allowing researchers to conveniently apply the techniques needed to extract the target insights.

Organizations working on their specific contexts have the ability and the motivation to create specific tooling for their needs. In the consulting industry however, as is the case of LTPlabs, there is a need to handle heterogeneous time series data from different clients and contexts, with distinct requirements.

This work was conducted within LTPlabs, an INESC-TEC (Institute for Systems and Computer Engineering, Technology and Science) spin-off advanced-analytics consultancy company [1]. The validation of our proposed design was conducted with empirical data from two anonymized clients in distinct industries, providing a non-context-specific analysis.

1.1 Motivation

An example of a mature SaaS solution for Time series forecasting is Amazon Forecast [4], built on top of distinct services belonging to the company's enterprise software ecosystem. Although this could be considered a firm implementation of the practices of microservices architecture design, these services do not compartmentalize specifically the TSA and TSF-related tasks.

These are implemented either in a monolithic fashion through AWS SageMaker in the case of TSF and data preparation tasks or through a proprietary state machine design through the "AWS

Step Functions forecasting automation workflow" for the TSF tasks. These tasks include the non-expandable processes of fitting, evaluating, and forecasting the limited selection of models (i.e., CNN-QR, DeepAR+, Prophet, NPTS, ARIMA, and ETS), integrated only internally through the proprietary, non-transparent forecasting algorithms [2].

We identified a single design proposal for an MSA-based time series forecasting system, in the relatively concise lecture notes by Uzun et al., for the stated purpose of this dissertation [63]. Citing the authors:

> "One of the future plans is to extend the architectural model with new services that potentially could accumulate other often used methods when working with time-series data" [63, chap. 5.2 Future Research]

We will attempt to build upon the design by decomposing a current monolithic solution for TSF through a data-driven approach, using event sourcing, allowing for more complex techniques including Ensemble and Hierarchical Forecasting. This decomposition aims to extend the proposed design's functionality while improving the scalability of the current monolithic solution.

1.2 Objectives

The central objective of this dissertation is to design and implement a microservices-based architecture design, allowing different diagnostic and deployment-focused workflows through a flexible set of independently deployable, finely granular TSA and TSF services.

We will explore various alternative strategies for service decomposition and apply the most suitable for our context to define service boundaries. The implementation will be done iteratively through an ordered list of services to be extracted. It will also consider the reusability of the developed components in the context of an internal organizational shift to a Microservice-based architecture paradigm.

To validate the design and implementation of the MSA system, we draw qualitative and quantitative comparisons between a prototype of our proposal and the pre-decomposition diagnostic solution.

Using the previously mentioned anonymized LTPlabs' clients' data, forecasting accuracy metrics, execution time, and computational resource usage, along with the assessment framework for microservices proposed by Auer et al. [7] we will compare the original monolithic application with the set of developed microservices.

1.3 Document Structure

The current chapter 1 provides an overview of the future complete dissertation, including an introduction, context, motivation, objectives, and this present sub-section.

Chapter 2 documents the literature review conducted for the purpose of this work, including an analysis of MSAs, TSF techniques, and a joint analysis of the terms.

In Chapter 3 we detail the current monolithic approach for time series analysis and describe the main workflows and system requirements. We give additional insight into the preexisting factors favouring the implementation of the new approach.

Chapter 4 follows the process of the service extraction and defines the microservices created and how the functional requirements are met by the presented proposed design.

Chapter 5 measures the outcomes of the decomposition approach by comparing metrics gathered from multiple executions of the original and decomposed solution.

Chapter 6 concludes the present work and includes suggestions for future research on the application of a MSA for time series forecasting.

Chapter 2

Literature Review

This section analyzes the current literature from academic sources and industry practitioners on the two knowledge areas relevant to this work, the Microservices Architectural style and TSF methods.

A separate initial analysis is presented for both areas, followed by a short review of recent work combining them. The separate analysis provides a review of each field restricted by the scope of our base design, as the extension of literature on time-series analysis in particular demands it. We include the time series techniques relevant to this work and the relevant principles and areas of concern of MSAs to guarantee that the proposed design can evolve with the continuous development of new TSA and TSF techniques, adapting to future research.

In the last subsection, we analyzed the currently limited literature on MSA designs for TSF.

2.1 Microservices Architecture

The use of the term microservices dates back to May 2011, from a software architecture workshop held near Venice, Italy [36]. The principles later encapsulated as microservices were already in use by industry professionals. Adrian Cockcroft had previously described a similar approach used at Netflix as a fine-grained SOA [18], expanding in its principles, benefits, and anti-patterns in November 2010 [16]. Soon after these ideas were presented by James Lewis at the 2012 edition of the 33rd Degree Conference in Krakow [35], the use of the term "Microservices" was cemented by the same group of practitioners present in the original 2011 workshop.

The defining features of microservices and how they differ from SOA remained in discussion until long after its inception. This separation was later tackled by Olaf Zimmermann in 2016 [70] when he defined in this study seven microservice tenets recurrent in the field. Through these themes, microservices can be broadly described as:

1. Single-responsibility units that encapsulate data and processing logic, exposing fine-grained interfaces remotely, independently deployed;

2. Conceptualized by Business-driven development practices and pattern languages;
3. Following Cloud-native application design (e.g., isolated state, distribution, elasticity, automated management, loose coupling);
4. Allowing multiple computing and storage paradigms, and polyglot services;
5. Deployed with Lightweight containers
6. Practicing continuous delivery and decentralization during service deployment
7. Employing light and automated approaches for holistic management (DevOps)

Also tackled in this study were the contrasting definitions of microservices prevalent at the time of its publication, comparing them with the existing definitions of SOA. The two explored definitions include Lewis and Fowler's nine characteristics of microservices [36] and Newman's seven principles [49]. The author analyzed these definitions through a comparative mapping of the ideas present in Lewis and Fowler's characteristics and Newman's principles. Following this, Zimmermann further compares these definitions with SOA pendants identified through a review of the literature available at the time, including books [24][23] and practitioner's publications [21][39][48][71].

The author concludes that the analyzed definition supports the idea that Microservices is a specific implementation of SOA, representing an evolutionary and complementary view to the Service orientation paradigm prevalent in the early 2000s. Specifically, the concepts of "business orientation, polyglot programming in multiple paradigms and languages, and design for failure" are included in SOA and transferable to MSAs. At the same time, the latter approach emphasizes decentralization and automation and independent service deployment. In the following year to Zimmermann's publication, the development of microservice architectures was detailed by Dragoni et al. [18], framing them as the second iteration of SOA.

Zimmermann also critiques the standard definition of microservices analogous to internal organizational contexts, following Conway's law on system design. He asserts that it has limited use for practitioners, as this view easily violates the stated single-responsibility characteristic of the independent modules [70]. In 2019, Zhang et al. conducted a practitioner's survey to identify the differences between the envisioned benefits and challenges, and the reality of implementation considering the actual costs and benefits in practice, determining the industry's perceived pains. Here, the surveyed practitioners identified an inverse of Conway's law, as the decomposition of the system influenced the organizational structure [69].

The benefits of componentization via services, as outlined by practitioners in this study, were independent upgradeability, scaling, development, testing, and deployment [69]. These benefits connect to the pain of "chaotic independence", where true independence between services is not adequately realized, leading to changes or new features affecting multiple services, and inappropriate boundaries or service versions create difficulties in testing. This pain tends to be more

predominant when services are over fine-grained. We will explore these concepts in the section 2.1.1.

A practice identified among the practitioners surveyed in this study was to "Compromise with database decomposition". Most practitioners did not decompose their original database when migrating to MSA, despite their awareness that this contributes to properly decoupling the microservices [69]. Database decomposition contributes to the pain of "Data Inconsistency", as a single transaction can be much more complex to handle in a decentralized data management approach when compared to the centralized alternative. We will further research these aspects in the section "Data Concerns" 2.1.2.

Also in this study is the analysis of the practice of "Choosing communication protocol", where HTTP Restful API, RPC, and Lightweight messaging were the common design choices, where the Ecosystem's maturity, the activity of its community, and the documentation were the most cited selection criteria for technology selection [69]. From this, Zhang et al. identified the pain of "Complexity of API management", where the complexity of consistently designing, managing, and documenting the APIs provided by the multiple microservices hinders and slows down development. We will explore the literature on communication patterns in the section "Communication" 2.1.3.

2.1.1 Refactoring Monoliths

The growth of MSA-related publications since 2015 [25] has led to refactoring efforts from monolithic architectures to microservices across the industry.

By applying refactoring strategies, microservices' practices have successfully tackled transitions from monolithic systems by gradually implementing new features or refactoring parts of the system, integrating them, and avoiding the need to develop an entirely new one from the start. [17]

To date, various software industry players moved from a monolithic paradigm to microservices, such as Google and eBay, Netflix [42]. The recent popularity of this flexible architectural design inevitably led to heterogeneity between implementation decisions and adopted patterns [11]. Taibi and Lenarduzzi contributed to this view by identifying harmful, frequent practices when implementing MSAs, defining the characteristics these systems must possess to achieve the perceived benefits [61].

A key aspect of migrating to MSA is extracting services from the system. In 2017, Mazlami et al. defined a formal methodology for service extraction, contrary to the preexisting informal ones [42]. After a number of different approaches were proposed, in 2021, Kirby et al. identified three distinct types of service relationships in the service extraction literature [33]. Structural relationships are defined through static or dynamic code analysis, including method calls, data dependencies, and shared resources. Semantic relationships identify elements based on business domain and developer naming conventions, for instance, class name similarity. Evolutionary relationships mine information on code repositories, identifying possible services that can be maintained and developed by a team through commit or contributor similarity.

For the purpose of this work, the Dataflow Driven analysis for service extraction proposed by Chen et al. [15] will be used, as we are working in a very data-driven domain. This approach involves creating a Dataflow diagram (e.g. Figure 3.1) and applying certain rules to simplify it and transform it into a Decomposable purified dataflow diagram from where the services are extracted. The application of these rules is detailed further for our use case in section 4.1.1.

2.1.2 Data Concerns

Of particular relevance for this work is the issue of data sharing concerns between microservices. "Shared persistence" was identified by Taibi et al. as a technical MSA anti-pattern characterized by distinct services accessing the same relational database. The solutions proposed by the authors are creating independent databases for each service, a shared database with a set of private tables for each service, and a private database schema for each service [62]. A recent mapping of data management solutions was conducted by Ntentos et al. [50].

In 2020, Munonye and Martinek compared five distinct data string patterns in MSA, including a shared database (anti-)pattern, three variations of the per-service database pattern, and the Command Query Responsibility Segregation (CQRS)/ Event Sourcing pattern [47]. Also compared in this work is the use of either a Relational Database Management System or a Document-Store Database. In this work, we will assume transferability between the findings drawn from the use of the latter and its umbrella category, key-values store NoSQL databases.

Before analyzing the findings of this comparative analysis, we will highlight the characteristics of the CQRS/Event Sourcing pattern. Using Richardson's microservices.io project's definition [54], CQRS is a response to the need to perform queries retrieving data split between services. In this pattern, a read-only view of the split microservices' data allowing the execution of the intended queries is managed by a service subscribing to the relevant events in the case of event-driven designs. Also referring the Richardson's microservices.io project [55], Event Sourcing is a pattern applicable only in conjunction with the CQRS pattern, used to maintain atomicity of Database transactions. For the implementation of this pattern, through the CQRS interface service, we define business entities as a sequence of entity state-changing events that are added to an appended-only list, representing the entity. When querying the state of the entity, the list of state-changing events is replayed. Both CQRS and Event Sourcing are explored in more depth in Richardson's book "Microservices patterns: with examples in Java" [56].

Returning to the findings of Munonye and Martinek [47], we highlight the better performance exhibited by Document-store options when compared to Relational Database Management Systems and the better performance of CQRS/Event Sourcing when compared both to the single Database (anti-)pattern and the database per-service pattern.

2.1.3 Communication

Communication between services is a key aspect of the design choices taken when migrating to a microservices-based system. Transforming a simple function call into a network-level interaction

transfers complexity from within the distinct logic components to the way they collaborate with each other [43]. Only through collaboration can a microservices-based system satisfy a business need. The two main patterns followed to structure the communications are orchestration and choreography.

The choreography pattern is achieved when no unit controls the end-to-end workflow of a business process that encompasses various services. This is often achieved through the use of an Asynchronous message queue with a publisher-subscriber system, implemented through a message broker such as Apache Qpid or RabbitMQ. Here, services subscribe to the types of events they need to fulfil their responsibility and publish events informing the system of state changes [56]. The advantages of choreography are the very loose coupling and the low chattiness, where data is exchanged only when there are state changes. At the same time, the weaknesses include poor process visibility, a more complex design, weak atomicity when dealing with cross-cutting data concerns, and the undetermined response time inherent to the process's eventual completeness in case of a service failure, guaranteed by the non-acknowledgement and eventual requeuing of the consumed event.

The orchestration pattern entails the existence of a composite microservice, the non independently deployable "Orchestrator", invoking a set of atomic Microservices informed by an internal representation of the complete processes workflow. Here, atomic microservices do not invoke each other, as the orchestrator always manages this. Usually, these calls are made in a request-response way through REST Endpoints [56]. Compared to choreography, this achieves clearer process visibility, a simpler design, stronger atomicity, and a more predictable response time. The trade-off is the tighter coupling of services, particularly the orchestrator, as a main point of failure for the satisfaction of the business need and the higher chattiness as there is data exchange between services at each of the workflow steps.

It is often the case that microservices architectures manage one single business entity. In these cases, implementing the workflow through orchestration is much easier than through choreography. However, this limits the evolution of the system, as changes to the model are of increasing complexity [13].

2.2 Time Series Forecasting

Time series forecasting is of high interest for many real-world problems, where we often find characteristics of non-linearity. As an example, in the case of sales forecasting, sudden unexpected stock-off periods, jumps in demand caused by the application of non-periodic promotions, or the simple chaos inherent to human purchasing behaviour. Naive and classical techniques often assume that the target series are linear and stationary, while machine-learning techniques can often model more complex behaviour. In section 2.2.1 we explore the forecasting models relevant for this work.

In the following sections 2.2.2 and 2.2.3 we also explore aggregation techniques, informed by the problem domain, for the purpose of increasing information density in the target time series.

2.2.1 Univariate forecasting models

Univariate models for forecasting attempt to predict a single target time series, while multivariate models may include multiple target time series, simultaneously predicting their evolution. In this section, we will explore a variety of univariate models with different advantages, ordered by their complexity. Less complex models tend to be more explainable and less resource-intensive, and should therefore be prioritized when achieving similarly accurate outputs.

2.2.1.1 Naive Models

Naive models utilize simple heuristics for generating fast predictions with very few computational resources. Baseline predictions include the Naive Drift Model, joining the first point of the Time series and the last known with a straight line and extending it into the future, the Naive Mean Model, which always predicts the mean value of the series, and the Naive Seasonal Model, always predicting the historical value from a predefined number of time steps before the target time [31]. Naive models generate essential baseline predictions, validating, by comparison, the more complex models.

2.2.1.2 Classical Models

Classical models encompase non-machine-learning models for TSF. Of relevance for our design proposal are Exponential Smoothing, moving average (MA), autoregression (AR), and AR integrated MA (ARIMA) models.

A predecessor to Exponential Smoothing models is the Simple Moving Average model. In this model, the prediction is the average of a predefined number of past observations before the target. Exponential smoothing models build upon this concept. For prediction of the target observation, exponential smoothing models attribute exponentially diminishing weights to past observations. The Simple Exponential Smoothing assumes that the time series has no trend or seasonality. By additionally modelling the trend of the time series and the seasonality component, we reach the reference for explainable exponential smoothing methods, Holt-Winters' Exponential Smoothing, also called triple exponential smoothing [66].

AR models predict the evolution of the series only from a set of values of the sequence called lags, constituting a rolling window capturing the information in the time series in a recursive fashion. This family of models is classified as "long memory" as every prediction of a timestep is a combination of the ever-diminishing effect of the training window's observations into the past. Moving Average (MA) models are very similar to AR-type models but predict the values of the target timesteps by a function of the error factor of the predictions done to past observations by the model. Initializing with a naive model such as the Naive Mean Model, MA models capture ever-increasing information about the observations until the current timestep. For this reason, these models are called "short memory" models [6].

The current standard, achieving similar results with significantly less complexity than AR or MA models, are ARIMA models that balance a set of MA and AR terms in their predictions.

ARIMA models assume that the time series is stationary, although this can be achieved through iterative differentiating processes, transforming the original sequence into the sequence of differences between consecutive terms [31]. This process is called Integration. ARIMA-based models have been extensively studied in literature [60], and the ever-growing family of ARIMA models include Seasonal ARIMA, considering seasonality, VARIMA, its multivariate counterpart, as well as many other variations.

2.2.1.3 Machine Learning Models

While the functioning of machine learning models is beyond the scope of this work, their recently growing relevance in both the industry and academia leads to a need to accommodate for their inclusion in our design proposal. This demands a high-level understanding of their requirements and outputs for proper integration in the architecture.

Regression Models used for TSF include various extensively researched algorithms, such as Random Forest [19], Linear Regression [20], and Gradient Boosting Machines [5] (GBM).

Neural networks can also be used for TSF, incorporating either convolutional layers for input dimensionality reduction, recurrent layers for iterative scanning of the series by a window, or an attention-based model aggregating temporal features through learned weights [20]. Current implementation industry standards of this type of model for TSF include DeepAR [57], NBEATS [51], and Temporal Fusion Transformers [37].

2.2.2 Ensemble Models

Joining predictive approaches has been shown to be relevant for increasing the accuracy of predictions drawn from the available data for the specific case of time series data [26]. The most studied combination-based models for this objective are Ensemble models. These models are used to combine the predictions of simpler base models to achieve better predictions. The advantages of ensemble learning include improvement of robustness and quality of the forecast and lower variance in predictor performance [68].

Polikar provided a non-time-series specific overview of the use of ensemble-based systems for decision-making, listing five reasons for the use of these techniques [52]. (1st) Because good performance on training data does not directly entail a good generalization performance, the use of multiple models reduces the risk of a non-representative predictor. (2nd) When there is too much data, dividing it between models for training and then combining their outputs for prediction is often a more efficient approach. (3rd) When there is too little data, training various models with different overlapping subsets of the total data, an approach named “bagging”, has been proven to be an effective technique. (4th) When the system or decision boundary is too complex to model through simple techniques, an aggregation of simple techniques can often represent the system better. (5th) Data Fusion is used when there are multiple heterogeneous data sources, with distinct sets of features requiring different models for each.

Recent proposals of ensemble models applied to TSF include an evolutionary model for combining Artificial Neural Network predictions by Zameer et al. [68], and an extremal optimization and Support Vector Regression Machine model combining a set of Long Short Term Memory Neural Networks predictions by Chen et al. [14], both used for wind-power-related forecasting.

2.2.3 Hierarchical Forecasting

Data hierarchies are often inherent to the problem domain when attempting to forecast a set of time series for real-world applications. As an example, a set of historical records of product sales in multiple stores could be aggregated by total store sales, SKU across stores, or product category. This data structure is referred to as Hierarchical time series [38].

Fliedner outlined the two classic approaches to achieve consistent Hierarchical time series forecasting [22]. When forecasting the individual time series variables and the derived hierarchically aggregate time series, consistency is guaranteed when the sum of the predictions of individual time series equals the direct aggregate prediction, and the disaggregation of the direct aggregate prediction by the relative weight of its component time series matches with the individual predictions. This is not guaranteed by general TSF techniques, although it can later be achieved through a reconciliation step.

There are various strategies for directly creating a fully consistent hierarchical structure by using "bottom-up", "top-down", and "middle-out" approaches or by eventually reaching consistency by using a reconciliation strategy for the different hierarchy levels. "Bottom-up" approaches generate the aggregate predictions solely from the sum of its lower-level components, while "top-down" disaggregates the higher hierarchical level predictions to achieve lower-level predictions.

Viswanathan et al. identified that the Bottom-up approach outperforms the Top-down approach only when the aggregation has few sub-component series and when the inter-order intervals have low variation [64]. This is of particular importance in the field of business-to-business sales forecasting, as there are often infrequent occurrences of non-zero values representing a "lumpy" demand profile at the lowest aggregation level. "Middle-out" approaches combine "bottom-up" and "top-down" approaches, forecasting at a middle level of the hierarchy, aggregating for higher levels, and disaggregating for lower levels.

Reconciliation methods place no restrictions under the base forecasting process, applied to all levels of aggregation. As an example, Hyndman et al. suggested in 2010 the use of a regression model to mutate the different level's predictions, achieving consistency [30]. The current state-of-art method in Hierarchical time series forecasting or grouped time series reconciliation, proposed in 2019 by Wickramasuriya et al., is trace minimization [64]. The authors showed that this approach generates forecasts at least as good as the base ones and should be preferred to bottom-up or top-down approaches, as these only make use of a limited amount of the available information.

2.2.4 Global Models

In TSF, a global model is one trained in multiple time series in order to generalize the learning process for the studied domain. This approach avoids the computationally expensive process of selecting and parametrizing a model for each of the target time series. While most of the TSF research to date has considered each time series as an individual data-set for training and testing, these models work under the premise that the multiple series with which it is trained are "related" in some way [28].

Hewamalage et al. compared Recurrent Neural Networks (RNN), Feed-Forward Neural Networks, Pooled Regression, and Light GBM (LGBM) as global forecasting models, demonstrating that the models with more complex modelling capabilities such as RNNs and LGBMs are good candidates for difficult prediction scenarios, such as data-sets containing heterogeneous or short time series [28].

Montero-Manso and Hyndman empirically showed that global and local forecasting methods are equally general [45], supporting the recent widespread success of global methods in comparative competition scenarios [40][41]. Additionally, the simplification achieved from having a single model to train allows for the application of increasingly complex models. This is of particular relevance as complex machine learning models such as deep networks are further researched outside of the specific field of TSF.

2.3 Microservices-based Time-Series Analysis

Finally, after we identified various synergies between the problem domain of time series analysis and MSAs, a literature analysis was conducted in the cross-section of these terms. The concise lecture notes of Uzun et al. [63] stand as the most relevant example of a design proposal for a MSA time series forecasting application.

Bellow, the proposed design includes four distinct microservices: A Dataset-collection service; a Dataset-management service; a Dataset-processing service; a Model-management service, and a Data-analysis service.

The objective of this work was to develop an architectural design for expandable and maintainable systems for TSA and TSF. The authors opted to redesign the system in a microservices-based architecture, as the monolithic solution proved to be difficult to evolve for the inclusion of new TSA techniques.

The communication between services is invocation-based, mediated through REST API Endpoints exposed by each service while keeping a choreography-based approach. While the authors show a clear improvement in development effort when employing the proposed system in comparison to Amazon Forecast Service and Manual Forecasting, they failed to provide evidence on the ease of evolution of the system when new desired functionalities do not fit in one of the described services. We note the partial adhesion to the outlined microservices tenets [70] and strong coupling between various services and the "Dataset-Management Service" in particular.

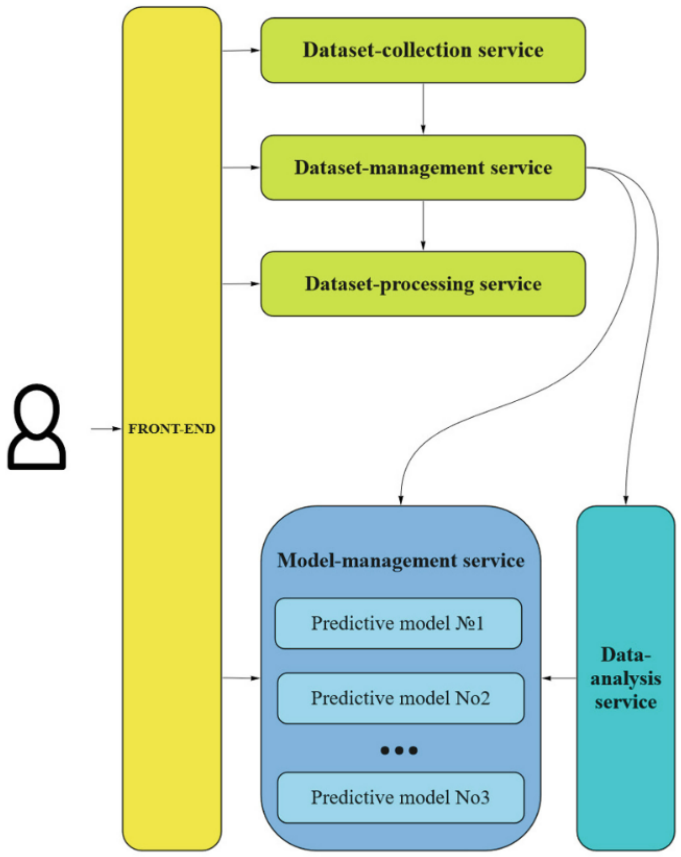

Figure 2.1: Architectural Model for Development of Time-Series Forecasting as a Service Application (adapted from Uzun et al. [63])

Chapter 3

Problem Statement

Within the context of analytics consultancy, designing software solutions that stand independent of the business context of the clients is a differentiating factor. Driven by the "boutique" nature of LTPlabs' approach, the current ad hoc development approaches, particularly in the scope of forecasting consultancy projects, extends the time-to-delivery of analytics solutions.

While delivering tailored solutions to specific clients is a crucial feature of the value proposition of a project, the adoption of standardised approaches and reusability of components previously developed for similar analytical processes is a strategic technological objective of the company.

The internal process most targeted by this standardisation approach is the pre-project diagnostic phase. During a diagnostic phase, LTPlabs receives data from potential clients' operations and delivers improvement suggestions within a two-week time window. The heavier time constraints of this phase, when compared to a possible subsequent project, are the main driving forces behind the development of standardised solutions by the in-house Data Science team.

In the scope of time series forecasting applications, mostly applied to sales and production data, the current diagnostic solution reflects a set of business capabilities focusing on defining the best models and model parameters for subsets of a hierarchical and grouped time-series dataset.

In the following section 3.1, we will explore in detail this solution, as it provides the starting point for iterative service decomposition. The decomposition effort of this monolithic application guided by the most up-to-date academic publications on the field of microservices architectures and the comparative analysis with the current approach through the assessment framework [7] proposed by Auer et al. are the main contributions of this work.

This decomposition effort is additionally driven by LTPlabs' long-term technological goal to move to a microservices-based development paradigm. Because of this, we will aim for the definition of domain-agnostic templates, processes and services for applicability outside forecasting workflows. The future reusability of the microservices extracted from the time-series forecasting diagnostic solution for subsequent forecasting project deployments is also a priority, as we aim to save the models trained during the first workflow for prediction generation.

Also discussed in this section are the identified preconditions for monolith to microservices migration, as this adoption is supported not only by the preexisting solution but also by already in-use cloud infrastructure, development operations practices, and validation data.

3.1 Current approach

The current solution for empowering the typical diagnostic process of time-series analysis within LTPlabs' projects is the "Time-series forecasting module" (TSFM). This module can be defined by a sequential set of operations applied to an input hierarchical and grouped time-series dataset, following the Cross-Industry Standard Process for Data Mining (CRISP-DM) [67] methodology. These hierarchies are introduced as identifier parameters for each time series and, as an example, often represent SKU categories or individual retail locations.

The complete workflow of this solution is illustrated in Figure 3.1 as a Dataflow Diagram (DFD) [15].

Loading and analysis

In an initial phase, this solution consumes the information of the whole dataset to extract cross-time-series insights from it. The TSFM creates from the input data an entity we will refer to as a Grouped Dataset. This entity contains not only the most disaggregated time seriesin what will be henceforth named the "bottom" dataset but also multiple aggregations of these individual time series, informed by the identifier parameters. These aggregations always include the "top" dataset, containing one single time series aggregating the values of all "bottom" time series.

Additional datasets represent partial aggregations by the several identifiers and their intersection in the case of non-hierarchical parameters. An example by Hyndman and Athanasopoulos [31], where Figure 3.2 represents aggregations respecting the hierarchical constraints, and Figure 3.3 represents non-strictly-hierarchical aggregations, as the bottom time series can be aggregated by the X and Y or A and B groupings, separately.

Following this aggregation during the instantiation of the Grouped Dataset, the extracted insights from the entire data include classifying each time series by the ABC, XYZ [12], and SEIL classification of forecasting potential [10] categories. This classification can be applied in a rolling manner to the individual time series, producing insights into the evolution of the categories over the historical reporting period. Outlier detection is also performed at this stage.

Univariate Forecasting

After this step, the TSFM performs the following stage of the CRSIP-DM methodology. It applies the modelling phase to the individual time series across all datasets. The modelling phase comprises the preparation of the train and test sets and definition of a set of backtesting time windows referred to as folds, and the selection of an algorithm and its hyper-parameters. In the case of the TSFM, the first part is done at a Grouped Dataset level. Folds and the train-test split are defined in the initial parameterisation of the Grouped Dataset instance, as model fitting should

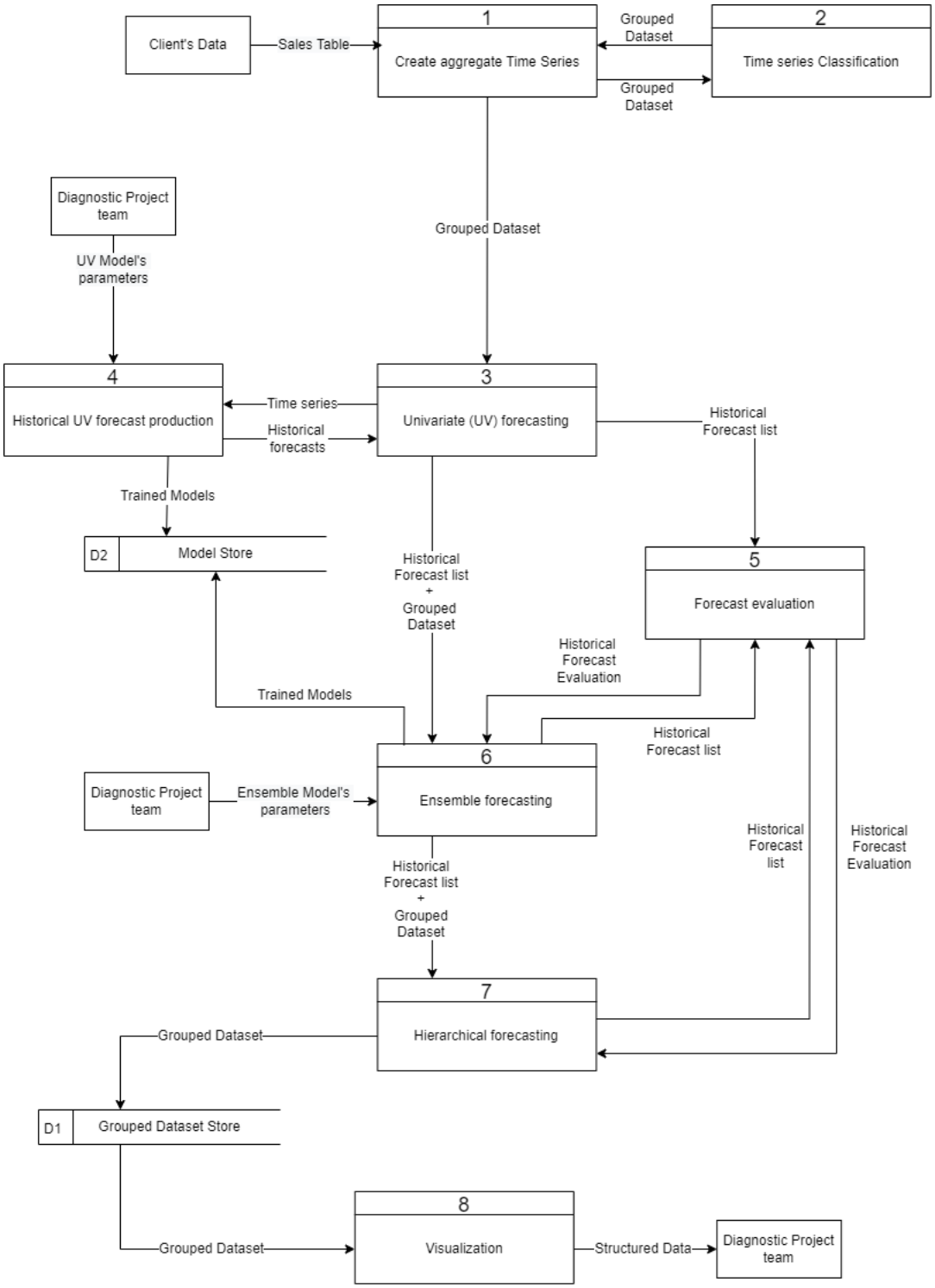

Figure 3.1: Current solution's Dataflow diagram

be done with the similarly time-located folds for every time series. This restriction derives from the later need to evaluate the models per fold across time series.

The following steps, algorithm and parameter selection, are made in a sequential fashion by

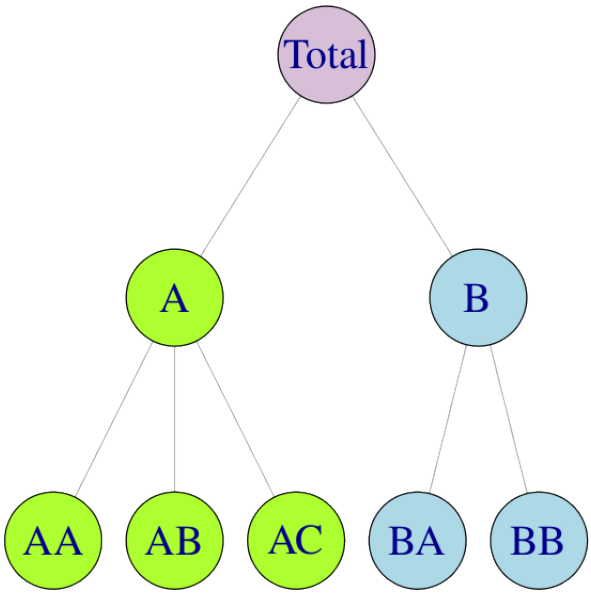

Figure 3.2: Hierarchical aggregation of time series (adapted from Hyndman and Athanasopoulos [31])

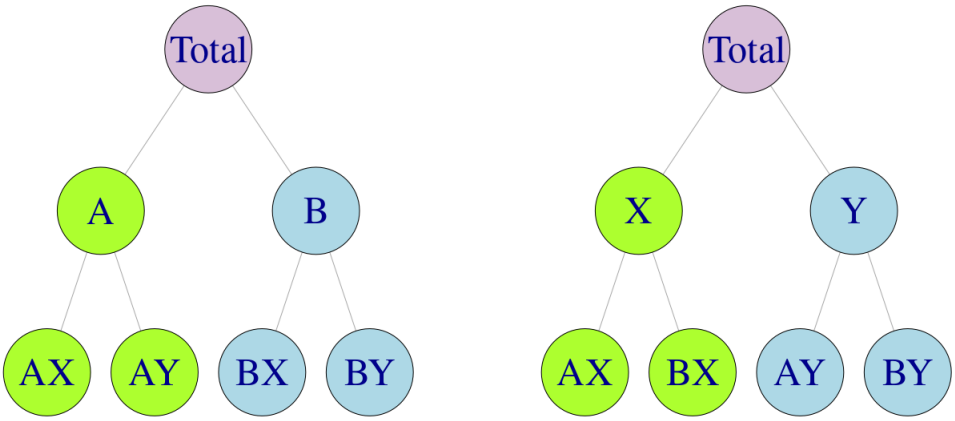

Figure 3.3: Grouped and hierarchical aggregation of time series (adapted from Hyndman and Athanasopoulos [31])

the TSFM, in consecutive independent function calls for each time series. A set of univariate time series forecasting models is parameterisable, including simple Naive models, statistical models including ARIMA and Exponential smoothing and Croston's method, and complex machine-learning models such as Neural network autoregression. All the models are fitted and applied once for each fold, producing a set of overlapping historical forecasts for backtesting purposes.

This phase has the most impact on the module's execution time, as the processing time linearly increases with the size of the Grouped Dataset. This is evident when analysing Figure 3.4, representing the execution time of each step of the process for a set of benchmark inputs. An intermediate evaluation step is performed for all produced historical forecasts, as is needed for the following business capabilities reflected by the TSFM, ensemble model application and hierarchical forecasting.

Ensemble Forecasting

Ensemble models, as discussed in section 2.2.2, join the predictions created by the univariate models, either by combining them linearly through a weighted average of contributions based on their evaluations or by combining them in a non-linear fashion by the application of machine learning models such as GBM. The fitting of the machine-learning ensemble models is done with all the historical forecasts of test folds of a subset of the time series. This subset can be defined by the aforementioned classifications (i.e. ABC, XYZ, SEIL) or by the inputted identifier parameters. The ensemble models are fitted per parameterised aggregation, targeting each hierarchical aggregation individually.

Hierarchical Forecasting

After a new evaluation step for the newly created ensemble models, a set of hierarchical forecasts is created for the entirety of the grouped dataset through "top-down" prediction deaggregation and "bottom-up" prediction aggregation methods as discussed in section 2.2.3.

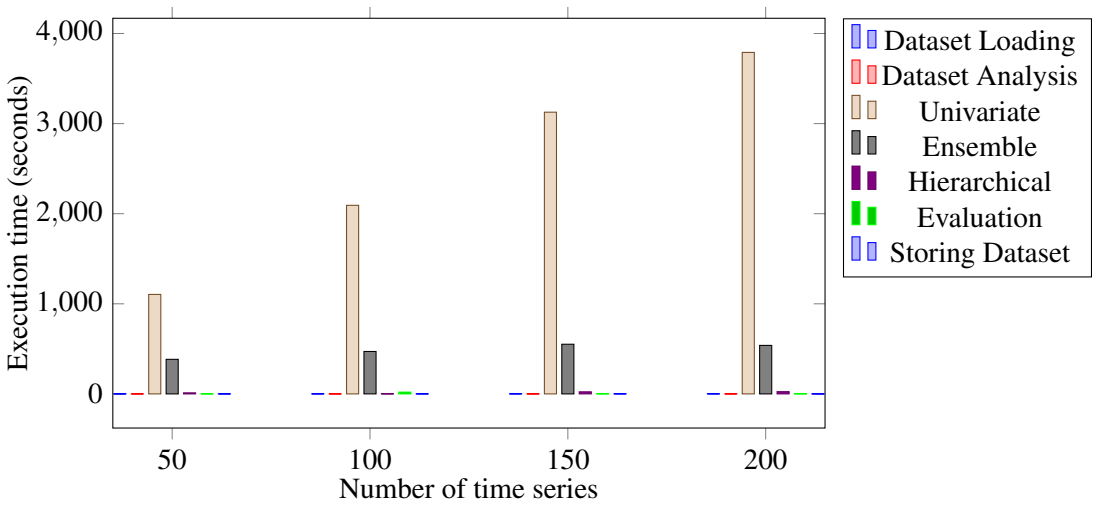

Figure 3.4: Pre-decomposition execution time of different components

3.2 Requirements

In this section we present the requirements of the decomposed system, including the pre-decomposition functional requirements and the microservices-specific objectives.

The main motivations behind the migration towards a microservices-based architecture span three main areas: reusability of service functionality for multiple workflows (section 3.2.1), independent scalability of the extracted services to lower execution time and allow for larger dataset processing (section 3.2.2), and integration of the migration in a wider organisational shift towards a data-driven microservices architecture paradigm (section 3.2.3).

3.2.1 Main workflows

The functional requirements of the proposed design are the eventual support for the two already mentioned workflows, diagnostic and deployment, representing the main high-level functionalities encompassed in LTPlabs' time series forecasting bounded context.

While the decomposition target will be solely the diagnostic solution, we aim to allow for the reusability of the defined services for the deployment workflow, as they share functionality and access the same subset of business entities.

Diagnostic

The diagnostic workflow follows the structure employed by the solution previously explored, followed by a manual model selection step done by the project team allocated to a client. This human choice step aims to interpret the forecasting results within the framework of constraints and goals of the specific client and project. In summary, the workflow counts with the following steps:

1. Manual Parametrisation
2. Aggregate time series datasets creation
3. Grouped Dataset level analysis
4. Univariate forecasting (for each time series)
    (a) Historical forecast creation (for each model)
        i. Fitting (for each backtesting window)
        ii. Predicting (for each backtesting window)
    (b) Save trained models
5. Univariate forecast evaluation
6. Ensemble model forecasting (for each time-series subset)
    (a) Historical forecast creation
        i. Fitting (for each backtesting window)
        ii. Predicting (for each backtesting window)
    (b) Save trained models
7. Ensemble forecast evaluation
8. Hierarchical forecasting
    (a) Top-down
    (b) Bottom-up
9. Hierarchical forecast evaluation
10. Forecast and evaluation visualisation

11. Informed manual model selection

Deployment

The deployment workflow aims to generate up-to-date forecasts for a client's Enterprise resource planning software or a stand-alone dashboard. To that effect, regularly scheduled diagnostic runs are performed to keep the models updated and for manual model re-selection when needed. From these diagnostic runs, the saved fitted models are used to generate the presented predictions. The generation of the predictions follows the structure below.

1. Parametrisation from the diagnostic's output
2. Aggregate time series dataset update
3. Univariate model forecasting (for each time series)
4. Ensemble model forecasting (for each time series)
5. Hierarchical forecasting
6. Forecast visualisation

3.2.2 Scalability

As demonstrated by the execution metrics of the current diagnostic solution detailed in section 3.1, its applicability within the time constraints of the analytics consulting diagnostic process is limited by the size of the dataset, as shown by Figure 3.4.

By guaranteeing independent scalability of the highly parallelisable univariate prediction step, where the generation of predictions for each time series is independent of each other, the migration to a MSA promises to reduce execution time by allowing for more allocation of cloud resources, supported by an event-driven, asynchronous communication structure, load balancing, and autoscaling.

Additionally, independent scalability of the ensemble model forecasting step would allow for parallel fitting of these models to different subsets of the Grouped Dataset time series. For this purpose, these requirements should inform the decisions taken when defining service boundaries.

3.2.3 Organisational requirements

The shift to a MSA paradigm goes beyond the bounded context of time-series forecasting within LTPlabs. For certain design decisions, the wider applicability to other analytical workflow processes of the designed components is desirable, as its re-utilisation within the organisation justifies the increased development costs and complexity associated with distributed systems and will eventually drive more development efforts on said components, leading to a more robust solution.

Setting development standards for microservice creation and integration at a larger organisational level is a key motivation of the proposed design.

In section 4.2, we further discuss design choices, including the creation of a common microservice template, the standardisation of event creation through versioned schemas adhering to CloudEvents specifications [3], the integration with preexisting infrastructure mentioned in section

3.3, and the creation of a software development kit for service integration with an asynchronous message queue.

Finally, the CQRS-based Command and Query services aim to be a generic interface for saving state changes and rebuilding business entities through interaction with an event-store database.

3.3 Supporting factors

Several preexisting technologies and practices within the organisation allow for the iterative migration to a MSA. These were indicative of the technological maturity of LTPlabs and indispensable for supporting the development of the solution proposed.

Furthermore, leveraging several clients' anonymised sales data and the heterogeneous nature of these datasets facilitates the testing and validation of the designed system and allows for the creation of benchmarks to draw comparisons between the TSFM and the proposed design.

3.3.1 Preexisting infrastructure

By applying the microservice tenets outlined by Zimmermann [70], discussed in section 2.1, we can identify preexisting conditions within the technological infrastructure of LTPlabs that contribute to the success of this migration.

For tenet six, practising continuous delivery and decentralisation during service deployment, the company's relationship with the clients already entails this practice, supported by already used technologies such as Kubernetes, Helm, and a mature AWS-based cloud infrastructure. By employing container registry through AWS' ECR (Elastic Container Registry) and deployment via AWS' EKS (Elastic Kubernetes Service), the deployment of the built microservices and the supporting infrastructure is facilitated and made independent. Additional deployments facilitated by this cloud-native infrastructure include an asynchronous message queue for inter-service communication (RabbitMQ), a file storage system for the Grouped Dataset entity (AWS' S3), and a key-value store database for event sourcing (AWS' DynamoDB).

For tenet seven, employing light and automated approaches for holistic management, we leveraged LTPlabs' GitLab pipelines for automating service redeployment and various observability platforms such as Rancher for Kubernetes cluster management, Prometheus for microservice monitoring and Grafana for aggregating service metrics, to manage the added complexity from distributed nature of the system.

3.3.2 Validation Data

LTPlabs clients for which time series forecasting projects are currently deployed are of different industries and scales. Diagnostic time series forecasting workflows were conducted for an even wider selection of clients.

We will utilise two of these datasets, anonymised, in order to validate the proposed design with time-series data of varying sizes, both in time span and in the number of time series. The

datasets also have different hierarchical structures, with a varying number of time series identifier parameters and, therefore, a varying number of aggregations.

3.4 Proposed solution

As mentioned in section 1.2, our design proposal will consist of an event-based TSA system, following the principles of the microservices architectural pattern, allowing for simple iterative addition of functionality through service creation. This system must sustain both occasional model fitting and regular parametrised predictive workflows.

These services should partially be extracted from the current monolithic solution used for TSF model fitting, evaluation, and selection. The extraction and definition of these services will follow a Dataflow-Driven approach as defined by Chen et al. [15]. The literature in service extraction when refactoring monolithic systems is explored in section 2.1.1.

Our design will make use of the CQRS/Event Sourcing pattern for data management across services. To achieve CQRS, we will isolate the interaction with the chosen data storage solution through a pair of mediator services for commands and queries separately. Applying Event Sourcing implies the definition of a set of business entities comprising an ordered append-only list of events subscribed by the CQRS command service.

The state of these business entities, relevant to the workflows used for validation, will then be derived from replaying the listed events by the query service. We focus on the industry-adopted responses to the data management complexity in MSAs in section 2.1.2.

The proposed system should follow the more decoupled approach of event-based inter-service communication while still defining REST endpoints mapped to event types: this approach guarantees message and documentation standardisation and possible invocation-based interactions when needed. Additionally, we aim to apply the choreography pattern in opposition to orchestration to further decouple the services. These alternatives and their characteristics are analysed in section 2.1.3.

Finally, we will validate the design by implementing it and running similarly parametrised diagnostic workflows with real-world client data in both the original TSFM and a prototype of the the proposed decomposed MSA design. The outcomes of these comparisons will be interpreted in light of the microservices assessment framework proposed by Auer et al., based on the ISO/IEC 25010 quality model [7].

Chapter 4

Design and Implementation

In this chapter, we will discuss the approach applied for service extraction, the design choices guided by the broader organisational shift towards microservices architectures, the events published by each service, and the mapping of the diagnostic workflow to ensure the fulfilment of the functional requirements.

4.1 Service Extraction

This section details the service decomposition process, guided initially by the "Dataflow-Driven" approach proposed by Chen et al. [15] to identify fine-grained candidate microservices. We define a set of coarser-grained services better aligned with the business logic by aggregating and generalising them from the identified candidates. Based on the requirements detailed in section 3.2, the outcome list of services will be ordered by migration priority to inform the iterative migration process.

4.1.1 Dataflow-Driven Decomposition

Employing the approach further discussed in section 2.1.1, we applied two transformations to the DFD representing the original design in Figure 3.1.

The first transformation step creates a "Purified DFD", a more data-focused representation of the workflow from the DFD. For this, data stores and external entities were first excluded, and operations and data entities were gradually refined to conform to the defined rules 1 and 2 [15, Section III, A. From Traditional DFD to Purified DFD]:

> **Rule 1.** (...) operations need to be detailed enough to represent all the individual data processing activities in the original business logic, while the data representation related to an operation should keep the semantic granularities of input and output data without further splits.

> **Rule 2.** (...) operations should be normative verbs or verb phrases that can reflect the semantic meaning of the corresponding data processing activities, while data should be named using semantically meaningful nouns or noun phrases that occurred in the original business logic.

The application of this process resulted in the diagram in Figure 4.1. As an example, the "Evaluate" operation was derived from process five in Figure 3.1, and replicated for each upstream operation to simplify its representation.

The second transformation step was applied to the Purified DFD to create a "Decomposable DFD", a decomposition-friendly dataflow representation. For this, we applied the following rules cited from the original work [15, Section III, B. - From Purified DFD to Decomposable DFD]:

> **Rule 3.** (...) each operation node needs to be adjusted to have one output data only.
>
> **Rule 4.** (...) each data node needs to be adjusted to have one type of precedent operation at most.
>
> **Rule 5.** (...) after applying Rule 3 and 4, the same operations with the same type of output data need to be combined into one operation with its output data. Since it is usually difficult to keep the same semantic meaning of the output data after combination, it would be needed to name the combined output data with an abstract semantic concept.

From the resulting Decomposable diagram, the candidate microservices are extracted by identifying the operation-output-data modules. The Decomposable diagram, as well as the numbered microservices candidates, are included in Figure 4.2. An example of the application of the rule three was the separation of the "Forecast Ensemble" operation in two operations, "Fit Ensemble" and "Ensemble Predict".

The candidate microservices labelled green (i.e. 10, 11, 12) are identified as possibly generalisable for uses outside this project's scope within the organisation and will be further discussed in section 4.1.2. The resulting service candidates exhibit a fine granularity and will be further aggregated into the final services list in the following section.

4.1.2 Candidate microservices

We identify a set of "ready-for-extraction" candidates within the extracted candidates. The *Save Model* and *Join Predictions* candidates (10 and 11) can be generalised as *Command* and *Query* services, following the CQRS pattern [47].

Model saving will therefore represent a state change in the system. We apply a similar principle to the publication of backtesting predictions, saved as append-only state-changing events by the *Command Service*. The Historical univariate prediction list will then act as a Business Entity created at query time by a generic Query Service. Joining Predictions and publishing the final business entity will therefore be its responsibility.

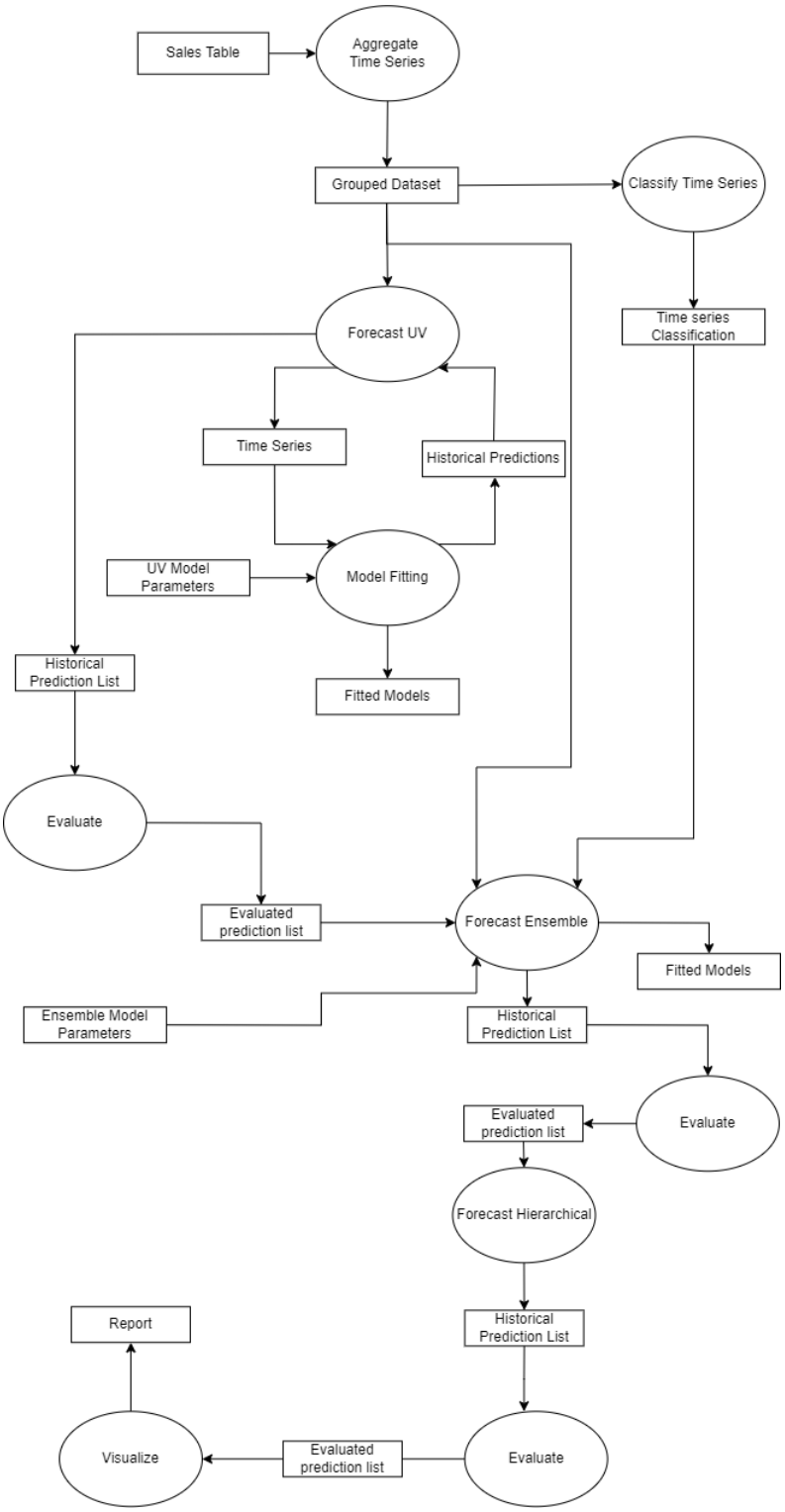

Figure 4.1: Purified Dataflow graph

By identifying the Grouped Dataset as a common business entity in the organisation, we can also consider the visualisation service as independent from the target system. The ability for consulting project teams to visualise the predicted test folds, the backtesting folds predictions,

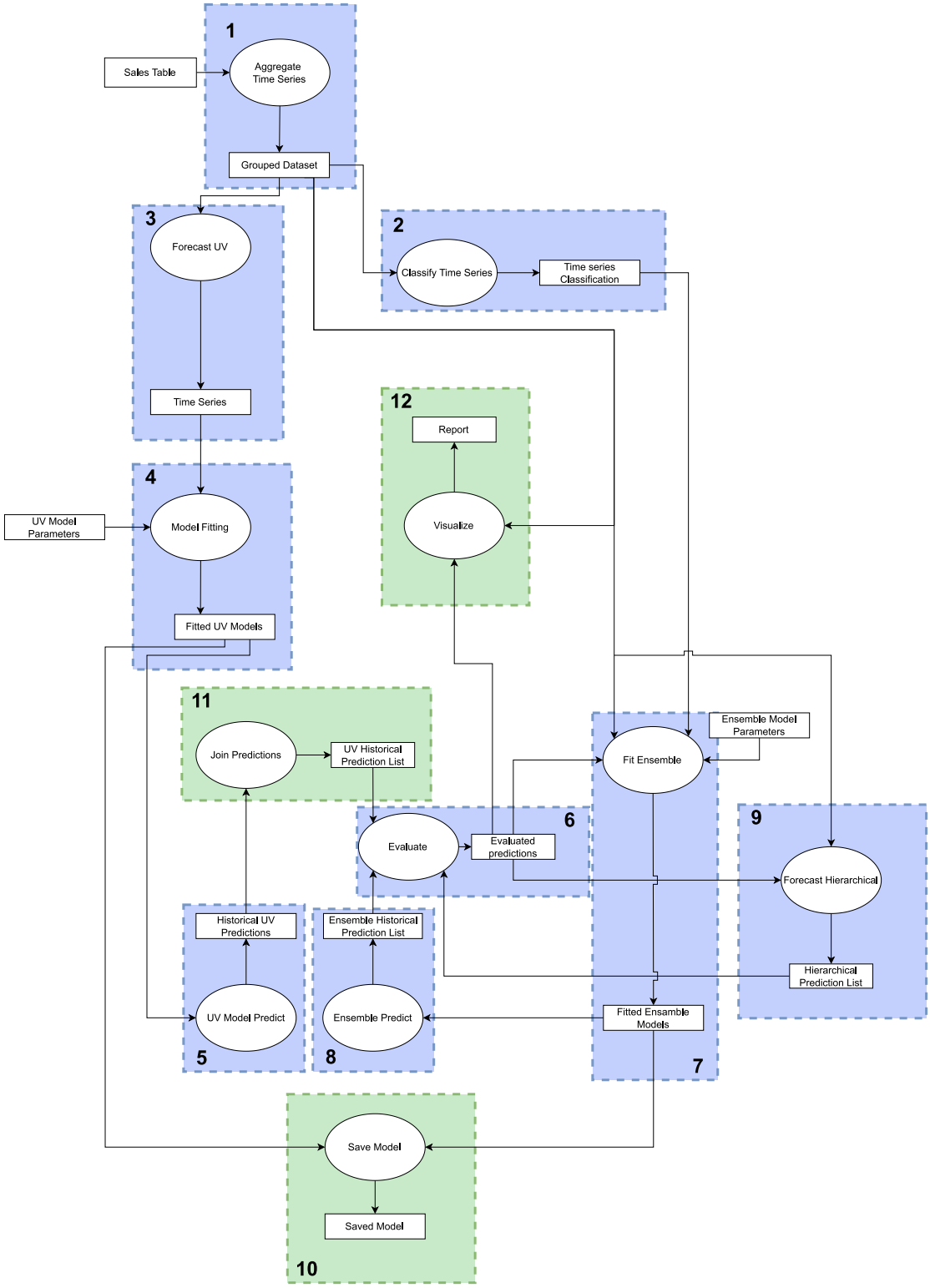

Figure 4.2: Candidate microservices

and the error metrics associated with them map a business capability that extends beyond the current implementation, able to contribute directly to ongoing forecasting projects. Therefore, this

visualisation will be extracted as a service we named *Forecasting Dashboard*.

For the migration's high parallelisation goals, we can identify the ability to consume individual time series in opposition to the current sequential approach as a necessary outcome. As such, we will define as a service boundary the dataflow between candidates 3 and 4. We also chose to aggregate candidates 4 and 5 into a single *Univariate Prediction service* to reduce the interdependency of these components, as they would both need to maintain internal representations of the models and perform sequential calls to each other to generate multiple instances of fitted models and forecasts for each of the parametrised backtesting windows. This option avoids the Cyclic dependency antipattern described by Taibi et al. [62].

Applying the same principles to the generation of ensemble forecasts, we will aggregate the candidate services 7 and 8 into an *Ensemble Prediction service*, responsible for managing ensemble forecasting models, named *Ensemble Prediction service*. Candidate service 9 generates the aggregated and disaggregated hierarchical predictions at a grouped dataset level without saving fitted models. As such, its model representations are already exclusive to it. We will name this service *Hierarchical Prediction service*.

Candidates numbered 1, 2 and 3 all perform trivial functions to the initially loaded sales information, creating, analysing and disaggregating the individual time series to be forecasted, respectively. We will aggregate them to reduce complexity and the aggregated service will be called the *TS Load service*.

Finally, the candidate service 6, representing the functionality of evaluating the generated backtesting and test forecasts, will be included in the candidates downstream in the dataflow for simplification, as its scalability is tied with them. The three upstream components consuming evaluated predictions are the already defined *Ensemble Prediction service*, *Hierarchical Prediction service* and the *Forecasting Dashboard*. To reduce the number of services mirroring this component, as the visualisation is downstream from the prediction services, the Forecasting Dashboard does not need to include this capability.

4.1.3 Extracted services

The services identified in the previous section were the CQRS Command and Query services, the Time-series Load Service, the Univariate Prediction service, the Ensemble Prediction service, the Hierarchical Prediction service and the Forecasting Dashboard.

To order these services by migration priority, we first considered the impact of individual service extraction on the scalability requirements.

For extracting the Univariate Prediction service, including the set of functionalities scaling most in terms of run time (section 3.1), and validating its scalability-specific features such as autoscaling and the event-driven design, we found it necessary first to extract the *TS Load service*, generating individual time-series publication events to be consumed by the Univariate Prediction service.

After the implementation of these two services, for the further integration workflow, the Command service was created to save the state-changing events already generated by the *TS Load* and

Univariate Prediction services. The *Query service* was then implemented to recreate the aggregated time-series predictions for reintegration with the Grouped Dataset.

We opted for extracting the visualisation Forecasting Dashboard at this point for the validation of the generated predictions. The *Ensemble* and *Hierarchical Prediction* services are the final targets for migration in that order, as the ensemble model training and forecasting can be parallelised when distinct groupings of time-series backtesting predictions are used for training different instances.

The final list of services, ordered by migration priority, is as follows:

1. TS Load service
2. Univariate Prediction service
3. Command service
4. Query service
5. Forecasting Dashboard
6. Ensemble Prediction service
7. Hierarchical Prediction service

A simplified diagram of communication strategies between the services is included in Figure 4.3. The communication between the services will be further detailed in section 4.3.

4.2 Organisational impact

The objectives discussed in section 3.2.3 were already present in the decisions of the previous section 4.1.3, in particular when defining the extraction of the generic Command and Query services and the Forecasting Dashboard.

The organisational objectives in other aspects of the designed system were also considered when designing a standard Microservice template and defining the event schemas for inter-service collaboration and event versioning.

4.2.1 Service template

The template was created to accelerate service development, manage automatic deployment, service configuration, logging, Asynchronous Message Queue (AMQ) connection, event subscription, event to endpoint mapping, metrics publishing, and documentation of the service boundaries through FastAPI [53] endpoints utilising CloudEvent [3] schemas.

The service template was containerised through Docker for automatic deployment, and a pipeline for image updating and publishing to AWS' ECR was created. Additionally, AWS' EKS was used for container management, pulling the latest images automatically.

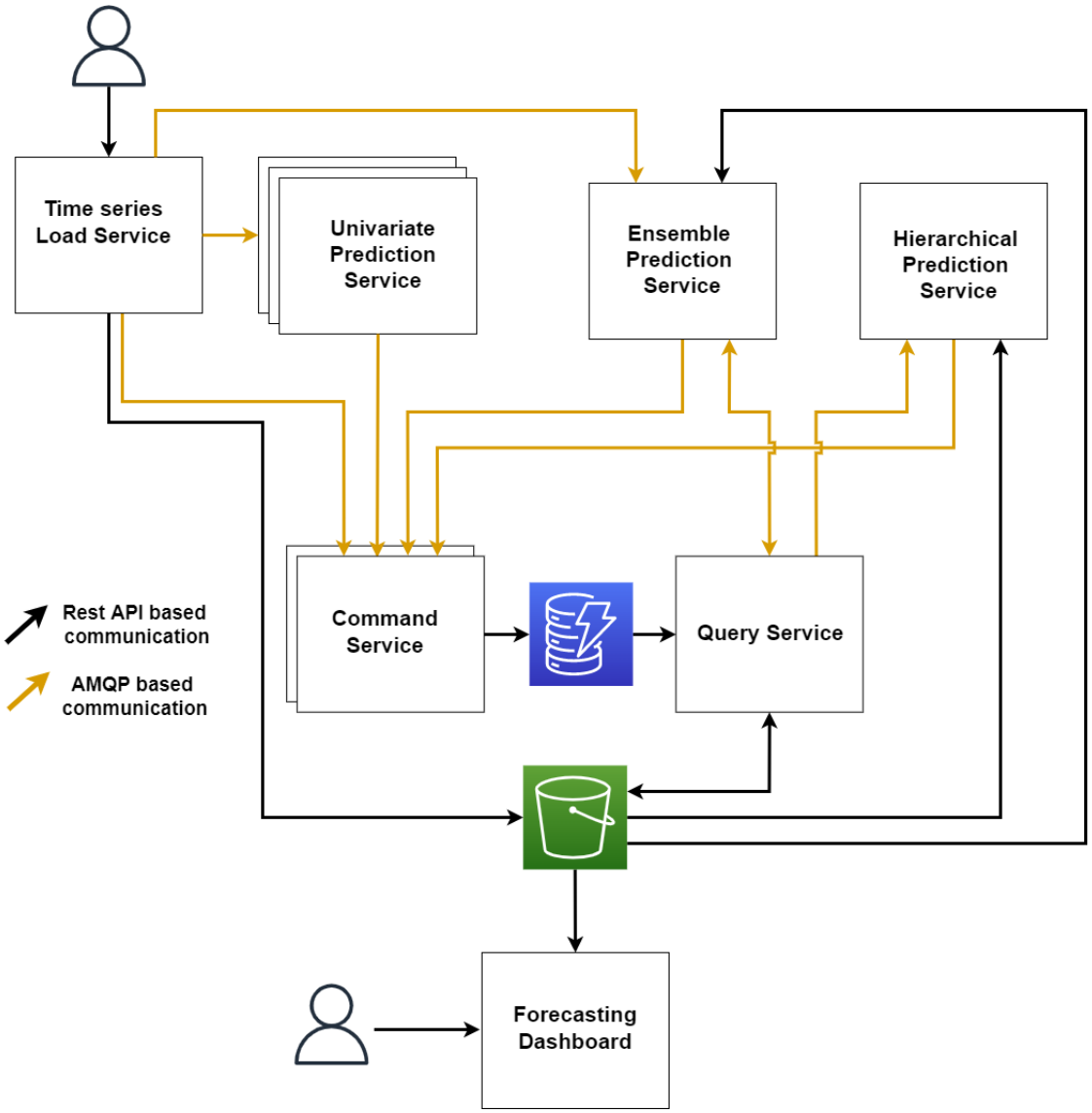

Figure 4.3: Service interaction diagram

For service configuration, Helm was used for creating a unified Configuration Map (configMap) [65] for each service, deployed by the AWS ECR to the images in the cluster. Logging is configured by the "Logging" file depicted in Figure 4.4 and can be managed easily across services by updating the configMap. This configMap does not, however, include the CloudEvent schemas further discussed in section 4.2.2.

An AMQ Protocol (AMQP) specific package used for connection with LTPlabs' RabbitMQ Cluster was also developed and included in the deployment workflow of the service template. This package remains separated from the service template as it can be updated for all services simultaneously if there is a need to migrate to another communication protocol. The package includes connection and channel pooling for threaded handling of multiple events and queue creation and subscriptions through the "RabbitMQ" configuration file, depicted in Figure 4.4.

When it receives a subscribed event, it issues a callback to a function parametrised by the service template. This function maps the event to an API endpoint based on its routing key and the "Endpoint Mapping" configuration file.

The REST endpoints implemented with FastAPI are metered by Prometheus and parse the information based on the generated CloudEvent classes. They can also be used directly, decoupling the services from the chosen communication solution, and acting as configurable input and output points for the overarching workflow, as is the case with the Time-Series Load service and the Forecasting Dashboard, which both expose REST endpoints to outside the system for interaction, for input and output respectively.

Finally, the documentation of the service boundaries is achieved through a combination of the CloudEvent schemas detailed in section 4.2.2 and the OpenAPI Specification used by FastAPI. This was done to avoid the pain of "Complexity of API management" identified by Zhang et al. [69].

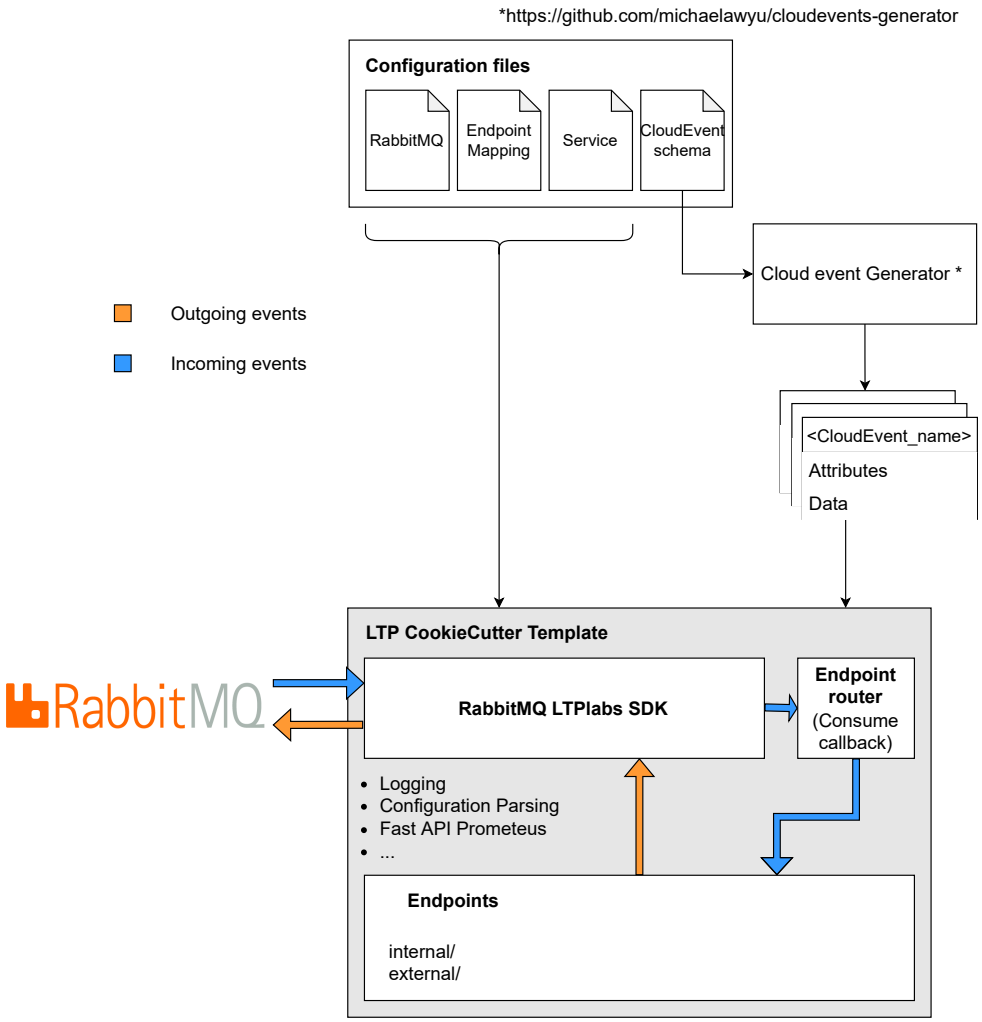

Figure 4.4: Service template

4.2.2 Event Schema

The Events to be created represent the publication of the data aggregations described by the Decomposed DFD. They are used to generate a package of versioned class files representing the events described in the schema. The generation of this package is done through a CloudEvent generator python application [44]. This package is also automatically pulled by the pipeline used for image updating.

In this schema, we can define an event attribute's type, default value, boundaries, description, and if it is required or not. This schema also guarantees the definition of the base attributes required by the CloudEvent specification (i.e. id, source, type and specversion),

Defining events in a standardised way within LTPlabs improves the ability to migrate other business capabilities to MSA outside of the scope of this implementation. This is also relevant for the Command and Query services as the event schema also acts as the schema of the aggregations stored in the DynamoDB event store.

4.3 Service integration

In this section, we will define the events published by each service based on the data aggregations present in the Decomposable DFD Figure 4.2.

TS Load service

Responsible for components 1, 2 and 3, this service publishes the Grouped Dataset, the individual time series and the time series classifications.

Each generated time series is published as a *NewTS* event. We aggregated the Grouped Dataset and the classifications into a single *NewDataset* event containing the classifications of each individual time-series referenced by ID and the reference to the Grouped Dataset object, stored in AWS' S3 and not shared by an event because of the AMQP's and DynamoDB's size constraints.

Univariate Prediction service

The Univariate Prediction service aggregates components 4 and 5, therefore being responsible for publishing the fitted models to be saved by the Command service and the backtesting predictions for the training folds. The models are mapped to a *NewModelUV* event, and the backtesting predictions are saved with the trained model, referencing it by id, to a *NewForecast* event.

Command service

The Command service captures events and saves them, not publishing any by consequence.

Query service

The query service publishes the lists of time-series backtesting forecasts for a subset of the time-series selected by a classification attributed by the Time-series load service and configured by each individual Ensemble Prediction service.

It first recreates the Grouped Dataset by replaying the events stored by the Command service, saving it again to AWS S3, and publishes an *EnsembleDataset* event referencing it and the name of the Ensemble. It also publishes in the same way a Grouped Dataset aggregating all Univariate and Ensemble predictions for the final hierarchical forecasting step. This constitutes the *HierarchicalDataset* event.

Forecasting Dashboard

The forecasting dashboard only consumes Grouped Dataset publications and visually displays the information to a user. As such, it does not publish any event.

Ensemble Prediction service

Similarly to the Univariate Prediction service, the Ensemble Prediction service aggregates the model fitting operations and the backtesting predictions generation. Because of this, it should publish the fitted models as *NewModelEns* events and the respective backtesting predictions as *NewForecastEns* events.

In addition, as the Ensemble Prediction service holds the configuration of the different subsets of time-series to train with based on the classifications attributed by the event, its responsible for issuing *EnsembleSet* events with the IDs of the time-series to train with to be aggregated by the Command service. This event is issued after the service consumes the *NewDataset* event, published by the TS Load service.

Hierarchical Prediction service

This service only publishes the generated and evaluated predictions by issuing evaluated Hierarchical forecasts for the test folds as *NewForecastHier* events.

4.4 Diagnostic Workflow Mapping

We start this section by introducing the sequence diagram pictured in Figure A.1 in Appendix A mapping the complete diagnostic workflow to the envisioned services.

The designed workflow was implemented in an iterative way, ordered by the priorities defined in section 4.3, with the state of the implementation at the point of the results discussed in this document being the disaggregation of all services except for the Ensemble and Hierarchical Prediction services, that remain aggregated in a single service, integrated with the rest of the infrastructure through the microservices template.

The Diagnostic Workflow mapping discussed in the following sections will be the implemented one and not the fully realised design. In Chapters 5 and 6, we will consider this limitation. It restricts the independent scalability of the Ensemble prediction service and, therefore, the training of ensemble models on different subsets of the forecasted time series in parallel.

With this in mind, when comparing the monolithic solution with the MSA-based prototype, we will train all ensemble models with all the individually forecasted time series.

4.4.1 Data disaggregation

The data deaggregation portion of the workflow for triggering the following parallelised prediction steps was the main contribution of this work to the responsibilities of the Time-Series Load service.

After the generation of the aggregate series from the input "bottom" series based on the hierarchical indexes, the service classifies all of the original and generated series. This classification is sent to the ensemble service in the *NewDataset* event. This event also propagates the configuration of desired models to the Ensemble Prediction service, while the individual *NewTimeseries* events propagate the configuration of desired Univariate Prediction models.

The *NewDataset* event references the IDs of the The *NewTimeseries* events and the stored Grouped Dataset, as this information would bring the size of the event over the recommended limit of RabbitMQ messages of 128 Megabytes and for large datasets over the theoretical maximum limit of 2 Gigabytes. With the *NewTimeSeries* events also containing the time series indexes, it is possible to reaggregate the Grouped Dataset from the information collected in the event store.

A key variable impacting the scalability at this workflow stage is the number of hierarchical indices, as they define the number of aggregations and, therefore, the total number of series to be forecasted. Of particular impact are the non-strictly hierarchical aggregations (e.g. purchases of a specific client), as they massively increase the number of time series forecasted.

This impact can be seen in Figure 4.5, where 2000, 4000 and 6000 individual bottom time-series were selected randomly from a test dataset. Note that the number of time series loaded to the bottom aggregation level is usually slightly lower than the number of selected time series as many series lack the minimum requirements for forecasting (e.g. all zero values).

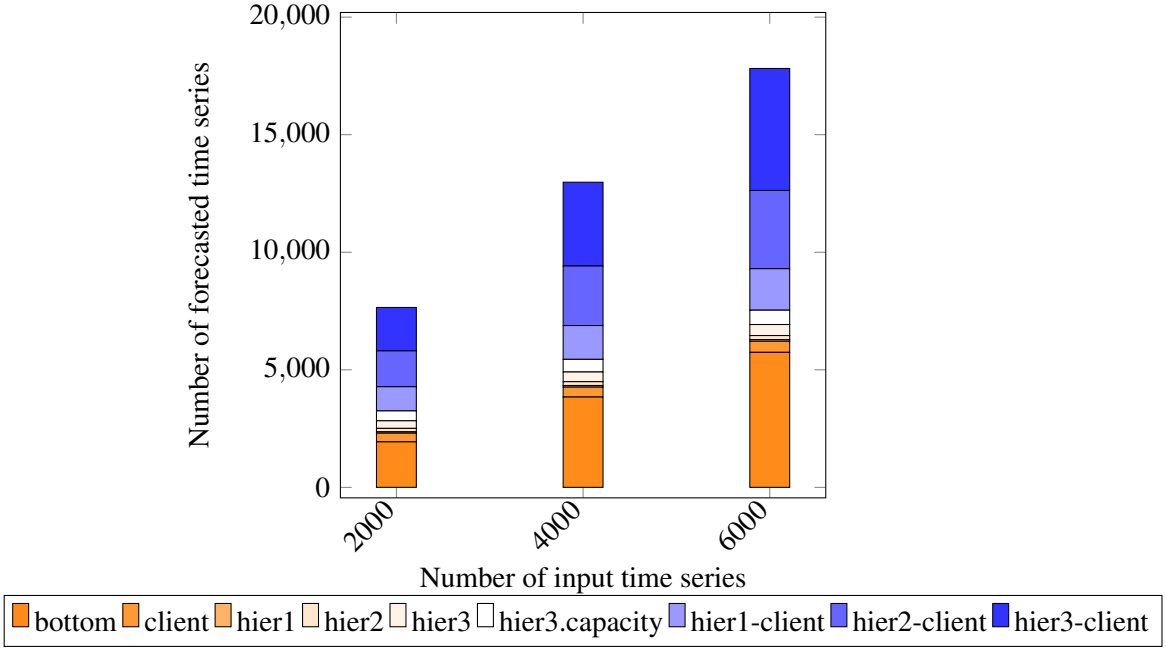

Figure 4.5: Number of time series per aggregation

Referenced in orange are all the original time series and aggregations generated by the strictly hierarchical indices, and in blue are those generated by the client grouping.

This increase in the run-time of the current solution is due to the scalability limitations of the Univariate Prediction models, discussed in the next section. In further comparisons, we will limit these aggregations to ease the current system's deployment. Still, we note this shows a clear need for independent scalability of the univariate prediction models for this use case.

4.4.2 Univariate Models

While the current solution allows for the parametrisation of multiple sets of hyperparameters for each parametrised type of model discussed in section 3.1, the prototype of the proposed design does not, as it attempts to streamline the process of performing hyperparameter tunning on a set of configured ranges and an error metric to be minimised, only saving the best model of each type.

At the point of result gathering, the model types parameterisable in the Univariate Prediction service were the NaiveDrift, NaiveSeasonal, NaiveMean, Exponential smoothing, and AutoArima, implemented through the DarTS [27]. These models differ from the monolithic approach, which uses the R programming CRAN models [29]. Naive Seasonal is only applied if a seasonal component is found within the individual time series. This choice was made to avoid the antipattern "Too many technologies" identified by D. Taibi et al. [62].

4.4.3 Data aggregation

The query service first receives a *NewDataset* event and a group of *EnsembleSet* events. These messages are requests for Grouped Dataset generation, as these services request information from the Query service. This request-response interaction is asynchronous, and any replica of the Ensemble Prediction service can receive the response, subscribed to by a shared queue, and initialise the fitting and backtesting of the ensemble models. This further reduces the interdependency of the Ensemble Prediction service and the Query service.

For rebuilding the hierarchical Dataset, there is a need to line up all backtesting windows of each model of all forecasted time series for ensemble model training. These windows can differ between series as different models have different minimum requirements for predictions. For example, autoARIMA needs a minimum of thirty observations to define the lag order and the degree of differencing hyperparameters.

The aggregation structure is maintained by saving the non-forecasted Grouped Dataset to S3 as a base for the reconstruction. This base is populated by the latest predicted backtesting windows saved in the DynamoDB event store by date.

Again, as the size of the Grouped Dataset far surpasses the theoretical maximum RabbitMQ message size, the reaggregated Grouped Dataset is stored in S3, and a reference to it is sent in the *EnsembleDataset* event.

4.4.4 Ensemble Models

All instances of the Ensemble Prediction service contain the parametrisation of the subgroups for Ensemble training. The hyperparameters of the models applied to all subgroups are parametrised in the TS Load service and sent in the *NewDataset* event, mirroring the behaviour of the Univariate Prediction service.

In response to the consumption of this event, the Ensemble Prediction service sends one *EnsembleSet* event for each subgrouping, containing the subgroup's name for later reference and the IDs of all *NewTimeseries* events of time-series included in the grouping. The name of the subgrouping is propagated by the Query service to the generated *EnsembleDataset* event, used to reference the models to be fitted when this event is consumed by an instance of the Ensemble Prediction service.

The aforementioned Grouped Dataset aggregated by the Query service is then used to train all parametrised models. The ideal number of Ensemble Prediction service replicas will therefore be the number of parametrised subgroups.

By allowing for the parametrisation of multiple cross-client subsets of time series and designing a reusable system for the use of multiple clients, this would allow for the inclusion of Global ensemble models after service extraction, discussed in section 2.2.4.

As discussed in section 4.4, the extraction of the Ensemble Prediction service was not yet complete at the point of result gathering.

The models parametrised in both the TSFM and the proposed solution for Ensemble Learning for the comparisons referenced in the results chapter 5 were a Generalised Linear Model (GLM), a Gradient Boosting Machine (GBM), both from the H2O library [34], and a simple linear model aggregating predictions by minimising the sum of absolute errors (SAE) (equation 4.1):

$$SAE = \sum_{t=1}^{n} |F_t - A_t| \tag{4.1}$$

4.4.5 Hierarchical Forecasting

The *NewDataset* event is additionally subscribed by the Query service, which generates a *HierarchicalDataset* event. This process is similar to the generation of the *EnsembleDataset* event. Using as a base the Grouped Dataset hierarchical structure stored in S3 by the TS Load service and triggered by the consumption of the *NewDataset* event, the Query service waits for all the time-series backtesting window forecasts of the univariate and ensemble models and reaggregates them, sending the new object's reference in the aforementioned *HierarchicalDataset* event.

This event is consumed by the Hierarchical Prediction service, triggering the generation of all possible top-down and bottom-up hierarchical forecasts referenced in section 2.2.3. The current prototype publishes the final Grouped Dataset at the end of this step after evaluating all forecasts, completing the analysis to be visualised in the forecasting dashboard.

Chapter 5

Evaluation

This chapter will detail the results gathered from testing the original solution and the proposed design. The prototype used for collecting the following metrics was at the sixth migration step of the list defined in section 4.1.3. As such, we will focus our analysis on the execution time results, forecasting error metrics and cost, not considering the ability to parallelise the training of multiple ensemble models for different subsets of time series.

We aim to show the value gained from a partial extraction and discuss the increased adaptability of the proposed solution to include new techniques in time series forecasting.

5.1 Experimental design

We compared the pre-decomposition solution running in AWS EKS with three allocated CPU cores and 7Gb of RAM, parametrised to use up to three threads for parallel computation, with an instance of the proposed design using three, six and twelve replicas of the Univariate Prediction Service. As with the Command Service, the Univariate Prediction service replicas are parametrised with one CPU core and just 700Mb of RAM. In contrast, the other services that handle the full Grouped Dataset are deployed with a limit of 7Gb of RAM. By this parametrisation, we hope to approximate the computing resource usage of the compared solutions.

Diagnostic workflows for two different datasets were executed for both solutions. For simplicity, all workflows were parametrised to generate predictions for ten backtesting windows, with a forecasting horizon of five for each of them.

To compare the prototype with the original approach, we parametrised the latter with the R CRAN models ARIMA, ETS (Error trend seasonality), and SES (exponential smoothing), two naive rolling mean models with different sizes of training windows, and a naive seasonal model, while the former includes Naive Drift, Naive Seasonal, Naive Mean, Exponential smoothing, and AutoARIMA, implemented through the DarTS [27] Python library.

To validate the design proposal, we used two different datasets. The first refers to a Portuguese Beverage producer's business-to-business sales data, and the second refers to an Electronics retailer's business-to-consumer sales data.

5.2 Results and Analysis

In this section we detail the overhead, execution time and forecasting results gathered from both systems for the validation datasets. We first used the dataset relating to business-to-business sales of products in the beverages category from an industry-leading Portuguese producer to highlight the overhead introduced by the decomposition. We used increasingly larger subsets of this dataset to compare the execution time of the current solution and the proposed design.

For execution time and forecast error metrics comparisons we also used the second dataset, comparing the results both between the two systems and the two datasets.

5.2.1 Overhead

Results

The following graphs in Figure 5.1 detail the execution time comparison of the different dataflow components for 100, 150, and 200 input time series.

We note that the impact of the Loading and analysis and Hierarchical Forecasting components on the total execution time of the diagnostic workflow is minimal. As we can see in Figure 5.1, the Univariate Forecasting and Ensemble Forecasting steps take multiple minutes to hours for completion while the other two components are executed in a matter of seconds.

Analysis

Through these results, we can see the overhead introduced by the distributed design across all execution components. The time taken to train the univariate models and generate the respective backtesting forecasts is increased by 55%, 35% and 65% in the 100, 150 and 200 time series benchmarks, respectively. The use of a different model library and the addition of hyperparameter tunning may have also played a role in this increase in execution time with a base configuration.

The overhead on the Loading and Analysis component can be attributed to the time to publish all individual time series messages. This overhead is minimal compared to the rest of the execution but represents a tenfold increase in the execution time of the component.

We expected similar execution times in the Ensemble and Hierarchical forecasting components as the prototype still counts with a single pre-decomposition module for both techniques. We attribute the decrease to the lower number of generated univariate forecasts and the non-deterministic nature of the previous forecasting techniques.

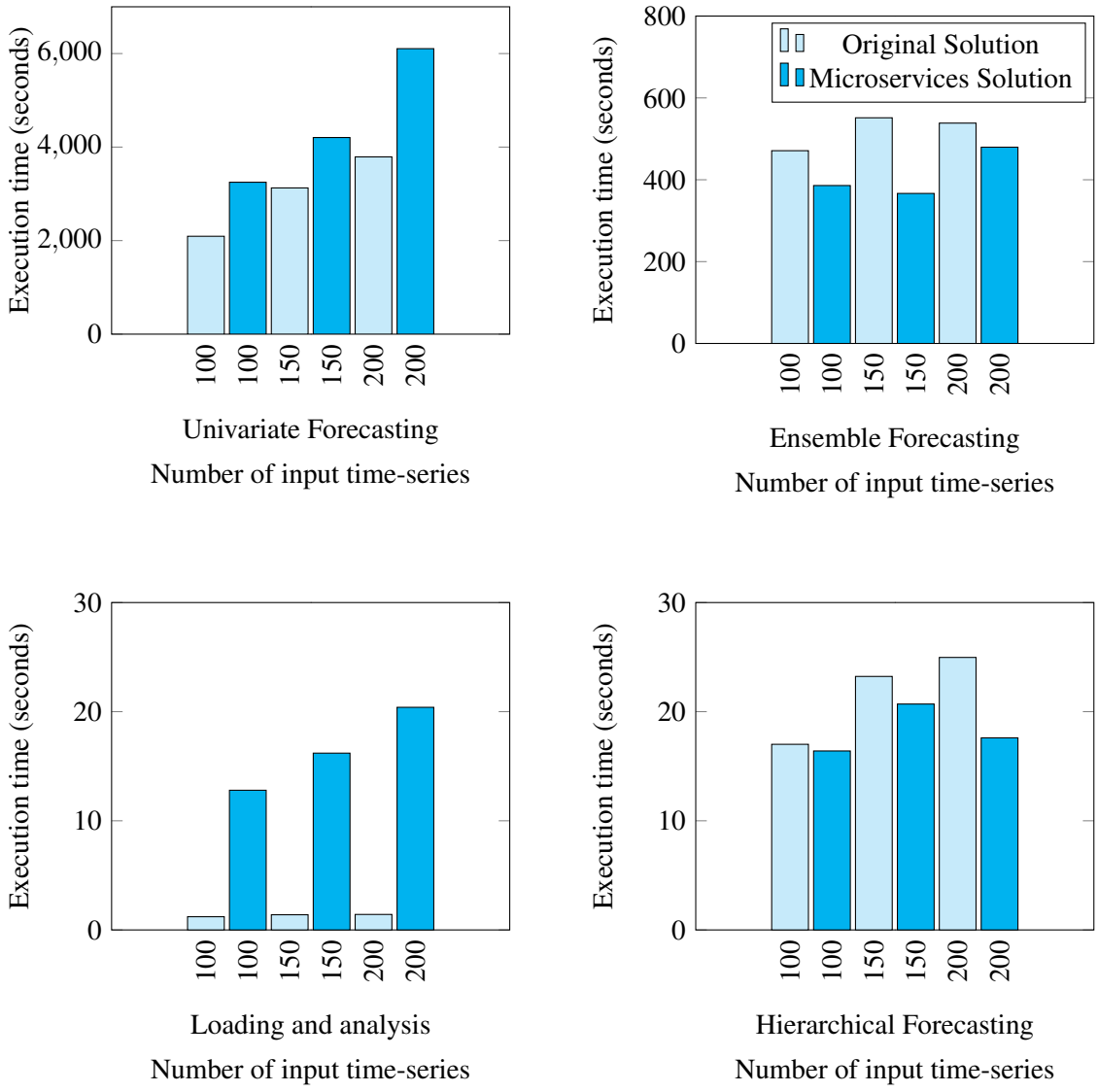

Figure 5.1: Execution time of components

5.2.2 Scalability

Results

As one of the systems' requirements is independent service scalability, we explored the execution of the prototype with a different number of replicas of the Univariate Prediction service to tackle the most time-consuming task. For this, we detail in the Figure 5.2 the progression of total queued messages in the asynchronous message queue. We note the addition of an indicator of the execution time of the pre-decomposition solution parametrised for parallelisation of model fitting with three threads, for a closer comparison with a base execution of the decomposed system with three single threaded instances of the Univariate Prediction service.

We then executed the workflow for 350 input time series with similar parametrisation for a different dataset, containing sales data of a client in the Electronics retail sector, with many more

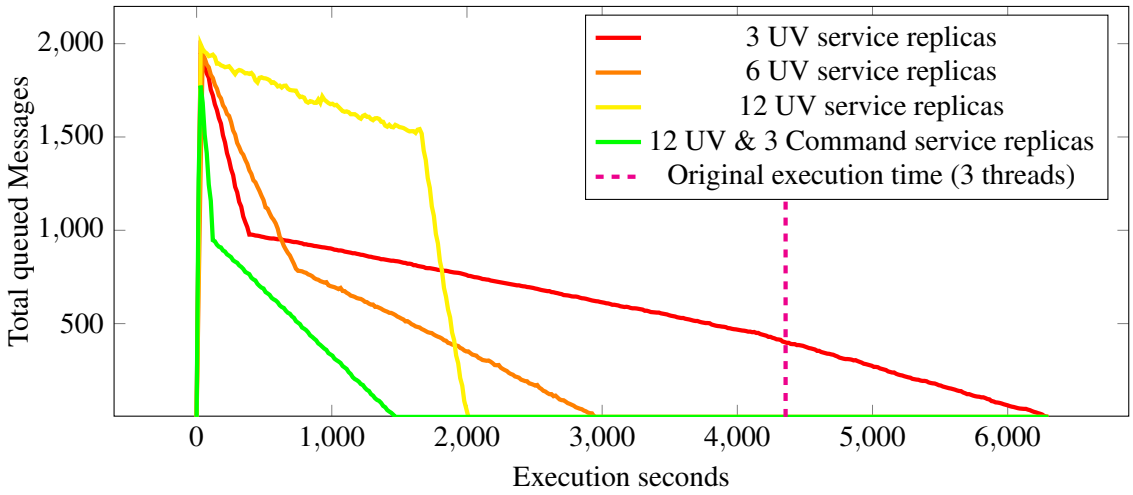

Figure 5.2: Queued messages, 200 input time series

aggregations but time series much smaller in length. Here, although many more "bottom" time series are produced, the length of the time series is not enough to fit the AutoARIMA model. The comparison of the executions of both datasets is detailed in Figure 5.3. All of the executions in this graph were done with six replicas of the Univariate Prediction service.

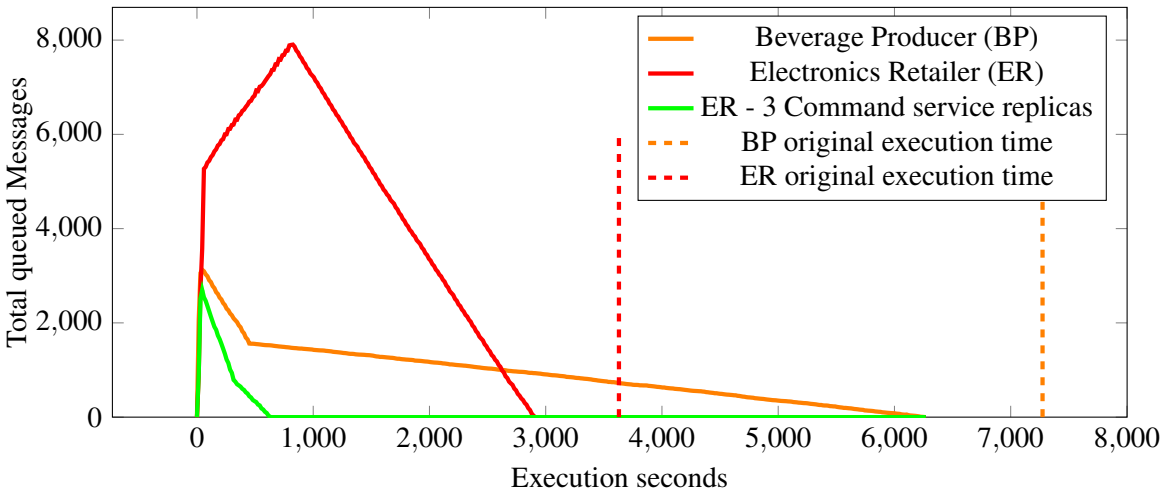

Figure 5.3: Queued messages, 350 input time series

Analysis

Analysing Figure 5.2, we can see the expected halving of execution time from three to six replicas, reducing the response time to below the current solution's benchmark and overcoming the overhead of the added complexity. In opposition, the redoubling to twelve instances of the Univariate Predictor did not reproduce this result.

The increased slope at the start of the three and six replica instances is due to the time-series messages consumed at the beginning of the workflow by the Command service. The Univariate

Prediction service also produces one forecast event per model per time series to be consumed by the Command Service.

The inverse slope change in the prototype's twelve instance version was caused by the Command service consuming fewer *NewForecastUV* events than the ones created. Once the Univariate Prediction service instances finish consuming all *NewTS* events, the Command service consumes the *NewForecast* and *NewModleUV* events left in its queue quickly.

The implemented architecture allows for the replication of the Consume service to overcome this new bottleneck. As detailed in the figure, by scaling the number of instances of the Command service to three replicas, we can overcome this bottleneck and reach the expected 50% decrease in execution time compared to the six instances of the Univariate Prediction service.

Analysing Figure 5.3 we can see that the Command service bottleneck already with only six Univariate Prediction service replicas. As such, we see again a need to scale the Command service.

This is due to the forecasts of the microservices architecture being produced much quicker for the second dataset, as its individual time series have less values than in the first dataset. We note that comparing the execution time with the original solution is only informative as the decomposed alternative generates fewer forecasts. With this, we show that the individual scalability of components offered by the microservices architecture is better at tackling varying workloads.

5.2.3 Forecasting Metrics

Results

We gathered error metrics for the multiple executed runs, and we drew comparisons of the ones generated in the executions detailed in Figure 5.3. These metrics are used to validate the design, as the differences of implementation of the univariate forecasting models difficults direct comparisons. Still, through the parametrisation detailed in section 5.1 we expected to reach comparable error metrics for the execution of both solutions.

From the many error metrics collected, we chose to analyse the Mean absolute percentage error (MAPE) (equation 5.1) and bias (equation 5.2), as these are the most often used decision metrics for demand planning and in LTPlabs' context of analytics consultancy.

$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right| \tag{5.1}$$

$$Bias = \frac{1}{n}\sum_{t=1}^{n}\frac{F_t - A_t}{A_t} \tag{5.2}$$

These metrics are the mean of the errors of the predictions done for all "bottom" dataset time series, with a gap of one between the last training observation's date and the target value's date, in the last three backtesting windows.

We present the results gathered from the individual Univariate models in tables 5.1 for the Beverage industry dataset and in table 5.3 for the Electronics retail sales data. The appendix B contains these results in full.

We further aggregated the Hierarchical and Ensemble Model error metrics into the table 5.2 for the Beverage industry dataset and table 5.4 for the Electronics retail dataset. In these tables, "Average Change" details the difference between the average error metrics of all trained models by the original solution and the MSA prototype. For the Bias metric, the average change details the difference between the absolute values of the average Bias. As such, for both metrics, a negative value represents an improvement.

	Models (MSA)	MAPE	Bias	Models (Original)	MAPE	Bias
Naive Models	NaiveMean	0.751	0.178	naive_mean_last_1	0.634	0.014
				naive_mean_last_3	0.636	0.001
	NaiveDrift	0.695	-0.053			
	NaiveSeasonal	0.596	0.066	seasonal_naive_12	0.31	0.063
Classical Models	ExponentialSmoothing	0.544	0.037	ses	0.601	0.020
				ets	0.491	0.047
	AutoARIMA	0.655	0.027	arima	0.414	0.062

Table 5.1: Univariate models forecast error metrics (Beverage Industry)

	Hierarchical Forecasts		Ensemble Forecasts	
	Mean Change	Models Improved	Mean Change	Models Improved
MAPE	0.157	4/18	0.189	0/3
Bias (absolute value)	0.033	4/18	0.012	1/3

Table 5.2: Hirearchical and Ensemble forecast error metrics summary (Beverage Industry)

	Models (MSA)	MAPE	Bias	Models (Original)	MAPE	Bias
Naive Models	NaiveMean	0.871	0.116	naive_mean_last_1	0.938	0.113
				naive_mean_last_3	0.920	0.159
	NaiveDrift	0.813	0.116			
	NaiveSeasonal	0.809	0.205	seasonal_naive_12	1.040	0.175
Classical Models	ExponentialSmoothing	0.818	0.158	ses	0.999	0.249
				ets	1.005	0.292
	AutoARIMA	None	None	arima	0.925	0.030

Table 5.3: Univariate models forecast error metrics (Electronics Retailer)

Analysis

The analysed aggregated results for the executions of the solutions for MAPE and Bias show different outcomes for each of the used datasets. The MSA system showed worse results than the

	Hierarchical Forecasts		Ensemble Forecasts	
	Mean change	Models improved	Mean change	Models improved
MAPE	-0.164	37/40	0.023	3/3
Bias (absolute value)	-0.262	20/40	-0.128	3/3

Table 5.4: Hirearchical and Ensemble forecast error metrics summary (Electronics Retailer)

original pre-decomposition solution in the target error metrics for the Beverage Industry dataset and an improvement in the Electronics retail benchmark.

A full comparison of the forecasting results for both larger and more variate datasets has to be done to obtain a complete picture of the forecasting performance of the solutions. Still, from the present comparisons, we argue that the functional requirements were met by the decomposed prototype, as it generates comparable predictions to the original solution.

5.3 Discussion

In this section, we will analyse our proposed design, the original monolithic solution, applying the assessment framework proposed by Auer et al. [7], exploring the alternatives under the categories of "Functional suitability", "Performance efficiency", "Reliability", "Maintainability", "Cost" and "Process related".

5.3.1 Functional suitability

When comparing the solutions, we can define the system's requirements using the included time series forecasting techniques.

By this measure, we can identify in the current approach and, by extension, in the proposed design, the inclusion of techniques not foreseen in the proposal by Uzun et al. [63]. These are Ensemble and Hierarchical forecasting.

Although not present in the prototype, the proposed design also allows for the definition of global models to be trained across datasets, building on the capabilities of the current solution. Finally, we also expect the future mapping of the deployment workflow detailed in section 3.2.1 to the same set of services as the ones defined for the diagnostic workflow in section 3.2.1. This requirement is not yet fulfilled and will be further discussed in section 5.3.4.

In section 5.2.3, the forecasting results were shown to be comparable to the ones generated by the original solution. This indicates that the new solution fulfills the same functionality.

5.3.2 Performance efficiency

The performance efficiency comparison focuses on hard metrics, gathered from the original solution and developed prototype.

The graph 5.2 shows the ability of the system to better adapt to "Time Behaviour" requirements. The system's response time is lowered by replicating its individually deployed components. This individual scalability of services allows the system to comply with the sporadic nature of the mapped workflows, adapting its resource utilisation through an autoscaler.

The proposed prototype does allocate more cloud resources from the overhead seen in Figure 5.1, and even more when scaling its components for the mentioned benefits. This is further addressed in section 5.3.5.

5.3.3 Reliability

While the individual service instances of the prototype have a lower mean time between failures than the original system, caused by the complexity added by the decomposition efforts, the fault tolerance of the design, through Kubernetes orchestration, allows for still high availability.

The decomposition also introduced the ability for the system to maintain a distributed state resilient to individual service shutdowns, completing the workflow when possible. At the same time, the previous solution had to be executed from the start in case of failure.

This is also relevant for Cloud computing costs as it allows the deployment in spot instances [1]. In contrast, the previous solution required the use of a server stable for the total duration of the diagnostic workflow, which often takes more than 12 hours. Spot instances significantly reduce the cost of the infrastructure (Figure 5.6).

5.3.4 Maintainability

The modularity of decomposed system increases code complexity by the addition of the template code and AMQP interface SDK (Figure 4.4), while within the time series specific tasks within them, it reduces code complexity, as we strip down dependencies used for non needed functionality in the extraction targets.

This common scaffold for implementing new microservices is highly reusable and standardises deployment and testing pipelines. The microservices themselves, in particular, the Command and Query services and the Forecasting dashboard, are applicable outside the context of the developed prototype.

The analyzability of the developed system is less than the original as it increases the number of independent executions, adding the complexity of the interaction between them. Even still, observability focused technologies (i.e., Prometheus, Loki, and Grafana) polling data from both the FastAPI endpoints and the AMQP instance were used to combat this issue, as well as to gather the results presented in section 5.2.2.

In terms of modifiability and changeability, adopting an event-driven asynchronous communication pattern for internal communications reduces coupling between services and allows for the seamless inclusion of new services, encapsulating new time series forecasting techniques. By

[1] Spot instances differ from On-Demand instances by utilising unused capacity of the Cloud provider's machines at a discount. The main drawback is the possibility of instance interruption, as the allocated capacity can be reclaimed by the provider when needed

subscribing to the routing keys of the relevant events and mapping them to their endpoints, a new event can be added to the workflow independently. Additionally, using the endpoints directly allows for easy testing, independent of the chosen message broker.

5.3.5 Cost

The development of the proposed system was done entirely in the context of this work. The effort of this development was in addition to the development costs of the original solution. Adding to this, the deployment costs of the new solution were also higher, as the coordination of a higher number of services introduces more complexity to the also developed deployment pipeline.

To draw comparisons of the infrastructure costs associated with both solutions, we simplified the cost structure by focusing solely on the costs of execution of the services, as the storage solution costs (DynamoDB and S3) and supporting infrastructure are either negligible or similar between the solutions.

Using as a baseline AWS's Elastic Compute Cloud R5 instances already used by LTPlabs, with 4 CPU cores and 32 GB of RAM, the graph in Figure 5.4 shows the number of needed machines by resource for the different executions presented in Figure 5.2.

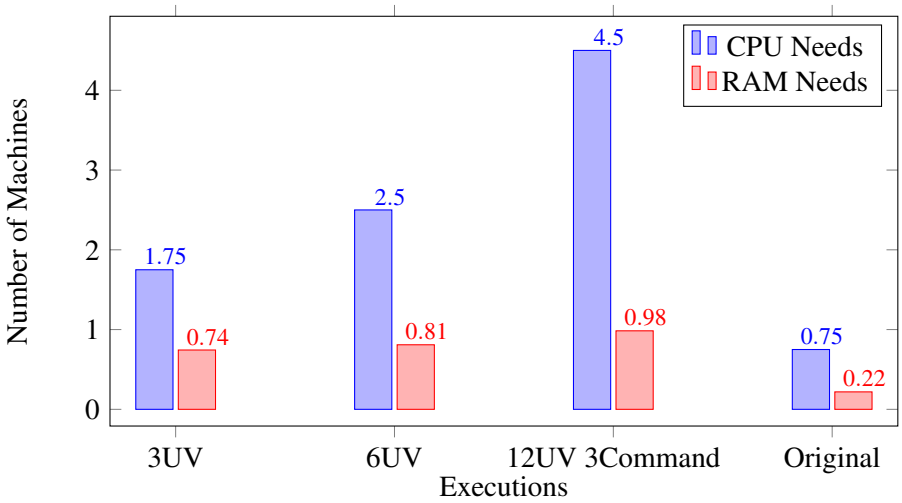

Figure 5.4: Allocated Machines (r5.xlarge)

The graph in Figure 5.5 details the execution time of the diagnostic workflow for the Beverage industry, multiplied by the number of allocated machines in Figure 5.4. We can see an improvement in the use of resources in the executions with more replicas and less total instance execution time.

The graph in figure 5.6 details a cost approximation of the mentioned executions, again only considering the resource allocation for the microservices themselves. For this, we considered the cost of 0.254$ per hour of use of an instance. We additionally detail the costs with the added consideration that the decomposed system allows for the use of spot instances. At the time of writing, this represented for LTPlabs a cost-saving of 60% for this type of instance.

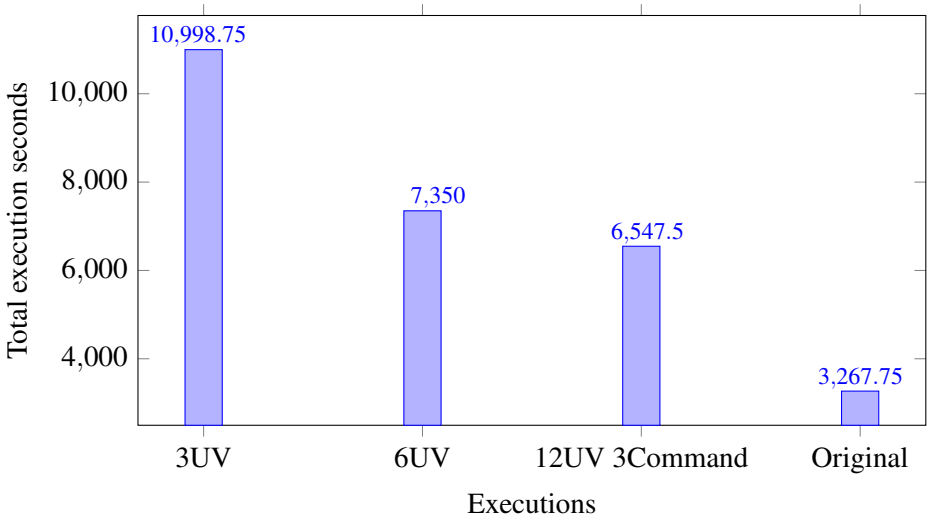

Figure 5.5: Total instance execution time (r5.xlarge)

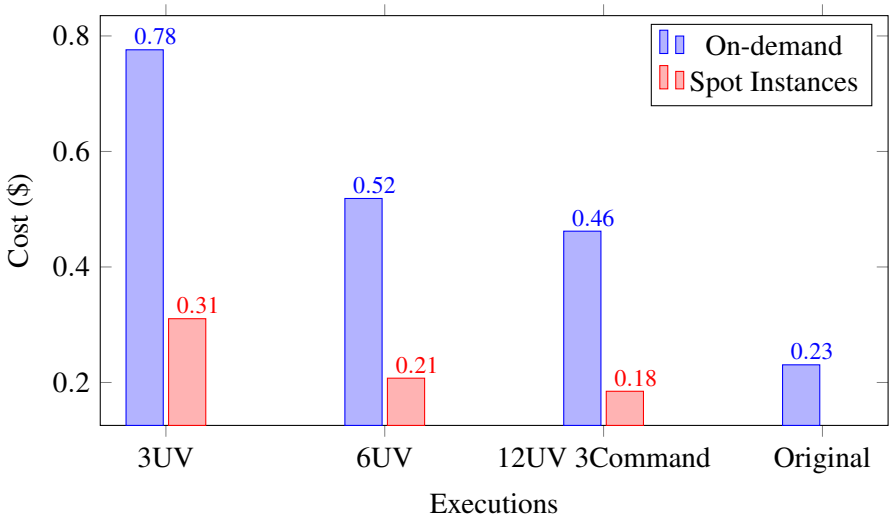

Figure 5.6: Cost approximation

We again note the simplifications made to the cost structure when analysing these results. Still, we can conclude that the prices of the executions are comparable, especially when using spot instances. The measured costs easily fall within the acceptable range for LTPlabs' context, and the reduced response time offsets any additional costs by reducing the individual Diagnostic project's lead times and associated development efforts.

5.3.6 Process related

In the category of process-related impacts, we highlight improvements in data management and the lowering of the diagnostic and deployment lead times.

By centralising the time series information across multiple projects, the management of this sensitive data can be standardised through AWS Identity and Access Management. Additionally, with the ability to extract cross-client insights with the now possible inclusion of global models detailed in section 2.2.4, LTPlabs may be able to generate more accurate time-series forecasts for its clients.

These lead times for diagnostic and deployment projects match the execution workflows detailed in section 3.2.1. With the reduced response time of the developed prototype, the Consulting project teams can generate forecasts quicker and iterate faster through different parametrisations, reducing the time to delivery of error metrics. Future deployment projects' lead time would also be reduced by the future mapping of this workflow to the implemented services, reusing the models trained in the diagnostic project.

5.4 Final notes

We propose that the objective of implementing a microservices-based architecture design was reached. As the prototype is still not fully realised, the continuous benefit of an iterative decomposition approach was also made evident by comparing the gathered metrics from the pre and post-decomposition solutions.

Even still, the increased costs of adoption, deployment, and maintenance, driven by the added complexity, should be considered before the migration of the current LTPlabs workflows to the new solution.

We stress the ability to extend the current prototype by including different forecasting models in the Univariate Prediction service and creating other services for multivariate approaches.

Chapter 6

Conclusions

Microservice architectures are inherently tied to their application in industry, as service boundaries are often defined by the set of business capabilities within a bounded context. Through this work, we aim to detail the impacts of a transition to this paradigm, informing industry practitioners of the followed approach and the measured and perceived outcomes.

In search of scalability of the monolithic solution for time series analysis and forecasting used by LTPlabs, we applied a decomposition technique, defined services and service boundaries, created a prototype of the design, and validated the approach through the comparisons presented in chapter 5.

This solution's shift to a microservices-based paradigm targets workflows often used within LTPlabs. Still, it was undertaken under the broader context of an organizational transition toward this paradigm. Creating standardized templates, event definition, and reusable services facilitated this change.

The applied decomposition technique proved to enable individual scalability of components as expected, reducing the execution time based on the available resources and the autoscaler configuration. With this, we were able to cut the execution time by more than 50%, as analysed in section 5.2.2, with comparable costs.

The execution time overhead added by the network level communication, new tooling for monitoring, and the added grouped dataset reconstruction step are negligible compared to the possible response time decrease gained by allocating more Cloud resources. The added infrastructure costs were reduced using spot instances, as shown in section 5.3.5.

The added extensibility of the system by creating new services representing more advanced forecasting techniques is of note, particularly when considering the tailor-made character of LTPlabs' value offer. Adding services to represent specific client needs, such as Causal, Probabilistic, and Multivariate Forecasting, is easier in the proposed design. By defining these particular needs as new services, new developments can be reused in similar future projects or used to upgrade ongoing deployments.

6.1 Future work

For further analysis of the current approach's suitability as a fully validated Microservices-based time series forecasting module, more univariate models should be included, and metrics should be generated for larger datasets.

Additional expansion should also be conducted by integrating more multivariate forecasting techniques. Monitoring this solution's development efforts would validate the gain in expansibility of the microservices-based solution.

The prototype is also still limited in input dataset size as it uses pandas data frames to internally represent the time series information. This limits the size by the allocation of RAM to the process, as the data frames are loaded into memory. We suggest processing the data in chunks or using the Dask API in place of pandas to scale to larger datasets. The collection of metrics to compare performance between these two alternatives would be a welcomed addition to this work.

References

[1] Ltp labs - about us, 2019. https://ltplabs.com/about-ltplabs/.

[2] Improving forecast accuracy with machine learning - architecture overview, 2020. https://docs.aws.amazon.com/solutions/latest/improving-forecast-accuracy-with-machine-learning/architecture-overview.html.

[3] Cloudevents - a specification for describing event data in a common way, 2022. https://cloudevents.io/.

[4] Time series forecasting - amazon forecasting, 2022. https://aws.amazon.com/forecast/.

[5] AGAPITOS, A., BRABAZON, A., AND O'NEILL, M. Regularised gradient boosting for financial time-series modelling. *Computational Management Science 14*, 3 (2017), 367–391.

[6] ALFARES, H. K., AND NAZEERUDDIN, M. Electric load forecasting: Literature survey and classification of methods. *International Journal of Systems Science 33*, 1 (2002), 23–34.

[7] AUER, F., LENARDUZZI, V., FELDERER, M., AND TAIBI, D. From monolithic systems to microservices: An assessment framework. *Information and Software Technology 137* (2021), 106600.

[8] BERRY, L. R., HELMAN, P., AND WEST, M. Probabilistic forecasting of heterogeneous consumer transaction–sales time series. *International Journal of Forecasting 36*, 2 (2020), 552–569.

[9] BOGNER, J., FRITZSCH, J., WAGNER, S., AND ZIMMERMANN, A. Microservices in industry: Insights into technologies, characteristics, and software quality. In *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, IEEE.

[10] BOYLAN, J. E., SYNTETOS, A. A., AND KARAKOSTAS, G. C. Classification for forecasting and stock control: a case study. *Journal of the operational research society 59*, 4 (2008), 473–481.

[11] BROWN, K., AND WOOLF, B. Implementation patterns for microservices architectures. In *Proceedings of the 23rd Conference on Pattern Languages of Programs*, pp. 1–35.

[12] BULINSKI, J., WASZKIEWICZ, C., AND BURACZEWSKI, P. Utilization of abc/xyz analysis in stock planning in the enterprise. *Annals of Warsaw University of Life Sciences-SGGW. Agriculture*, 61 Agric. Forest Eng. (2013).

[13] CERNY, T., DONAHOO, M. J., AND TRNKA, M. Contextual understanding of microservice architecture: Current and future directions. *Applied Computing Review 17*, 4 (2017), 29–45.

[14] CHEN, J., ZENG, G.-Q., ZHOU, W., DU, W., AND LU, K.-D. Wind speed forecasting using nonlinear-learning ensemble of deep learning time series prediction and extremal optimization. *Energy Conversion and Management 165* (2018), 681–695.

[15] CHEN, R., LI, S., AND LI, Z. From monolith to microservices: A dataflow-driven approach. In *2017 24th Asia-Pacific Software Engineering Conference (APSEC)* (2017), IEEE, pp. 466–475.

[16] COCKCROFT, A. Netflix in the cloud, 2010.

[17] DE LAURETIS, L. From monolithic architecture to microservices architecture. In *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, IEEE.

[18] DRAGONI, N., GIALLORENZO, S., LAFUENTE, A. L., MAZZARA, M., MONTESI, F., MUSTAFIN, R., AND SAFINA, L. *Microservices: Yesterday, Today, and Tomorrow*. Springer International Publishing, 2017, pp. 195–216.

[19] DUDEK, G. *Short-Term Load Forecasting Using Random Forests*. Springer International Publishing, 2015, pp. 821–828.

[20] DUDEK, G. Pattern-based local linear regression models for short-term load forecasting. *Electric Power Systems Research 130* (2016), 139–147.

[21] EDWARDS, M. Service component architecture (sca), 2011. http://www.oasis-opencsa.org/sca.

[22] FLIEDNER, G. Hierarchical forecasting: issues and use guidelines. *Industrial Management Data Systems 101*, 1 (2001), 5–12.

[23] FLOWER, M. *Patterns of Enterprise Application Architecture*. Pearson Education, Inc., Boston, MA, USA, 2002.

[24] FOWLER, M. *Patterns of Enterprise Application Architecture: Pattern Enterpr Applica Arch*. Addison-Wesley, 2012.

[25] FRANCESCO, P. D., MALAVOLTA, I., AND LAGO, P. Research on architecting microservices: Trends, focus, and potential for industrial adoption. In *2017 IEEE International Conference on Software Architecture (ICSA)*, IEEE.

[26] HAJIRAHIMI, Z., AND KHASHEI, M. Hybrid structures in time series modeling and forecasting: A review. *Engineering Applications of Artificial Intelligence 86* (2019), 83–106.

[27] HERZEN, J., LÄSSIG, F., PIAZZETTA, S. G., NEUER, T., TAFTI, L., RAILLE, G., POTTELBERGH, T. V., PASIEKA, M., SKRODZKI, A., HUGUENIN, N., DUMONAL, M., KOŚCISZ, J., BADER, D., GUSSET, F., BENHEDDI, M., WILLIAMSON, C., KOSINSKI, M., PETRIK, M., AND GROSCH, G. Darts: User-friendly modern machine learning for time series.

[28] HEWAMALAGE, H., BERGMEIR, C., AND BANDARA, K. Global models for time series forecasting: A simulation study.

[29] HYNDMAN, R. Cran task view: Time series analysis, 2022. https://cran.r-project.org/web/views/TimeSeries.html.

[30] HYNDMAN, R. J., AHMED, R. A., ATHANASOPOULOS, G., AND SHANG, H. L. Optimal combination forecasts for hierarchical time series. *Computational Statistics Data Analysis 55*, 9 (2011), 2579–2589.

[31] HYNDMAN, R. J., AND ATHANASOPOULOS, G. *Forecasting: principles and practice*. OTexts, 2018.

[32] JAMSHIDI, P., PAHL, C., MENDONCA, N. C., LEWIS, J., AND TILKOV, S. Microservices: The journey so far and challenges ahead. *IEEE Software 35*, 3 (2018), 24–35.

[33] KIRBY, L. J., BOERSTRA, E., ANDERSON, Z. J., AND RUBIN, J. Weighing the evidence: On relationship types in microservice extraction. In *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, IEEE.

[34] LEDELL, E., AND POIRIER, S. H2o automl: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML* (2020), vol. 2020.

[35] LEWIS, J. Micro services – java the unix way, 2012.

[36] LEWIS, J., AND FOWLER, M. Microservices - a definition of this new architectural term, 2014. https://martinfowler.com/articles/microservices.html.

[37] LIM, B., ARIK, S. O., LOEFF, N., AND PFISTER, T. Temporal fusion transformers for interpretable multi-horizon time series forecasting, 2020.

[38] LIU, Z., YAN, Y., AND HAUSKRECHT, M. A flexible forecasting framework for hierarchical time series with seasonal patterns. In *The 41st International ACM SIGIR Conference on Research Development in Information Retrieval*, ACM.

[39] MACKENZIE, C. M., LASKEY, K., MCCABE, F., BROWN, P. F., METZ, R., AND HAMILTON, B. A. Reference model for service oriented architecture 1.0. *OASIS standard 12*, S 18 (2006).

[40] MAKRIDAKIS, S., SPILIOTIS, E., AND ASSIMAKOPOULOS, V. The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting 36*, 1 (2020), 54–74.

[41] MAKRIDAKIS, S., SPILIOTIS, E., AND ASSIMAKOPOULOS, V. M5 accuracy competition: Results, findings, and conclusions. *International Journal of Forecasting* (2022).

[42] MAZLAMI, G., CITO, J., AND LEITNER, P. Extraction of microservices from monolithic software architectures. In *2017 IEEE International Conference on Web Services (ICWS)*, IEEE, pp. 524–531.

[43] MEGARGEL, A., POSKITT, C. M., AND SHANKARARAMAN, V. Microservices orchestration vs. choreography: A decision framework. In *2021 IEEE 25th International Enterprise Distributed Object Computing Conference (EDOC)*, IEEE.

[44] MICHAELAWYU. Cloudevents generator, 2019. https://github.com/michaelawyu/cloudevents-generator.

[45] MONTERO-MANSO, P., AND HYNDMAN, R. J. Principles and algorithms for forecasting groups of time series: Locality and globality. *International Journal of Forecasting 37*, 4 (2021), 1632–1653.

[46] MONTGOMERY, D. C., JENNINGS, C. L., AND KULAHCI, M. *Introduction to time series analysis and forecasting*. John Wiley Sons, 2015.

[47] MUNONYE, K., MARTINEK, P., AND IEEE. Evaluation of data storage patterns in microservices archicture. *2020 Ieee 15th International Conference of System of Systems Engineering (Sose 2020)* (2020), 373–380.

[48] MURER, S., AND HAGEN, C. Fifteen years of service-oriented architecture at credit suisse. *IEEE Software 31*, 6 (2014), 9–15.

[49] NEWMAN, S. *Building microservices*. " O'Reilly Media, Inc.", 2021.

[50] NTENTOS, E., ZDUN, U., PLAKIDAS, K., SCHALL, D., LI, F., AND MEIXNER, S. Supporting architectural decision making on data management in microservice architectures. In *European Conference on Software Architecture*, Springer, pp. 20–36.

[51] ORESHKIN, B. N., CARPOV, D., CHAPADOS, N., AND BENGIO, Y. N-beats: Neural basis expansion analysis for interpretable time series forecasting, 2020.

[52] POLIKAR, R. Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine 6*, 3 (2006), 21–45.

[53] RAMÍREZ, S., ET AL. Fastapi framework. *Github, https://github. com/tiangolo/fastapi* (2020).

[54] RICHARDSON, C. Microservices pattern: Command query responsibility segregation (cqrs), 2017. http://microservices.io/patterns/data/cqrs.html.

[55] RICHARDSON, C. Microservices pattern: Event sourcing, 2017. http://microservices.io/patterns/data/event-sourcing.html.

[56] RICHARDSON, C. *Microservices patterns: with examples in Java*. Simon and Schuster, 2018.

[57] SALINAS, D., FLUNKERT, V., AND GASTHAUS, J. Deepar: Probabilistic forecasting with autoregressive recurrent networks, 2019.

[58] SARAH, A., LEE, K., KIM, H., AND IEEE. Lstm model to forecast time series for ec2 cloud price. *2018 16th Ieee Int Conf on Dependable, Autonom and Secure Comp, 16th Ieee Int Conf on Pervas Intelligence and Comp, 4th Ieee Int Conf on Big Data Intelligence and Comp, 3rd Ieee Cyber Sci and Technol Congress (Dasc/Picom/Datacom/Cyberscitech)* (2018), 1085–1088.

[59] SHASHA, D. E., AND ZHU, Y. *High performance discovery in time series: techniques and case studies*. Springer Science Business Media, 2004.

[60] SHUMWAY, R. H., AND STOFFER, D. S. *ARIMA Models*. Springer International Publishing, 2017, pp. 75–163.

[61] TAIBI, D., LENARDUZZI, V., AND PAHL, C. Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. *IEEE Cloud Computing 4*, 5 (2017), 22–32.

[62] TAIBI, D., LENARDUZZI, V., AND PAHL, C. *Microservices Anti-patterns: A Taxonomy*. Springer International Publishing, 2020, pp. 111–128.

[63] UZUN, I., LOBACHEV, I., GALL, L., AND KHARCHENKO, V. Agile architectural model for development of time-series forecasting as a service applications. In *International Scientific Conference "Intellectual Systems of Decision Making and Problem of Computational Intelligence"* (2021), Springer, pp. 128–147.

[64] VISWANATHAN, S., WIDIARTA, H., AND PIPLANI, R. Forecasting aggregate time series with intermittent subaggregate components: top-down versus bottom-up forecasting. *IMA Journal of Management Mathematics 19*, 3 (2007), 275–287.

[65] VOHRA, D. Using configmaps. In *Kubernetes Management Design Patterns*. Springer, 2017, pp. 257–277.

[66] WINTERS, P. R. Forecasting sales by exponentially weighted moving averages. *Management science 6*, 3 (1960), 324–342.

[67] WIRTH, R., AND HIPP, J. Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining* (2000), vol. 1, Manchester, pp. 29–40.

[68] ZAMEER, A., ARSHAD, J., KHAN, A., AND RAJA, M. A. Z. Intelligent and robust prediction of short term wind power using genetic programming based ensemble of neural networks. *Energy Conversion and Management 134* (2017), 361–372.

[69] ZHANG, H., LI, S., JIA, Z., ZHONG, C., AND ZHANG, C. Microservice architecture in reality: An industrial inquiry. In *2019 IEEE International Conference on Software Architecture (ICSA)*, IEEE.

[70] ZIMMERMANN, O. Microservices tenets. *Computer Science - Research and Development 32*, 3-4 (2017), 301–310.

[71] ZIMMERMANN, O., MILINSKI, S., CRAES, M., AND OELLERMANN, F. Second generation web services-oriented architecture in production in the finance industry. In *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pp. 283–289.

Appendix A

Sequence diagram

In this appendix, we include for reference the sequence diagram mapping the diagnostic workflow of LTPlabs's time series forecasting solution to the decomposed MSA design. The services detailed operate in a publisher-subscriber model through AMQP, and rely on AWS' S3 to share large aggregations of data.

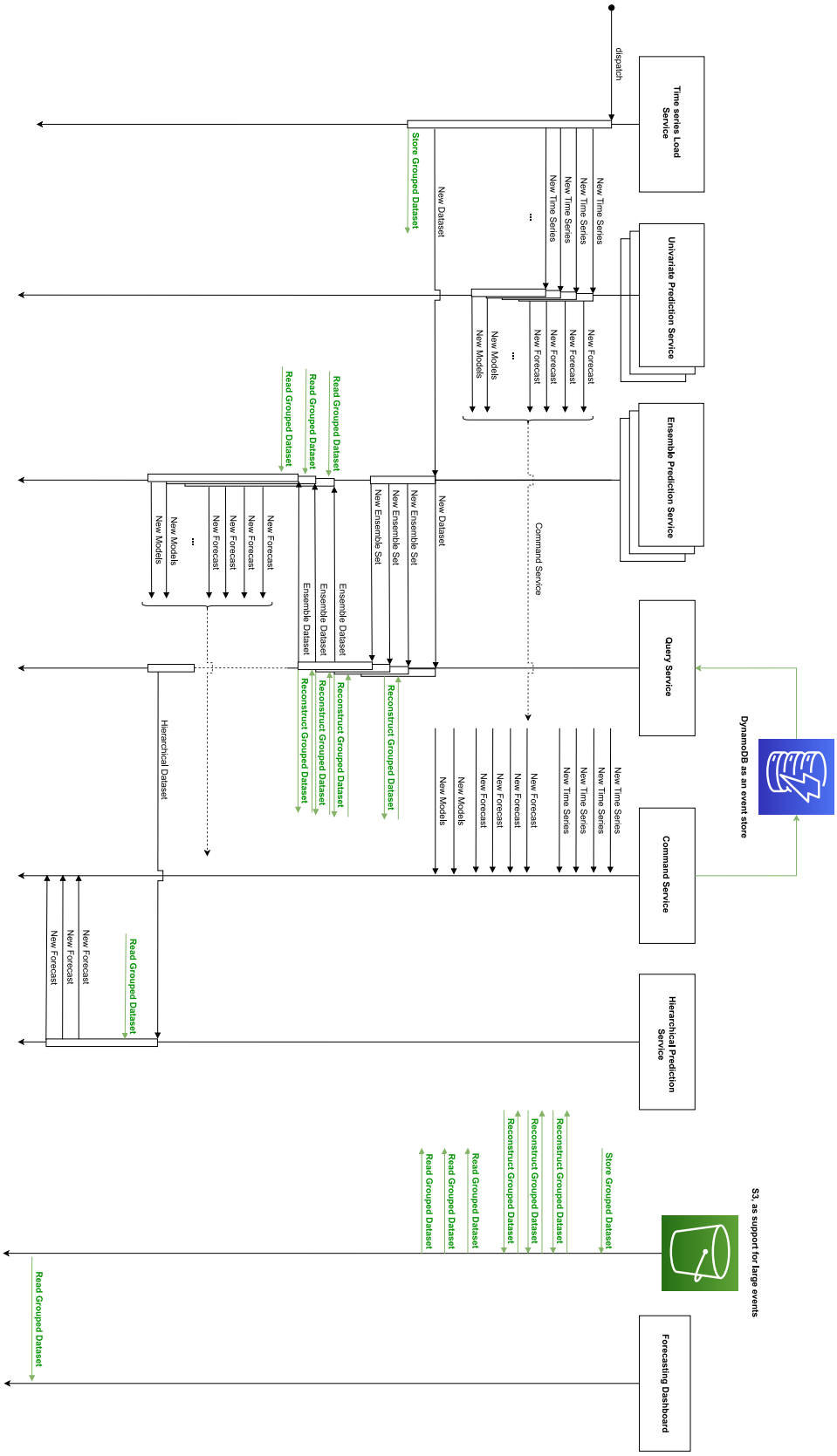

Figure A.1: Proposed sequence diagram

Appendix B

Error metrics comparison

Error metrics were collected from the forecasts for the test folds created by both the pre and post-decomposition systems. These error metrics were compared between both approaches for two representative datasets, a Portuguese Beverage producer and an Electronics retailer.

These datasets contain 350 individually labeled hierarchical time series but differ in levels of aggregation and time-series length, and forecastability.

From a broader subset of metrics gathered, this appendix details the Mean absolute percentage error (MAPE) and Bias of the predictions generated with ten backtesting folds for the individual Univariate forecasting models.

NaiveDrift, NaiveSeasonal, NaiveMean, Exponential smoothing, and AutoArima, implemented through the DarTS [27] were used for the decomposed MSA solution.

Naive mean with different time windows, seasonal naive, Simple Exponential Smoothing (SES), Exponential smoothing state space (ETS), and Arima were used for the original solution with R CRAN models [29].

Additionally, we detail the change in MAPE and in the absolute value of Bias for the ensemble and hierarchical forecasts from the original solution to the implemented microservices solution for each of the datasets.

We note that these metrics are only measured for the predictions with a gap of one, meaning the models are trained with all observations before the up to the month of the predicted values from which the aggregated metrics were gathered. The ten forecasted backtesting windows were used for the ensemble model training, but only the three last ones were used for this evaluation. The present evaluation is only done at the level of the disaggregated time series and, as such, only includes top-done hierarchical forecasts.

Note that for the Electronics retailer, no forecasts were generated by the AutoARIMA model as it requires a minimum of 30 observations for fitting, which was not met for this dataset.

	Models (MSA)	MAPE	Bias	Models (Original)	MAPE	Bias
Naive Models	NaiveMean	0.751	0.178	naive_mean_last_1	0.634	0.014
				naive_mean_last_3	0.636	0.001
	NaiveDrift	0.695	-0.05312			
	NaiveSeasonal	0.596	0.066	seasonal_naive_12	0.31	0.063
Classical Models	ExponentialSmoothing	0.544	0.037	ses	0.601	0.020
				ets	0.491	0.047
	AutoARIMA	0.655	0.027	arima	0.414	0.062

Table B.1: Beverage Industry univariate forecast error metrics

	Models	MAPE change	Bias change
Hierarchical forecasts	tdfp_hier2_cliente	0.272	0.019
	tdfp_hier1_cliente	0.281	0.018
	tdma_hier3_cliente	0.273	0.011
	tdfp_hier1	0.279	0.027
	tdfp_hier3_capacidade	0.305	0.107
	tdfp_cliente	0.260	-0.013
	tdfp_hier3	0.270	0.081
	tdfp_top	0.198	-0.027
	tdma_hier2_cliente	0.227	0.019
	tdma_hier1_cliente	0.212	0.018
	tdma_cliente	0.093	-0.013
	tdma_hier3_capacidade	0.014	0.106
	tdma_hier3	-0.016	0.081
	tdma_hier2	-0.037	0.077
	tdma_hier1	-0.084	0.027
	tdma_top	-0.296	-0.036
Ensemble forecasts	ens_lin_auto_all	0.159	-0.004
	gbm_all	0.276	0.006
	glm_all	0.133	0.034

Table B.2: Beverage Industry Ensemble and Hierarchical forecast error metric comparisons

	Models (MSA)	MAPE	Bias	Models (Original)	MAPE	Bias
Naive Models	NaiveMean	0.871	0.116	naive_mean_last_1	0.938	0.113
				naive_mean_last_3	0.920	0.159
	NaiveDrift	0.813	0.116			
	NaiveSeasonal	0.809	0.205	seasonal_naive_12	1.040	0.175
Classical Models	ExponentialSmoothing	0.818	0.158	ses	0.999	0.249
				ets	1.005	0.292
	AutoARIMA	None	None	arima	0.925	0.030

Table B.3: Electronics Retailer univariate forecast error metrics

	Models	MAPE change	Bias change
Hierarchical forecasts	tdfp_un_base_id_store_id	0.032	0.059
	tdfp_brand_id	-0.126	0.145
	tdfp_cat_id_brand_id	-0.025	0.181
	tdfp_subcat_id_store_id	0.073	0.128
	tdfp_un_id_brand_id	-0.031	0.157
	tdma_subcat_id_store_id	-0.018	0.040
	tdfp_brand_id_store_id	-0.242	-0.116
	tdfp_un_id_brand_id_store_id	-0.258	-0.136
	tdfp_un_base_id_brand_id	-0.090	0.121
	tdma_un_id_brand_id_store_id	-0.264	-0.141
	tdfp_cat_id_brand_id_store_id	-0.234	-0.095
	tdma_brand_id_store_id	-0.251	-0.128
	tdma_cat_id_brand_id_store_id	-0.239	-0.097
	tdfp_subcat_id_brand_id_store_id	-0.236	-0.123
	tdma_subcat_id_brand_id_store_id	-0.238	-0.124
	tdma_brand_id	-0.280	-0.033
	tdma_subcat_id_brand_id	-0.252	-0.115
	tdma_un_id_store_id	-0.226	-0.035
	tdma_un_id_brand_id	-0.196	-0.075
	tdfp_sku	-0.255	0.074
	tdfp_un_id_store_id	-0.147	0.082
	tdma_cat_id_store_id	-0.216	-0.041
	tdfp_un_base_id	-0.026	0.158
	tdma_cat_id_brand_id	-0.196	-0.052
	tdfp_cat_id_store_id	-0.134	0.072
	tdma_store_id	-0.063	0.085
	tdma_un_base_id_brand_id	-0.241	-0.112
	tdfp_store_id	0.209	0.412
	tdfp_top	-0.198	0.101
	tdma_subcat_id	-0.208	0.010
	tdfp_un_id	-0.188	0.194
	tdfp_cat_id	-0.187	0.195
	tdfp_subcat_id	-0.003	0.196
	tdma_sku	-0.310	-0.038
	tdma_un_base_id	-0.182	-0.006
	tdma_cat_id	-0.309	-0.013
	tdma_un_id	-0.312	-0.014
	tdma_top	-0.353	-0.111
Ensemble forecasts	gbm_all	-0.260	-0.115
	glm_all	-0.360	-0.262
	ens_lin_auto_all	-0.166	-0.007

Table B.4: Electronics Retailer Ensemble and Hierarchical forecast error metric comparisons