Dynamic Scaling of Distributed Dataflows Under Uncertainty

Stuart Jamieson

School of Computing
Newcastle University

This dissertation is submitted for the degree of
Doctor of Philosophy

May 2024

I would like to dedicate this thesis to my loving parents, I will be forever grateful...

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Stuart Jamieson
May 2024

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my primary supervisor, Dr Matthew Forshaw, for his unwavering support, guidance, and encouragement throughout my PhD journey. His expertise, insights, and patience have been invaluable in shaping my research and helping me navigate the challenges of this endeavour. I am truly fortunate to have had the opportunity to work under his supervision and learn from his vast knowledge and experience.

I would also like to extend my sincere thanks to my co-supervisors, Nigel Thomas and Stephen McGough, for their valuable input and support. Their expertise and feedback have greatly contributed to the development of my research, and I am grateful for their guidance and assistance.

A special thank you goes to Jennifer Wood, the Programme Manager, for her tireless efforts in keeping everything running smoothly and providing essential support throughout my PhD. Her dedication and assistance have been crucial in ensuring the success of my research and the timely completion of my thesis.

Finally, I am forever grateful to my family and friends for their unwavering love, understanding, and support throughout this challenging journey. Their belief in me has been a constant source of strength and motivation, and I could not have achieved this milestone without their endless encouragement and support.

Abstract

Performant Distributed Stream Processing Systems (DSPSs) are essential to processing large volumes of high-velocity data in a reliable and timely fashion. The global event stream processing industry is becoming ever more important to the world economy, projected to grow from its current $930 million valuation, to $2.4 billion by 2030. These systems commonly experience highly variable, bursty and unpredictable workloads, presenting a challenge when provisioning compute to meet the needs of these workloads. Rightsizing systems for peak demand leads to often-unacceptable financial cost, motivating the need for adaptive approaches to meet the needs of changing workloads. The choice of parallelism of workload operators are commonly governed by autoscalers, but their behaviour is often case specific and highly sensitive to the choice of tunable parameters and thresholds. Current approaches to evaluating DSPS and contemporary autoscaler performance provides limited visibility into the robustness of performance metrics, nor worst-case performance of systems under specific operating conditions. These issues present a challenge to practitioners wishing to understand the performance implications of their decisions.

In this thesis, we make the following contributions:

- Empirically study undesirable behaviours experienced by a state-of-the-art autoscaling controller and contribute a categorisation of failures exhibited by autoscaling mechanisms.
- Demonstrate the potential of moving average models to augment existing autoscalers to help mitigate these behaviours, successfully mitigating over 90% of undesirable extreme parallelism shifts and significantly reducing scaling-behaviour volatility.
- Challenge current approaches to quantifying and measuring streaming system robustness, and propose the use of non-parametric goodness-of-fit tests to quantify streaming system robustness.
- Investigate the potential of our suggested robustness quantifier and Response Surface Methodology (RSM) to capture the complex relationship between incoming workload characteristics and system latency, providing valuable insights into the behaviour of DSPSs under diverse operating conditions.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Introduction

We are in the era of "Big Data". We are witnessing a dramatic shift toward a data-driven economy, where the ability to efficiently analyse huge amounts of data, in a timely manner, is a key driver of commercial success. Many systems and applications in use today involve a large volume of continuous data streams along with large-scale, diverse, and high-resolution datasets that offer the potential for data-intensive decision-making. Autonomous vehicles, physical sensors, social-network activities (e.g. Facebook and Twitter), and stock-exchange markets are typical examples of such systems.

Distributed Stream Processing Systems (DSPSs) are used to process such streaming applications by distributing the workload among multiple nodes in a cluster. Because of the advantages offered by cloud infrastructure, DSPSs are frequently deployed in cloud-native environments. However, these "Big Data" applications can present significant challenges in terms of both latency and throughput for existing DSPSs.

Performance variability within DSPSs can arise from rapid changes in offered load, workload skew and performance interference observed when running atop public cloud infrastructures. Unpredictable circumstances, such as a higher than expected system load or variability, can cause parallel and distributed streaming systems to experience performance degradation [75, 106, 101, 102]. It is possible for a system to perform to acceptable levels under normal operating circumstances and yet exhibit catastrophic failure when subjected to slight disturbances [32]. Even a minor degradation in application performance can have a high penalty on system operators. Recorded instances of the effects of such performance degradation include:

- On March 12th 2015, Apple suffered a DNS issue in which the iTunes Store and App Store were taken offline. It took Apple 12 hours to get their systems restarted, costing the company about $25 million in lost sales [65].
- In 2017 part of Amazon Web Services stopped working, costing companies in the S&P 500 index $150 million [171].
- In 2021 a seven-hour global outage at Facebook cost the company up to $100million in direct revenue, with another $47 billion wiped off its stock market value in the company's second-worst day on record [149].
- In 2024 a two-hour outage at Meta (previously Facebook) that affected Facebook, Instagram and Messenger cost the company approximately $100 million in revenue due to the platforms being down worldwide [126].

Besides revenue loss and damaged reputation, enterprises spend many work-hours diagnosing and restoring services. The average cost of IT downtime is $5,600 per minute [178], with each server downtime event taking 117 minutes on average to resolve [18].

Service providers of large software system strive to offer assurances to their user base regarding Quality of Service (QoS) metrics. Fulfilling these assurances, in turn allows a certain level of user experience to be maintained. Service Level Agreements (SLAs) are usually entered into as a formal recognition of QoS assurances. Failure to comply with the SLA can often lead to a diminished user-base for a company's service and subsequent loss of revenues.

It is very common for commercial websites and online software applications to experience highly variable, bursty and unpredictable incoming workloads. These workloads typically display numerous characteristics including strong time-of-day and time zone correlation, flash crowd behaviour [170], cyclicality and general periodicity [85]. Such incoming workloads can make it difficult to plan a system's capacity requirements and ensure an SLA is fulfilled, while also keeping costs low. If capacity is set to manage expected peak load, performance will be maintained, but costs will rise accordingly as resources lay idle for significant periods of time. If capacity is set with to manage average expected load, costs will be curtailed, but performance will degrade when faced with above average loads.

To ensure high throughput and low latency with the massive amount of data, SPSs need to parallelise processing. This parallelism comes with two major challenges; first, how to parallelise the processing in SP operators, and second, how to continuously adapt the level of parallelisation when the conditions of the SP operators (e.g. the workload or available resources, change at runtime).

Performant DSPSs are essential to allow such huge volumes of high-velocity data to be processed in a reliable and timely fashion. Recent years have seen the emergence of myriad streaming systems, e.g. Apache Storm [181], Spark [162] and Flink [29], among others. With these systems have come efforts to develop DSPSs with the ability to scale system resources in an *on-demand* manner. One approach to this has been to implement the use of an *autoscaler*.

Streaming data differs from traditional organisational data not only in its size but also in its origin and characteristics. It is generated by distinct sources such as industrial sensors, clickstreams, servers, and user app activity, while conventional data stems from systems of record like Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), finance, and Human Resources (HR) systems. As a result, managing streaming data demands a specific set of tools rather than simply increasing the size of the database storage. The dissimilarities between these two types of data create unique challenges when handling streaming data within the traditional relational database models.

Traditional datasets are characterised by fixed time and space parameters, such as the current inventory level or the number of new employees hired in a specific period. In contrast, streaming data is continuously generated and delivered as a constant flow of small files that capture event-based interactions on a second-by-second basis, such as servers reporting their current operational status or a log of user activity on a mobile app. Attempting to query this data in batch Extract, Transform, Load (ETL) processes necessitates the establishment of arbitrary start and end points, resulting in difficulties related to data freshness, absent records, and synchronisation.

Because event-based data is generated at a high velocity and has a widely varied nature, it often lacks suitability for storage in traditional relational databases that power most systems and applications built on tabular data. Aside from the structural suitability consideration, the relative size of streaming data sets, and the fact they are unbounded can make it impractically expensive to store in the way non-streaming data tends to be.

Furthermore, conventional data sources are amenable to established methods of analysis and reporting, while streaming data's potential can be more easily realised via exploratory techniques, predictive modelling, and machine learning. These approaches necessitate flexible access to data, which is often hindered by any requirement to conform to existing traditional data warehousing storage paradigms.

Due to the inherent limitations of processing unbounded, high-velocity, streaming data using a traditional database storage and batch-processing architecture, streaming data specific architecture and DSPSs have become increasingly important for real-time data processing and analysis. DSPSs are designed to process data in real-time as it is generated,

allowing the system to handle data with a much higher velocity and volume than processing systems with a batch-oriented nature. Additionally, traditional data solutions typically involve a fixed schema for data, while stream processing systems are designed to handle data with a flexible schema. This allows stream processing systems to work with data that is constantly changing, such as data from social media or Internet of Things (IoT) sensors.

Another important benefit of DSPSs is their ability to provide real-time analytics and insights. Traditional data solutions are typically optimised for complex analytical queries, where the need for accuracy or complexity of the query outweighs the need for results in real-time. Stream processing systems can quickly detect trends or anomalies in data as they occur [133], rather than having to wait for the data to be processed in batches.

Finally, stream processing systems are typically designed to be highly scalable and fault-tolerant, whereas traditional data solutions may struggle with scalability and fault-tolerance when dealing with large amounts of data. This scalability element allows system resources to be scaled up or down as needed.

1.2 Problem Statement

Auto-scaling is a critical feature for DSPSs to ensure efficient resource utilisation and cost savings. Auto-scaling a system is a process that automatically scales the number of resources and aims to maintain an acceptable trade-off between achieving the required QoS and the cost of providing the resources needed to do so [131]. However, commonly, such autoscalers are governed by tunable parameters/thresholds; from the perspective of the user, choosing those values, determining when and how to resize the application, and defining a proper auto-scaling process often proves extremely difficult.

Autoscaler system behaviour can be highly sensitive to the choice of such input values, resulting in high levels of volatility, and therefore uncertainty regarding the outcome of any particular choice of such values. This presents a challenge to practitioners wishing to understand the performance implications of their decisions. Uncertainty in a business or general operating environment can be considered as undesirable.

Performance measurement of DSPSs and autoscaler behaviours often fails to capture sensitivity to changes in operating environment [111]. Such sensitivity is important to account for as a DSPS can display different ranking outcomes, versus another, when considering different measures of the same performance metric. For example, one may display lower overall latency but display higher variation (i.e. higher sensitivity) in the average, minimum and maximum across a set of latency measurements recorded in the face of differing workloads [111].

Furthermore, we often find performance metrics that fail to factor in the worst-case performance experienced throughout the test or describe the underlying distribution of collected values (e.g. not showing maximum latency along with system latency percentile metrics, not providing the significant moments' values of a reported metric's underlying distribution) [78, 130].

In this thesis, we will challenge commonly held beliefs around the operation of DSPS autoscalers and look to break a number of strong assumptions that often accompany state of the art autoscalers (e.g. linearity of performance under scaling, stability of workload arrival rate during scaling implementation). We empirically study undesirable behaviours experienced by a state-of-the-art controller, explore the feasibility of applying moving average models and demonstrate the potential to augment existing autoscalers to help mitigate these behaviours. Additionally, we raise the question of how DSPS robustness is to be quantified and measured. We evaluate several robustness metrics based on non-parametric goodness-of-fit tests, aiming to identify emerging best practice. Finally, draw on RSM to help systematically study and model the performance degradation of streaming systems under various workloads, and quantify the effects of different workload model parameters on system latency. RSM is a collection of mathematical and statistical techniques employed for the modelling and analysis of complex processes. We focus on both model performance and model stability and compare and contrast the results for each of our generated models.

1.3 Research Questions and Objectives

In order to investigate the above-mentioned research problems, we identify the following research questions and objectives:

1.3.1 Weighting and Windowing

In Chapter 4 we address the following research questions and objectives. This work culminates in a publication as detailed in Section 1.5.3.

1. **RQ1a:** How do we design our experimental approach and create the framework to successfully provide the quantitative and qualitative feedback necessary to begin modelling of the observed autoscaler behaviors.

   How to identify and model potential deficiencies in distributed stream processing system autoscalers.

2. **RQ2a:** What are the effects of differing workloads on a state-of-the-art autoscaler operation and output. How to broadly categorise autoscaler deficiencies, grouping together or differentiating between broad behavioural patterns and characteristics.

3. **RQ3a:** What constitutes an "improvement" in autoscaler behaviour and how that can be quantified and measured?

4. **RQ4a:** Whether selections of moving average models can be applied in a manner that successfully alleviates the observed autoscaler deficiencies.

5. **RQ5a:** DS2 is an automatic scaling controller for distributed streaming dataflows. It operates on a performance model that assumes operator instances repeatedly perform three activities in sequence: deserialisation, processing, and serialisation. When an operator instance is scheduled for execution, it pulls records from its input, deserialises them, applies its processing logic, and serialises the results, which are then pushed to the output. This model fits all types of operators in most modern streaming dataflow systems, including Heron [118], Flink [29], and Timely [28]. The DS2 *activation period* is defined as the number of consecutive policy decisions considered by the autoscaler before issuing a re-configuration. This research question investigates whether it is preferential to substitute the choice of the DS2 activation period input value with, instead, the choice of input parameter values relevant to the application of a range of windowing and weighting methods (i.e. can we show that it is "easier" to select a "good" set of windowing and weighting method input values and "easier" to get the activation period input value "wrong"?).

1. **RO1a:** Investigate how DS2 reacts to various workload generation functions across a range of input parameter values. Compare and contrast the differing outcomes, identifying and describing any over-arching manifestations.

2. **RO2a:** Categorise observations and group/segregate workloads and input parameter value ranges according to similarity or dissimilarity of outcomes.

3. **RO3a:** Study the effects on DS2 observed behaviour of applying various combinations of weighting methods and windowing approaches. Identify combinations and categories of effects identified that successfully lessen negative behaviour.

1.3.2 Robustness

In Chapter 5 we address the following research questions and objectives. This work culminates in a publication at detailed in Section 1.5.2.

1. **RQ1b:** How can we model and quantify the robustness of a DSPS in the face of changes in the characteristics of an incoming workload?

2. **RQ2b:** Can we identify emerging best practice which could then inform future performance analysis research considering robustness quantification in distributed systems?

1. **RO1b:** Study the sensitivity of streaming system performance to shocks or perturbations in incoming workload characteristics, across a range of representative workload generation and arrival rate functions.

2. **RO2b:** Using the KS-Statistic as a base, and a *One-Factor-At-a-Time (OFAT)* experiment design approach [55], quantify the level of system performance degradation (as applied to the Empirical Cumulative Distribution Functions (ECDFs) of recorded system latency measurements).

3. **RO3b:** Evaluate several robustness metrics based on non-parametric goodness-of-fit tests.

1.3.3 Response Surface Methodology

RSM is a statistical technique used for optimising processes, where a response of interest is influenced by several variables. The objective of RSM is to model the response as a function of input variables and find the optimal conditions for the desired outcome. This method involves designing experiments, estimating the response surface from experimental results, and using the model to optimise the response. RSM is widely applied in fields such as engineering, chemistry, and manufacturing for efficient process optimisation.

In Chapter 6 we address the following research questions and objectives.

1. **RQ1c:** How can we apply RSM approaches to systematically study the performance degradation of streaming systems under various workloads with the key aim of quantifying the effects on system latency of the various different system workload parameters?

2. **RQ2c:** Rather than just quantifying the effects on system latency (as in RQ1c), can we build upon that to understand and model the performance degradation of our distributed stream processing system under varying RSM experiment designs, workload characteristics and generative functions, latency percentiles measurements, and goodness-of-fit tests?

3. **RQ3c:** How can we quantify both model performance and model stability to provide recommendations and actionable insights to practitioners on which combination of methods to use?

1. **RO1c:** Using the Kolmogorov-Smirnov (KS), Weighted Kolmogorov-Smirnov (WKS), Anderson-Darling (AD), Cramér-von Mises (CVM), Epps-Singleton (ES), Kullback-Leibler Divergence (KLD) and Wasserstein Distance (WD), and 2^kF, CCD and BBD experiment design approaches, quantify the level of system performance degradation (as applied to the Empirical Cumulative Distribution Functions (ECDFs) of recorded system latency measurements).

2. **RO2c:** Capture the relationships between the factors and the response variable using a linear regression model, testing first-order models with interaction terms.

3. **RO3c:** Measure the model performance using predictions made by each fitted model. Apply a bootstrapping method using to run iterations for each goodness-of-fit test, latency percentile, workload model and experiment design combination and record all values and metrics of interest.

4. **RO4c:** Compare and contrast the models based on performance and stability of the fitted model.

1.4 Thesis Overview

This Section provides a brief overview of each chapter within the thesis.

Chapter 1 Provides an introduction to the thesis and describes the background and motivations behind the work carried out, highlighting the general problem statement and main contributions of the research. Finally, we describe the related peer-reviewed publications produced throughout the course of the PhD.

Chapter 2 Presents a comprehensive literature review covering stream processing, streaming system autoscalers, workload generation and modelling, and streaming system benchmarking and evaluation.

Chapter 3 Outlines the overall research methodology applied throughout the course of the PhD. It first describes the over-arching methodological paradigm followed, and follows with details of the research systems employed along with any relevant requirements, namely the DS2 autoscaler, Apache Flink streaming system and the chosen dynamic

workload generation models. It also presents as discussion of the limitations of this work and threats to validity.

Chapter 4 Systematically explores the impact of parameter tuning for a state-of-the-art autoscaler, identifying impacts in terms of Stability, Accuracy, Short settling time, and no Overshoot (SASO) properties [2, 108], as well as behavioural phenomena outside the scope of SASO. We empirically study undesirable behaviours experienced and contribute a categorisation of autoscaler mechanisms. The feasibility of using moving average models to augment existing autoscalers to mitigate a selection of those undesirable behaviour categories is also tested. Applicable methods are established to allow the systematic evaluation of these models. We demonstrate the potential of such an approach to produce more robust autoscaler decisions and significantly reduce a number of undesirable behavioural phenomena selected for analysis. The work presented in Chapter 4 was published at DEBS 2023 [98].

Chapter 5 Investigates the question of how to measure and quantify a DSPS's level of robustness in the face of disturbances in the operating environment. We present, compare and contrast a range of non-parametric goodness-of-fit tests which can act as quantifiers of a system's level of robustness. We show that different tests produce differing relative measures of system robustness, affected not only by the test statistics' inherent characteristics, but also by the particular performance metric (i.e. latency percentile) under scrutiny. The work presented in Chapter 5 was published at EPEW 2022 [97].

Chapter 6 Builds upon and develops out the foundations laid down in Chapter 5. We apply a selection of non-parametric goodness-of-fit tests (first presented in Chapter 5, with two new additions) and draw on RSM to help systematically study the performance degradation of streaming systems under various workloads. We quantify the effects of different workload model parameters on system latency. We focus on both model performance and model stability, and compare and contrast the results for each of our generated models, under varying RSM experiment designs, workload characteristics and generative functions, latency percentiles measurements, and goodness-of-fit tests.

Chapter 7 Summarises the conclusions of the work presented in this thesis and motivates future directions for work in the area.

1.5 Related Publications

Throughout the course of my PhD I have published in the following venues:

1.5.1 DEBS 2020

A Doctoral Symposium paper was released at the ACM International Conference on Distributed and Event-based Systems (DEBS). Over the past decade DEBS has become the premier venue for cutting-edge research in the field of event processing and distributed computing, and the integration of distributed and event-based systems in relevant domains such as Big Data, AI/ML, IoT, and Blockchain.

Stuart Jamieson. 2020. Dynamic scaling of distributed data-flows under uncertainty. In Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems (DEBS '20). Association for Computing Machinery, New York, NY, USA, 230–233. DOI: 10.1145/3401025.3406444. URL: https://doi.org/10.1145/3401025.3406444.

Abstract

Existing approaches to dynamic scaling of streaming applications often fail to incorporate uncertainty arising from performance variability of shared computing infrastructures, and rapid changes in offered load. We explore the definition and incorporation of risk and uncertainty, and advocate for risk-adjusted measures of performance and their application in improving the robustness of autonomic scaling of streaming systems.

1.5.2 EPEW 2022

A full paper was released at the European Performance Engineering Workshop (EPEW). EPEW 2022 aims to bring together researchers interested in understanding and improving the performance of systems where the flow of information is random by means of proper modelling and solution methods working on real-world or realistic applications of the methods applied in stochastic modelling, and on theoretical aspects arising as solutions to needs emerging from the study of real-world or realistic cases, across a broad spectrum of research fields.

Jamieson, S., Forshaw, M. (2023). Measuring Streaming System Robustness Using Non-parametric Goodness-of-Fit Tests. In: Gilly, K., Thomas, N. (eds) Computer Performance

Engineering. EPEW 2022. Lecture Notes in Computer Science, vol 13659. Springer, Cham. DOI: 10.1007/978-3-031-25049-1_1. URL: https://doi.org/10.1007/978-3-031-25049-1_1.

Abstract

Due to unpredictable disturbances in the operating environment, stream processing systems may experience performance degradation and even catastrophic failure. Streaming systems must be robust in the face of such uncertainty in order to be deemed fit for purpose. Measuring and quantifying a system's level of robustness is a non-trivial task. We present, compare and contrast a range of non-parametric goodness-of-fit tests which can act as quantifiers of a system's level of robustness. We show that different tests produce differing relative measures of system robustness, affected by not only the test statistics inherent characteristics, but also by the particular latency percentile under scrutiny

1.5.3 DEBS 2023

A full paper was released at the ACM International Conference on Distributed and Event-based Systems (DEBS). Over the past decade DEBS has become the premier venue for cutting-edge research in the field of event processing and distributed computing, and the integration of distributed and event-based systems in relevant domains such as Big Data, AI/ML, IoT, and Blockchain.

Stuart Jamieson and Matthew Forshaw (2023). On Improving Streaming System Autoscaler Behaviour using Windowing and Weighting Methods. In: Proceedings of the 17th ACM International Conference on Distributed and Event-Based Systems (DEBS). Association for Computing Machinery, 2023. New York, NY, USA. DOI: 10.1145/3583678.3596886. URL: https://doi.org/10.1145/3583678.3596886.

Abstract

Distributed stream processing systems experience highly variable workloads. This presents a challenge when provisioning compute to meet the needs of these workloads. Rightsizing systems for peak demand leads to often-unacceptable financial cost, motivating the need for adaptive approaches to meet the needs of changing workloads.

The choice of parallelism of workload operators are commonly governed by autoscalers, but their behaviour is often case specific and highly sensitive to the choice of tunable parameters and thresholds. This presents a challenge

to practitioners wishing to understand the performance implications of their decisions.

We systematically explore the impact of parameter tuning for a state-of-the-art autoscaler; identifying impacts in terms of SASO properties as well as behavioural phenomena such as extreme parallelism shifts and robustness.

Autoscalers commonly make decisions on instantaneous system performance, without incorporating historical information. This seeks to mitigate challenges of being overly influenced by historical values, to be able to respond in response to the evolving system state.

We demonstrate the potential to augment existing state-of-the-art autoscaling controllers with windowing and weighting methods to make more robust decisions, successfully mitigating over 90% of undesirable extreme parallelism shifts and significantly reducing scaling behaviour volatility.

Chapter 2

Literature Review

This chapter covers a number of topics relevant to this thesis, namely:

1. **Stream processing**: In Section 2.1 we provide a definition of stream processing and outline its importance and relevance. We discuss key concepts and terminology, common stream processing architectures, and highlight domain specific challenges and research directions.

2. **Autoscalers**: In Section 2.2 we provide a definition of steaming system autoscalers and outline their importance and relevance. We discuss the fundamental aims, goals and objectives of an autoscaler and include the inherent trade-offs and major challenges faced as a result. Common architectures and algorithms are presented, along with discussion of their applications, benefits and limitations, and domain specific challenges and research directions.

3. **Workload generation and modelling**: Section 2.3 we provide a definition of workload modelling and outlines its importance and relevance in relation to stream processing. We draw the distinction between fixed and streaming workloads, discussing common workload modelling methodologies and applications of workload modelling in Stream Processing Systems (SPSs) and discuss domain specific challenges and research directions.

4. **Benchmarking and evaluation**: In Section 2.4 we provide a definition of benchmarking and evaluation and outlines its importance and relevance in relation to stream processing, and include a historical development of benchmarking and evaluating stream processing systems. Benchmarking and evaluation methodologies and tools within the stream processing domain presented including discussion of domain specific challenges and research directions.

2.1 Stream Processing

Stream processing, also known as event processing, real-time analytics, or streaming analytics, is a computing paradigm that focuses on the continuous processing and analysis of unbounded data streams in real-time or near real-time [109]. These data streams consist of sequences of data elements generated at a high rate, typically from sources such as sensors, social media platforms, web applications, financial transactions, or Internet of Things (IoT) devices [4]. The primary goal of stream processing applications is to extract valuable insights and patterns from the incoming data as it is generated, without the need to store it in a database or other storage system first. By continuously aggregating, filtering, and analysing the data items, stream processing enables fast insight and rapid response to observed situations, making it a key component of various systems and applications.

The importance and relevance of stream processing in computer science stem from several factors [93, 159]:

1. **Real-time decision-making**: Stream processing allows organisations to make informed decisions in real-time by providing timely insights into data. This is especially critical in industries like finance, healthcare, and transportation, where immediate response to events or changes in data can have significant consequences.

2. **Scalability**: SPSs can handle large volumes of data generated at high rates, making it a suitable solution for modern big data applications. These systems are designed to scale horizontally, distributing the processing load across multiple nodes to manage the increasing volume of data.

3. **Fault-tolerance**: SPSs often implement fault-tolerance mechanisms to ensure data consistency and reliability, even in the case of node failures. This feature is crucial for applications that require high availability and accurate data processing.

4. **Flexibility**: Stream processing allows for the implementation of various data processing techniques, such as filtering, aggregation, and transformation, depending on the application requirements. This flexibility enables businesses to adapt their data processing pipelines to the changing needs of their organisations.

5. **Integration with other technologies**: SPSs can be integrated with other big data technologies, such as machine learning, data warehousing, or data analytics tools, to provide more comprehensive and valuable insights.

2.1.1 Background

The historical development of stream processing has been shaped by a series of pivotal milestones and influential contributions. This progression can be broadly categorised into several distinct phases. Collectively, they provide a comprehensive overview of the evolution of stream processing.

1. **Early beginnings (1960s-1990s)**: The conceptual underpinnings of stream processing can be traced back to the emergence of event-driven programming, a paradigm that emphasised processing data as events occurred, as opposed to the traditional batch processing approach [115]. Concurrently, computer scientists delved into data flow programming, which aimed to create systems capable of handling continuous data streams, thereby laying the groundwork for future advancements in stream processing.

2. **Data Stream Management Systems (DSMSs) (Late 1990s-2000s)**: As the necessity for real-time processing of data streams became increasingly apparent, the concept of DSMSs Data Stream Management Systems (DSMS) emerged to address the inherent challenges associated with this task [64]. Two pioneering projects in this area were the Telegraph project at the University of California, Berkeley, and the STREAM project at Stanford University. These groundbreaking initiatives introduced continuous queries and sliding window operators, which greatly enhanced the efficiency of processing streaming data.

3. **Publish/Subscribe Systems (Late 1990s-2000s)**: To facilitate asynchronous and real-time communication between distributed systems, publish/subscribe systems, commonly referred to as pub/sub systems, were developed [52]. These systems enabled efficient data dissemination between producers (publishers) and consumers (subscribers) and played a critical role in advancing real-time data processing capabilities.

4. **Apache projects and open-source stream processing (2010s)**: The stream processing landscape experienced a significant transformation with the advent of open-source projects such as Apache Kafka [160], Apache Storm [181], Apache Spark [198], Apache Flink [29], and Apache Samza [138]. These projects provided scalable, fault-tolerant, and high-performance SPSs that were rapidly adopted by numerous organisations to build sophisticated real-time data processing pipelines.

5. **Cloud-based stream processing (2010s)**: The proliferation of cloud computing prompted major cloud service providers, including Amazon Web Services [191], Google Cloud

Platform, and Microsoft Azure, to offer managed stream processing services (e.g. Amazon Kinesis, Google Cloud Dataflow, and Azure Stream Analytics). These services empowered organisations to deploy and scale SPSs without having to manage the underlying infrastructure, thereby streamlining the adoption of this technology.

6. **Integration with machine learning (2010s)**: As machine learning gained prominence, stream processing began to integrate with machine learning frameworks and libraries [72]. This integration facilitated the development of real-time machine learning models and applications in areas such as fraud detection, recommendation systems, and predictive analytics.

Throughout its evolution, stream processing has continually adapted to meet the growing demand for real-time data processing and analysis. Today, it plays an indispensable role in various industries and applications, enabling organisations to derive actionable insights from large volumes of data generated at high velocity.

2.1.2 Key Concepts and Terminology

Stream processing is a multifaceted domain that encompasses a range of key concepts and terminology, including the following:

1. **Stream Processing Engine**: The specific core component within an SPS that deals with the real-time processing of data streams. It consists of the computational algorithms and methods that operate on the incoming data streams. This includes tasks like filtering, aggregating, transforming, and analysing the streaming data.

2. **Event Processing**: Event processing, also known as Complex Event Processing (CEP), refers to the identification and analysis of meaningful events, patterns, or relationships within data streams. Complex Event Processing (CEP) systems can detect and respond to specific conditions, triggers, or patterns in real-time, facilitating rapid decision-making and action.

3. **Streaming Analytics**: Streaming analytics is a subfield of stream processing that focuses on the real-time analysis and processing of data streams in order to extract actionable insights. This term is often used interchangeably with **real-time analytics**.

4. **Data Stream**: A data stream is a continuous, unbounded sequence of data elements generated at a high rate from sources such as sensors, social media platforms, web applications, financial transactions, or IoT devices. Data streams are characterised by their dynamic nature and potential for rapid change.

5. **Source**: In stream processing systems, a source is the component responsible for ingesting data streams from external data producers. Sources can be connected to various data providers, such as message brokers, databases, or APIs, and can handle data serialisation and deserialisation, as well as schema management.

6. **Sink**: A sink is the component in a SPS that outputs the results of data processing to external systems, such as databases, message queues, or storage systems. Sinks are responsible for handling data serialisation and deserialisation, as well as managing connections and data consistency with the target system.

7. **Operator**: Operators are the processing elements in a SPS that perform transformations, computations, or aggregations on the data streams. Operators can be stateless, meaning they do not maintain any internal state between processing events, or stateful, meaning they maintain internal state to perform computations across multiple events.

8. **Parallelism**: Parallelism in SPSs refers to the concurrent execution of multiple instances of processing operators, enabling the system to process data streams more efficiently and achieve higher throughput. Parallelism can be achieved at the level of individual operators or across entire processing pipelines.

9. **Resource Allocation**: Resource allocation is the process of assigning computational resources, such as CPU, memory, and network capacity, to the various components of a stream processing system. Effective resource allocation is crucial for maintaining system performance, scalability, and cost-efficiency and can be guided by workload models and autoscaling algorithms.

10. **Autoscaling**: Autoscaling is the dynamic adjustment of resource allocations and processing parallelism in response to changes in workload patterns or system performance. Autoscalers use workload models, system monitoring data, and scaling policies to make resource allocation decisions, helping to ensure that SPSs maintain performance and resource utilisation targets.

11. **Data Stream Management System (DSMS)**: A DSMS is a specialised system designed to handle the unique challenges associated with processing data streams. It typically includes features such as continuous queries, sliding window operators, and mechanisms for handling out-of-order data.

12. **Continuous Query**: A continuous query is an ongoing query that processes data streams in real-time, as opposed to traditional queries which operate on static data

sets. Continuous queries enable the continuous extraction of insights from streaming data, allowing for real-time or near-real-time decision-making.

13. **Windowing**: Windowing is a technique used in stream processing to divide data streams into finite, manageable subsets called windows. Windows can be defined based on time intervals, the number of events, or custom criteria, and are used to perform aggregate computations or pattern detection on the data stream.

14. **Publish/Subscribe Systems (Pub/Sub)**: Pub/Sub systems are messaging frameworks that enable asynchronous, real-time communication between distributed systems. In this paradigm, data producers (publishers) disseminate data to data consumers (subscribers) who have expressed interest in specific data types or topics. Pub/Sub systems streamline data dissemination in real-time processing environments.

2.1.3 Stream Processing Architectures

Streaming system architecture refers to the design and organisation of the various software components and technologies that make up a distributed stream processing system, built specifically to ingest large volumes of streaming data from multiple sources with increased efficiency and reliability of data ingestion, processing and display. There are several architectural patterns that can be used to build a streaming system, each with its own set of trade-offs and considerations.

Lambda Architecture

The Lambda Architecture [116] (Figure 2.1) is designed to handle both real-time and batch processing in a single system. This architecture consists of three main components: a real-time layer, a batch layer, and a serving layer.

The real-time layer is responsible for processing the incoming data streams in real-time and providing near-instantaneous results. This is typically done using a stream processing engine such as Apache Storm, Apache Flink, or Apache Kafka Streams. The real-time layer is designed to handle high-speed data streams and provide low-latency results.

The batch layer is responsible for processing the data in larger batches, typically on a schedule, such as hourly or daily. This is typically done using a batch processing framework such as Apache Hadoop or Apache Spark. The batch layer is designed to handle large amounts of data and provide more accurate results.

The serving layer is responsible for providing a unified view of the results from the real-time and batch layers. This is typically done using a data storage and query technology such

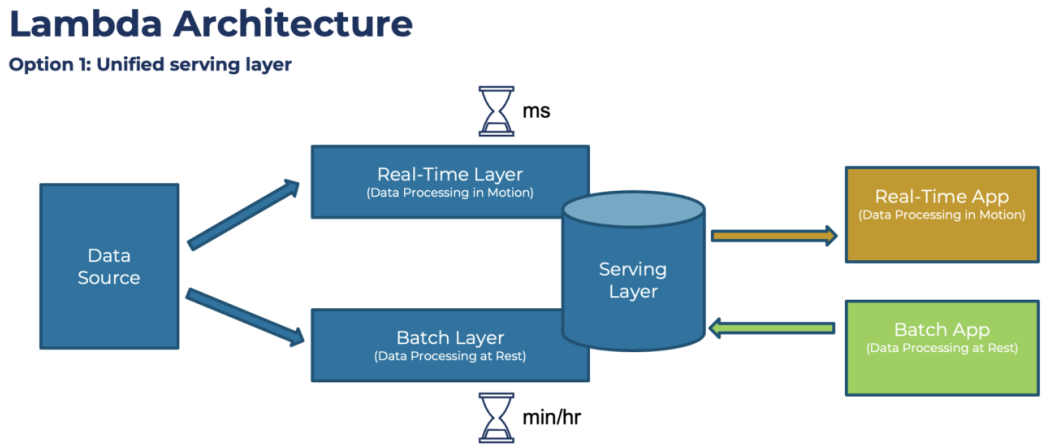


Fig. 2.1 A diagram illustrating the components and data flow of the Lambda Architecture [187], which consists of a batch layer, speed layer, and serving layer, to handle both batch and real-time processing of data.

as Apache Cassandra or Elasticsearch. The serving layer provides a single point of access for querying the results of both the real-time and batch layers.

Kappa Architecture

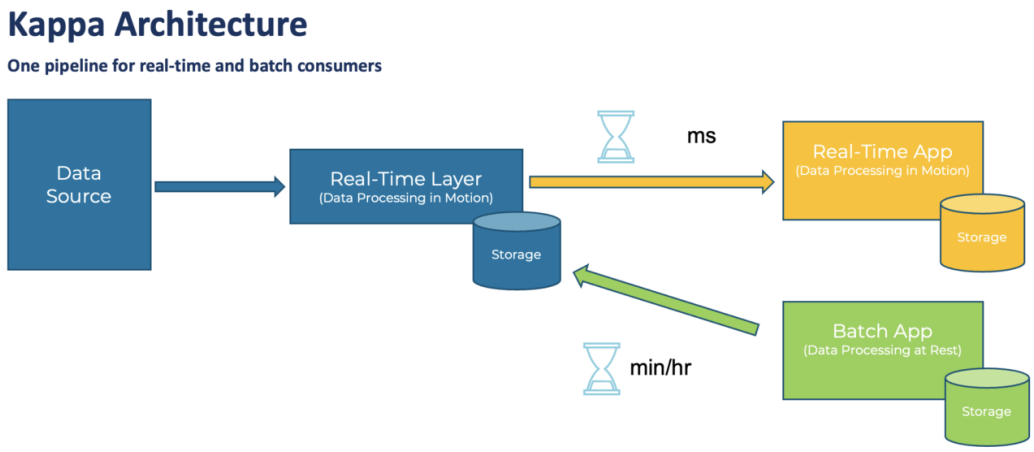


Fig. 2.2 A diagram depicting the Kappa Architecture [187], which uses a single stream processing engine to handle both real-time and batch processing, simplifying the architecture compared to the Lambda Architecture.

The Kappa Architecture [143] (Fig. 2.2) is designed to handle only real-time processing. This architecture consists of a single layer that processes the incoming data streams in real-time and also serves as the query and storage layer. This is typically done using a

stream processing framework such as Apache Kafka Streams, Apache Flink or Apache Pulsar, which also provide built-in storage and query functionality. The Kappa architecture is simpler than the Lambda architecture and does not require separate layers for real-time and batch processing, but it also does not provide the same level of accuracy as the Lambda architecture.

Pipe-and-Filter Architecture

The Pipe-and-Filter Architecture [142] consists of a series of independent filters that process the data streams in a pipeline fashion. Each filter is responsible for performing a specific task such as filtering, mapping, or reducing the data. This architecture is typically used when the system needs to be highly flexible and easily adaptable to changing requirements.

Micro-Batch Architecture

The Micro-Batch Architecture is a hybrid approach between real-time and batch processing. This architecture processes the data streams in small batches, typically on a schedule of seconds or milliseconds, rather than continuously. This allows for more complex processing and a better trade-off between latency and accuracy.

2.1.4 Stream Processing Systems

The stream processing model is a Directed Acyclic Graph (DAG) of stream processing nodes, connected through streams of events. The main objective of a Distributed Stream Processing Engine (DSPE) is to provide the infrastructure and APIs necessary to create and execute the stream processing graph with a continuous stream of messages. The runtime engine converts the User Graph to the Execution Graph.

	Kafka Streams	Spark Streaming	Apache Storm	Apache Samza	Apache Flink
Current Version	3.3.2	3.3.1	2.4.0	1.8.0	1.16.0
Category	ESP	ESP	ESP/CEP	ESP	ESP/CEP
Event Size	Single	Micro-batch	Single	Single	Single
Delivery Guarantees	At-least once	Exactly once, at least once	At-least once	At-least once	Exactly once
State Management	Local and distributed snapshots	Checkpoints	Record aknowledgements	Local and distributed snapshots	Distributed snapshots
Fault Tolerance	Yes	Yes	Yes	Yes	Yes
Out-of-Order Processing	Yes	No	Yes	Yes	Yes
Event Prioritisation	Programmable	Programmable	Programmable	Yes	Programmable
Windowing	Time-based	Time-based	Time-based, count-based	Time-based	Time-based, count-based
Back-Pressure	N/A	Yes	Yes	Yes	Yes
Primary Abstraction	Kafka Streams	Dstream	Tuple	Message	DataStream
Data Flow	Processing topology	Application	Topology	Job	Streaming dataflow
Latency	Very low	Medium	Very low	Low	Low
Resource Management	Any process manager	YARN, Mesos	YARN, Mesos	YARN	YARN
Auto-Scaling	Yes	Yes	No	No	No
In-Flight Modifications	Yes	No	Yes	No	No
API	Declerative	Declarative	Compositional	Compositional	Declarative
Core Language	Java	Scala	Clojure	Scala	Java
API Languages	Java	Scala, java, Python	Scala, Java, Clojure, Python, Ruby	Java	Java, Scala, Python

Table 2.1 Overview of Stream Processing Systems

There are two main types of processing engines:

1. Open-Source Composition-Based Engines: Composition-based stream processing engines depend on the initial establishment of the Directed Acyclic Graph (DAG) prior to data processing. This approach simplifies the code, but it requires developers to meticulously design their framework to prevent processing inefficiencies. These engines represent the first generation of stream processors and can be challenging to manage.

2. Managed Declarative Engines: Managed declarative engines enable the chaining of stream processing functions. As a result, the engine determines the DAG while receiving data and can optimise the DAG during execution. This Stream Processing Engine (SPE) category is more manageable and offers various managed service options. However, the initial setup of the pipeline can be a costly endeavour, with expenses covering all aspects from source to storage and analysis.

Five of the most popular SPSs (in no particular order), are Apache Samza, Apache Storm, Apache Spark, Apache Flink and Kafka Streams. Table 2.1 gives an overview of these four systems, along with Kafka Streams, another popular SPS.

Apache Samza

Overview: Apache Samza is a distributed stream processing framework designed for stateful processing of data streams, originally developed at LinkedIn and designed to work with Apache Kafka for message passing. It uses a compositional engine and is based on the concept of Publish/Subscribe Task. It listens to the data stream, processing messages as they arrive and passes on the outputs to another stream. The stream can be broken down into multiple partitions with a copy of the task spawned for each individual partition.

Strengths: Samza has native support for stateful processing, which is useful for applications that require maintaining state across processing steps. It also provides fault tolerance and durability, ensuring that state is not lost during processing. Additionally, Samza integrates well with Apache Kafka, which makes it suitable for organisations already using Kafka.

Weaknesses: Compared to the other frameworks, Samza has a smaller community and fewer built-in features. Furthermore, its primary focus on Kafka may not be suitable for organisations using other messaging systems.

Use cases: Apache Samza is well-suited for applications that require stateful processing, such as event-driven applications and real-time analytics, particularly when integrated with Apache Kafka.

Apache Storm

Overview: Apache Storm is a distributed real-time computation system designed for processing large volumes of data at high speeds. Storm was one of the first open-source stream processing frameworks and has been widely adopted in the industry. It uses a compositional engine and is based on the concept of Spouts and Bolts. Spouts are sources of information which are pushed to the Bolts, which carry out the data processing. Bolts can be chained together, and the Spout/Bolt topology becomes the DAG.

Strengths: Storm is known for its low-latency processing capabilities, which enables real-time analytics and processing. It is also highly scalable and fault-tolerant, allowing it to handle large volumes of data and recover from failures gracefully. Additionally, Storm supports a wide range of data sources and processing languages, offering flexibility in implementation.

Weaknesses: Storm lacks native support for stateful processing and windowing, which can make it challenging to implement certain types of applications. Moreover, Storm's programming model can be more complex than those of other frameworks, which might lead to a steeper learning curve.

Use cases: Apache Storm is ideal for applications that require low-latency processing and real-time analytics, such as fraud detection, recommendation engines, and network monitoring.

Apache Spark

Overview: Apache Spark is a general-purpose distributed data processing engine that supports batch processing, interactive queries, machine learning, and graph processing in addition to stream processing through its Spark Streaming module. It uses a declarative engine and is based on the concept of Resilient Distributed Datasets (RDDs), (i.e., immutable tables of data). The RDDs are split up into pieces and sent to workers to be executed, similar to the map-reduce concept with each worker processing its own pieces and combining the output to create a complete final output. The code defines the processing functions to be applied to the data, and Spark infers the DAG from the functions.

Strengths: Spark provides a unified Application Programming Interface (API) for both batch and stream processing, making it easy to transition between the two. It also offers built-in libraries for machine learning and graph processing, enabling more advanced analytics. Spark's in-memory processing capabilities allow for faster data processing compared to other frameworks.

Weaknesses: Spark Streaming processes data using micro-batches, which may result in higher latency compared to other frameworks like Storm or Flink that process data on a per-record basis. Additionally, while Spark supports stateful processing, its support is not as robust as in Samza or Flink.

Use cases: Apache Spark is well-suited for applications that require both batch and stream processing, or those that require advanced analytics, such as machine learning and graph processing. Examples include Extract, Transform, Load (ETL) pipelines, recommendation systems, and predictive analytics.

Apache Flink

Overview: Apache Flink is a distributed stream processing framework that supports both stream and batch processing. Flink is designed for low-latency, high-throughput, and stateful processing of data streams. It uses a declarative engine and is based on the concept of Streams and Transformations. Data enters the system via a Source and exits via a Sink, with the Streams and Transformations making up a flow of data through the system. The DAG is implied by the ordering of the Transformations, which can be re-ordered by the engine if it is detected that one Transformation does not depend on the outcome from a previous Transformation.

Strengths (continued): Flink's support for event-time processing and advanced windowing functions allows for more accurate handling of out-of-order data and complex event patterns. It also provides low-latency processing and strong support for stateful processing, which makes it suitable for various use cases. Like Spark, Flink offers a unified Application Programming Interface (API) for batch and stream processing, making it easy to switch between the two modes.

Weaknesses: Flink's focus on advanced features and a flexible programming model may lead to a steeper learning curve for newcomers.

Use cases: Apache Flink is suitable for a wide range of applications, including real-time analytics, event-driven applications, and complex event processing. Examples include fraud detection, anomaly detection, and real-time dashboarding.

Kafka Streams

Overview: Kafka Streams is a lightweight stream processing library that is part of Apache Kafka. It is designed to work natively with Kafka for building real-time data processing applications. Kafka Streams is not a standalone distributed processing framework like the others mentioned, but rather a library that can be integrated into applications to process

Kafka data streams. It uses a declarative engine and offers advanced windowing and stateful processing capabilities.

Strengths: Kafka Streams is tightly integrated with Kafka, which allows for seamless handling of Kafka data streams and offers strong support for stateful processing. Kafka Streams is lightweight and has a lower operational overhead compared to other distributed processing frameworks, making it easier to deploy and manage. It also offers event-time processing and advanced windowing functions, similar to Apache Flink.

Weaknesses: Kafka Streams is limited to processing data from Kafka and lacks built-in connectors for other data sources. Additionally, since it is a library rather than a full-fledged distributed processing system, it may not be suitable for applications with very large-scale processing requirements or those requiring complex topologies.

Use cases: Kafka Streams is well-suited for applications that require real-time processing of Kafka data streams, particularly when stateful processing is needed. Examples include real-time analytics, event-driven applications, and data enrichment pipelines.

Apache Flink was chosen as the stream processing system for the experiments in this thesis due to several key factors:

- **Stateful processing:** Flink's native support for stateful processing and event-time processing aligns well with the requirements of the experiments, which involve analysing system behavior and performance under various workload conditions.
- **Low-latency and consistency:** Flink's ability to provide low-latency processing and strong consistency guarantees is crucial for accurate and reliable performance measurements.
- **Flexible windowing and state management:** Flink's advanced windowing and state management capabilities allow for more sophisticated analysis and modeling of system behavior.
- **Unified API:** Flink's unified API for batch and stream processing simplifies the development and deployment of experimental workloads.
- **Active community and rich ecosystem:** Flink has a large and active community, as well as a rich ecosystem of connectors and libraries, which can facilitate the integration of experimental workloads with various data sources and sinks.

2.1.5 Challenges and Research Directions

Despite the numerous advantages that stream processing provides, it is not without its challenges and limitations. As the demand for real-time processing and analysis of large-

scale data streams grows, researchers and practitioners face several hurdles in optimising and expanding the capabilities of stream processing frameworks. Some key challenges and limitations include:

1. **Scalability** [140]: As data streams continue to grow in volume and velocity, SPSs must efficiently scale to handle the increased load. Ensuring horizontal scalability, wherein additional resources can be added to distribute the processing load, is crucial for maintaining high performance and low latency. However, achieving linear scalability in distributed systems is a challenging task due to factors such as network latency, data partitioning, and workload distribution.

2. **Fault Tolerance and Reliability** [189]: SPSs must provide fault tolerance and reliability guarantees to handle failures in distributed environments. This involves maintaining state consistency, ensuring at-least-once or exactly-once processing semantics, and implementing robust recovery mechanisms. Achieving these guarantees while maintaining low latency and high throughput is a complex challenge that requires careful design and implementation of the underlying system components.

3. **Handling Out-of-Order Data** [188]: Data streams often exhibit out-of-order arrivals due to network delays, clock skew, or other factors. This poses challenges for stream processing systems, particularly when processing time-sensitive data or performing event-time-based analytics. Developing mechanisms for handling out-of-order data while maintaining low latency and preserving processing semantics is an ongoing research area.

4. **Resource Management** [38]: Efficient resource management is critical for optimising the performance of stream processing systems. This includes allocating resources such as Central Processing Unit (CPU), memory, and network bandwidth based on the workload characteristics and dynamically adjusting resource allocation in response to changing workloads. Designing effective resource management strategies that balance performance, cost, and energy efficiency is a challenging task, particularly in large-scale, multi-tenant environments.

5. **Complex Event Processing and Pattern Detection** [196]: Many real-world stream processing applications require the detection of complex patterns and event sequences within data streams. Developing expressive and efficient techniques for specifying and identifying complex patterns is an ongoing research challenge. Additionally, implementing these techniques in a scalable and fault-tolerant manner requires careful consideration of system design and resource management.

6. **Integration with Existing Systems**: SPSs often need to interact with existing data storage, processing, and analytics infrastructure. This requires seamless integration with various data sources, sinks, and processing tools, which can be a challenge due to differences in data formats, APIs, and processing semantics. Addressing these integration challenges is essential for ensuring the smooth operation and adoption of SPSs in complex data processing environments.

2.2 Autoscalers

The variability in structure and arrival rate of current day streaming data poses a serious challenge to software systems and applications designed to ingest and process such data. With the advent of the cloud computing paradigm [135], it is now possible for businesses and streaming system operators to scale provided resources up and down to manage variability in the incoming workload rate and type. There exist ongoing efforts to design processes and solutions which are able to manage the scaling up and down of streaming system resource provision in an autonomous fashion.

A system that automatically adjusts the resources needed by an application is called an *autoscaling system* or *autoscaler*. The aim of an autoscaler is to:

1. Dynamically adjust the amount of resources made available to an application or system, depending on the input workload.

2. Find the optimal trade-off between meeting the SLA of the application and minimising the cost of resource provision.

If carried out successfully, autoscaling offers numerous benefits including improving customer experience and maintaining a cost-efficient infrastructure. By enabling quick and automated adjustments to changes in demand, auto-scaling helps organisations keep up with increases in system load and to deliver a positive customer experience, the most important yardstick by which organisations measure application performance. Autoscaling can also help organisations minimise their expenses, even when there is a sudden and unforeseen growth in demand that puts a significant strain on their systems. Rather than having to invest in additional infrastructure to cope with such increased peak workloads, which may remain unused for prolonged periods when facing sub-peak rate incoming workloads, companies can now pay only for the resources they currently need and expand their capability to handle sudden surges in demand.

Every business that operates some form of commercial website or online software application, which can experience unpredictable incoming workloads and the performance

of which needs to be maintained within certain levels, could benefit from a successfully implemented auto-scaling system. For example, Netflix [20] and Facebook [3], two massive global services, both employ autoscaling systems to manage the load on their services.

2.2.1 Background

Performance optimisation for real-time streaming applications is a non-trivial process. The goal of the optimisation process is to find an optimal trade-off between two states of an application: underutilisation (i.e., under-provisioning) and overloading (i.e., over-provisioning). In this sense, "optimal" means the most desirable or favourable outcome, considering the constraints and priorities of the system. It should also aim to find this optimal level without experiencing *oscillation*. These three concepts are defined as follows:

- Over-provisioning: the system or application is provided with more resources than are needed. This results in resources operating below their capacity, or indeed sitting idle. In many real-world instances, some controlled level of over-provisioning of resources is effected to cope with minute workload fluctuations and is considered desirable [131].
- Under-provisioning: the system or application is not being provided with enough resources needed to operate at a level to comply with the SLA in place.
- Oscillation: oscillations manifest when scaling decisions are enacted in rapid succession, acquiring resources and then releasing them, or vice versa, resulting in frequent occurrences of over-provisioning and under-provisioning. This behaviour causes resource wastage and can lead to SLAA breaches.

There are numerous major challenges that arise when attempting to solve this trade-off. Each challenge can be mapped to one of the four components within the overall autoscaler design process. The auto-scaling process is often abstracted as a *Monitor, Analyse, Plan, Execute (MAPE)* control loop (Figure 2.3), with the four components stated [114].

- Monitoring: During the Monitoring phase, the system must observe certain performance metrics to assess whether scaling operations are necessary and how they should be carried out. This phase poses significant challenges, such as:
  - Performance indicators: Selecting appropriate performance indicators, which is crucial for the autoscaler's effectiveness.

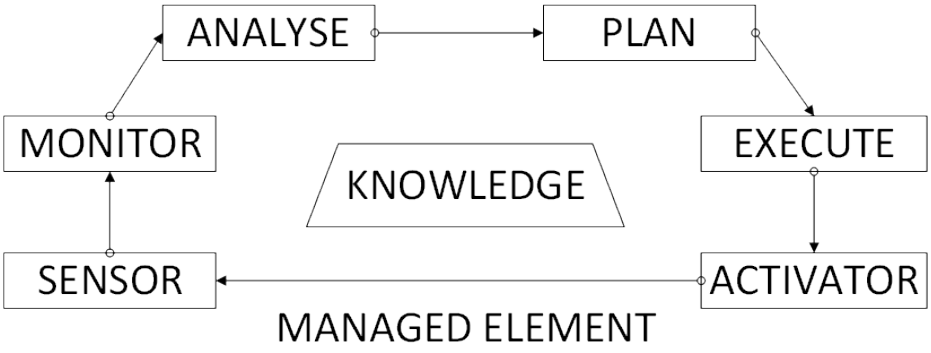


Fig. 2.3 MAPE Loop

  - Monitoring interval: The sensitivity of the autoscaler is determined by the monitoring interval. Shorter intervals provide greater sensitivity but result in higher monitoring costs and can cause oscillations, while longer intervals can slow down the system's response time.

- Analysis: The system determines whether a scaling action is necessary based on the information received during monitoring. The major challenges are:

  - Scaling timing: Determining when to scale, whether to proactively take scaling action based on predicted workload changes or to react to changes in workload.
  - Workload prediction:Accurately predicting future workload if proactive scaling is the chosen approach.
  - Adaptivity: Ensuring adaptivity of the autoscaler's model and settings when faced with significant changes in workload or streaming system application.
  - Mitigating oscillations caused by frequent, contradictory actions in a short time, which result in resource wastage and more SLA violations.

- Planning: During the Planning phase, the system must estimate the number of resources to be provisioned or de-provisioned in the scaling action and optimise resource composition to minimise financial costs. The major challenges are:

  - Resource estimation: Quickly estimating the necessary resource level to handle the current or incoming workload without the ability to observe the results of executing the scaling plan.
  - Resource combination: Selecting the combination of factors that optimises the total resource cost for the provisioned resources. This presents a significant opti-

misation challenge as the optimisation space is extensive, and efficient solutions must be found within a short period of time.

- Execution: Responsible for actually executing the scaling plan. The major challenges are:
  - Engineering flexibility: Ensuring engineering flexibility, as executing the plan involves calling cloud providers' APIs. Supporting APIs from different providers is challenging from an engineering perspective.

Approach	Logic	Advantages	Limitations
Static and threshold-based policies	Based on chosen performance metrics and pre-defined thresholds	Simplicity and ease of application	Depends on threshold quality, requires tuning and susceptible to system oscillations under bursty workloads
Reinforcement learning	Trial-and-error approach without using any a priori knowledge or model of the system	No prior knowledge of system and target scenario required	Long learning phases and convergence times in real-world applications
Queuing theory	Based on estimates of different performance metrics by modelling the system as a queue of requests	Computational efficiency	Too rigid and need to be recomputed following changes in the application or workload
Control theory	Based on measurements that are influenced both by external stimuli as well as its own actions	Closed-loop operation	Accuracy may decrease for new workload patterns
Time series analysis	Application of a range of methods to detect patterns and predict future values on a sequence of data points	Main enabler of proactive auto-scaling techniques	Sensitive to technique and parameter selection

Table 2.2 Overview of Overview of Auto-scaling Approaches for DSPSs

2.2.2 Autoscaler Architectures and Algorithms

Autoscaling involves dynamically adjusting the resources of a system in response to changes in the workload, to ensure that the system can handle the current and anticipated demands. Dynamic scaling systems commonly rely on two components [33, 63, 78, 194]: firstly, a centralised subsystem that collects current information about network traffic and available resources to make informed decisions that optimise a specific performance metric, and secondly, a scaling policy that determines when to initiate a scale-out, scale-in, or reconfiguration. There are several autoscaler architectures that can be used to achieve this, each with its own benefits and challenges. Figure 2.4 provides a visual taxonomy of the autoscaler landscape, while Table 2.3 provides an overview of auto-scaler approaches in state-of-the-art literature (Extension of Table 1 found in [108] - number of autoscalers included increased from 13 to 22, and a more comprehensive legend provided). This extension allows for a more thorough comparison of various autoscaling approaches in DSPSs. Further details of each category are provided below.

Type

In deciding when to scale a system there are two main groups into which an autoscaler can be classified:

1. **Reactive Autoscaling**: Reactive autoscaling is a widely-used approach that adjusts resources based on the observed performance of the system and scale only when judged necessary according to the current environment. It monitors key performance metrics, such as Central Processing Unit (CPU) utilisation, memory usage, and throughput, and triggers scaling actions (e.g., adding or removing processing nodes) when these metrics breach predefined thresholds. Reactive systems are often preferred when the workloads change gradually.

   Advantages:

   - Simple to implement and understand.
   - Effective in handling gradual changes in workload.

   Limitations:

   - Can lead to oscillations or over-provisioning, as scaling actions are based on past performance and do not account for future trends.
   - Delayed response due to the time required to observe performance metrics and provision new resources.

2. **Proactive Autoscaling**: Proactive autoscaling aims to anticipate future changes in workload by analysing historical data and forecasting resource requirements. Such proactive systems are preferred when faced with rapid changes in workload in order to avoid frequent SLA violations. Proactive systems rely upon predictive techniques, with three main underlying considerations to each: (1) the workload prediction data source (e.g. past workload history, external domain specific data etc.), (2) the prediction horizon and (3) the prediction algorithm. [153] provide a survey of proactive scaling and predictive algorithm employed by state-of-the-art autoscalers.

   It leverages machine learning techniques, such as time series forecasting and regression models, to predict future workload patterns and proactively scale resources accordingly.

   Advantages:

   - Better resource utilisation by accounting for future trends and avoiding overprovisioning.
   - Can help minimise the impact of workload fluctuations on system performance.

   Limitations:

   - May not be effective in handling sudden, unexpected changes in workload.
   - The accuracy of the autoscaling decisions is highly dependent on the quality and representativeness of the historical data.

Adaptivity

The adaptivity of an autoscaling system refers to the level of ability it has to adapt to changes in the operating environment (e.g. workload characteristics), or even changes in the application itself. Streaming systems by nature operate in a dynamic environment which can change frequently and abruptly, making adaptability an important quality for autoscaling systems.

Non-adaptive systems such as Amazon Autoscaling Service [10] are governed by a predefined model and scaling decisions are made solely based upon the current stat and current input. They also do not allow automatic adjustment of autoscaler settings or parameter values during run-time. This can often mean users must spend substantial time and effort to find suitable settings in prior offline testing.

Self-adaptive systems have fixed core control models, but have the ability to autonomously tune their parameter settings according to real-time feedback and observation regarding

the quality of current scaling actions. Self-adaptivity is often found in control theory based autoscalers, analytical models and machine learning approaches.

Self-adaptive switching systems are those made up of multiple, connected, non-adaptive or self-adaptive controllers with the ability to dynamically switch between them based on their observed, real-time performance. Controllers are substituted in and out in this manner, however there is only ever one controller active and able to make scaling decisions at any point in time.

Scaling Indicators

Autoscalers make scaling decisions based on specific performance metrics that provide insight into the current load and capacity of a system. These can be split into *low-level metrics* and *high-level metrics*, dependent on the level of infrastructure at which they are produced and monitored. Low level metrics are those at the physical/hypervisor level (e.g. CPU utilisation, memory usage), while high-level metrics are those at the application level (e.g. throughput, latency). For some autoscaler systems, both high and low-level metrics are monitored in a hybrid approach.

Resource Estimation Techniques

1. **Threshold-Based Rules**: Rule-based autoscaling is a reactive approach that involves defining a set of rules or policies that dictate scaling actions based on specific conditions [131]. These rules can be user-defined or generated by the system and can incorporate a combination of performance metrics, workload characteristics, and cost constraints. Scaling actions are triggered when the specified conditions are met.

   Advantages:

   - Offers more fine-grained control over scaling decisions.
   - Allows for the incorporation of domain knowledge and cost constraints.

   Limitations:

   - Can be complex to manage, particularly when dealing with large numbers of rules or conflicting policies.
   - May not be as adaptive as other approaches, as rules must be manually updated to account for changes in system behaviour or workload patterns.

2. **Fuzzy Inference Rules:** In the context of stream processing system autoscaling, fuzzy inference rules can be highly advantageous, introducing more granularity and adaptability into the decision-making process than threshold-based rules based on traditional binary logic [131]. Fuzzy inference rules are based on a mathematical methodology that enables systems to deal with uncertainty and imprecision by allowing the use of continuous variables that exist in a state between the absolute "true" or "false" values seen in binary logic systems; i.e. the concept of partial truth is accommodated for.

   The relationships between input and output variables are defined, not in terms of precise mathematical values, but rather in terms of linguistic variables and membership functions. For example, a fuzzy inference rule regarding CPU utilisation might be articulated using imprecise terms such as, *low*, *high* and *very high* which have no definitive true value, but rather reflect the ambiguities of an issue.

   By dealing with linguistic terms and accommodating degrees of truth, fuzzy inference rules allow for a more robust, flexible, and context-sensitive scaling strategy that can better handle the inherent uncertainty and variability in workloads and system performance.

3. **Application Profiling:** Application profiling involves the evaluation and measurement of an application under various configurations and workloads, systematically characterising the system's key behaviours such as memory usage, CPU utilisation or network bandwidth requirements. It plays a pivotal role in predictive autoscaling approaches. By understanding the application's performance characteristics and their relationship with the differing characteristics of incoming workloads, an autoscaler can make more informed scaling decisions and respond more accurately and effectively to workload variations.

   Implementing application profiling in autoscaling, however, is not without challenges. It requires careful design and execution of performance tests, comprehensive monitoring, and sophisticated analysis techniques to accurately model application behaviour. Changes in application characteristics or workloads may also necessitate updates to the underlying application profile.

   Singh et al. [165], Gandhi et al. [60], and Qu et al. [152] employed this approach.

4. **Analytical Modelling and Queuing Theory:** Analytical modelling provides a mathematical foundation to understand and predict system behaviour under varying workload conditions and resource configurations. It involves creating a mathematical

representation of the system that captures key elements such as resources, workloads, and performance metrics. This model can be used to predict system behaviour, evaluate different configuration options, and optimise resource provisioning strategies.

Queuing theory forms a cornerstone of analytical modelling and is particularly pertinent to SPSs, being used to model scenarios where work items arrive at a system and wait in a queue for processing by one or more servers. Key parameters in queuing models include the arrival rate of work items, the service rate of servers, the number of servers, and the queue discipline (i.e. the order in which work items are processed). These parameters allow the calculation of certain performance metrics which can prove crucial in assessing system performance and informing autoscaling decisions (e.g. average queue length, average queue processing times).

It should be noted, however, that the accuracy of analytical modelling and queuing theory relies heavily on the precision of model parameters and assumptions. Inaccurate, oversimplified or miss-specified models can result in suboptimal or even detrimental autoscaling decisions. Various approaches can be found in relevant literature; Gandhi et al. [58], Gandhi et al. [57] and Gergin et al. [67] choose to abstract the whole application/tier/service as a single queue with one server. [5], Jiang et al. [103], [9] and Han et al. [82] instead use a model with a single queue, incorporating multiple, individual servers. Ghanbari et al. [68], Kaur and Chana [113] and Spinner et al. [169] model each server as a separate queue.

5. **Control Theory:** Control theory [2] [81] is based on the concept of manipulating the inputs to a system, to obtain the desired effect on the output of the system, adjusting system resources based on the difference between the actual system performance and a desired reference. This difference is referred to as the "error" and the objective of the controller is to minimise this error over time. For instance, if the system's current throughput is lower than the desired throughput, the controller would take action to increase system resources, thus improving throughput.

   One advantage of control theory is its ability to continuously adjust its resources based on feedback control and the observed error values within the system, thus allowing the system to maintain a desired level of performance even under variable and unpredictable circumstances. The application of control theory in autoscaling faces similar challenges to those faced by analytical models. Systems dynamics may be complex and non-linear, while inaccurate or oversimplified system models can lead to poor controller performance or instability.

   Advantages:

- Adaptive and responsive to changing system conditions.
- Provides fine-grained control over performance objectives.

Limitations:

- May require tuning of control parameters (e.g., PID gains) for optimal performance, which can be complex and time-consuming.
- Sensitive to noise in performance metrics, which can lead to instability or oscillations in resource allocation.

6. **Machine Learning:** Machine learning techniques applied to autoscaling can be broadly categorised into two groups: reinforcement learning (RL) and regression [153]. Reinforcement learning is a subset of machine learning in which an agent learns to interact with its environment by executing actions and observing their effects. The agent is rewarded for actions that are beneficial and penalised for actions that result in a negative outcome. Iteratively, the agent learns to act in a way that maximises its cumulative reward over time.

   The most commonly applied RL algorithm in SPSs autoscaling is the Q-learning algorithm [131]. In this context, the autoscaler acts as the agent with its scaling decisions being the actions. The SPS represents the environment, the system performance metrics provide measurement of the outcome, while the reward system is based on the overall autoscaling system objectives. While able to handle highly complex systems and problem spaces if applied successfully, designing appropriate reward functions is a non-trivial task and agents can require substantial amounts of time and data to learn effective policies.

   Advantages:

   - Capable of learning complex, nonlinear relationships between system states and resource requirements.
   - Can adapt to changing workloads and system conditions without manual intervention.

   Limitations:

   - Requires a significant amount of training data and computational resources to learn effective policies.
   - May take longer to converge to an optimal policy compared to other techniques, especially during the initial learning phase.

- The quality of the learned policy is dependent on the quality of the reward function, which can be challenging to design.

Numerous other machine learning techniques have been employed in autoscaling, including regression models [83], random forests and decision trees [6], and "black-box" style neural networks [161]

7. **Time-Series analysis:** In the context of autoscaling, time-series analysis involves the use of historical data to predict future system behaviour [131]. The data points in a time series are typically collected at regular time intervals and are sequentially correlated. Metrics that autoscalers typically monitor, such as CPU usage, memory usage, network I/O, and job queue lengths, are all time-series data. By analysing these historical data sequences, the autoscaler can identify patterns, trends, and seasonal effects that may inform future system behaviour, thus enabling proactive scaling.

   There are numerous techniques for time-series analysis that could be employed, such as moving averages and smoothing techniques, or more sophisticated approaches such as autoregressive integrated moving average (ARIMA), Seasonal ARIMA (SARIMA) models, Dynamic Time Warping techniques, or various Machine Learning techniques. By leveraging temporal correlations and patterns in system metrics, time-series analysis can help the autoscaler make more informed, proactive, and effective scaling decisions.

   Advantages:

   - Better resource utilisation by accounting for future trends and avoiding over-provisioning.
   - Can help minimise the impact of workload fluctuations on system performance.

   Limitations:

   - May not be effective in handling sudden, unexpected changes in workload.
   - The accuracy of the autoscaling decisions is highly dependent on the quality and representativeness of the historical data.

8. **Hybrid Models:** Hybrid models refer to approaches that combine multiple resource estimation techniques in an attempt to improve the accuracy and robustness of an autoscalers decisions. For instance, a hybrid model may combine analytical modelling with machine learning, or time-series analysis with fuzzy inference rules. Such hybrid combination models can be designed in various different ways, namely: (1)

sequentially, where the output of one method is used as the input for the next, (2) simultaneously, where different methods are used in parallel and their outputs are combined in some way, or (3) hierarchically, where different methods are used at different levels of the decision-making process.

While hybrid models offer increased flexibility and can handle a wider range of scenarios and uncertainties, they can also be more complex and computationally expensive to implement than using a single method.

Advantages:

- More adaptive and resilient to a wider range of workload patterns and system conditions.
- Can provide better overall performance by combining the strengths of different autoscaling techniques.

Limitations:

- Can be more complex to implement and manage compared to single-technique approaches.
- Requires careful tuning and configuration to ensure that the different autoscaling components work effectively together.

Oscillation Mitigation

Oscillations occur when scaling operations are too frequent or the autoscaler has been poorly configured. Two main categories of solutions are commonly employed and have been widely adapted across industry [10]; (1) implementing a cooling-time and (2) the use of dynamic parameters. Implementing a cooling-time involves waiting a fixed minimum amount of time between scaling decisions, while the use of dynamic parameters involves dynamically tuning the triggering thresholds depending on, for example, being in peak-time versus non-peak time [128, 127]. Other dynamic mechanisms can be seen across previous autoscaler literature including a *stability factor* [144, 123] and a *hysteresis parameter* [21].

Scaling Methods

When enacting a scaling decision, depending on the specific system environment, it can be performed either horizontally or vertically, or indeed a combination of both. Vertical scaling involves the provisioning up or down of resources available to existing virtual machines (VMs) (e.g. CPU, memory). Horizontal scaling involves adding or removing more nodes or

VMs to the system. In SPSs, both approachers can be useful, but horizontal scaling often provides more flexibility and resilience, especially for very large data streams.

Service Level Agreement

In the context of DSPSs, and SLA is a contract between a service provider and user that defines the level of service expected, explicitly stating the metrics by which service is to be measured (along with the remedies and penalties should the agreed-upon level of service not be achieved). The measurement metrics usually fall into one of the following categories:

1. **Response Time:** the time it takes a system to react to some request from a user or another system (e.g./ latency of web service or the execution time of a compute task).
2. **Throughput:** the number of tasks or requests a system can process in a given.
3. **Performance:** refers to a variety of metrics that quantify the speed and efficiency of a system, including response time and throughput, but also metrics such as error rates and system down-time.
4. **Cost:** the financial expenses associated with using the cloud resources to run the streaming application.

These goals often trade-off against each other, for instance increasing resources can improve performance but also increase costs. It is important to balance these goals based on the specific requirements and priorities of an application.

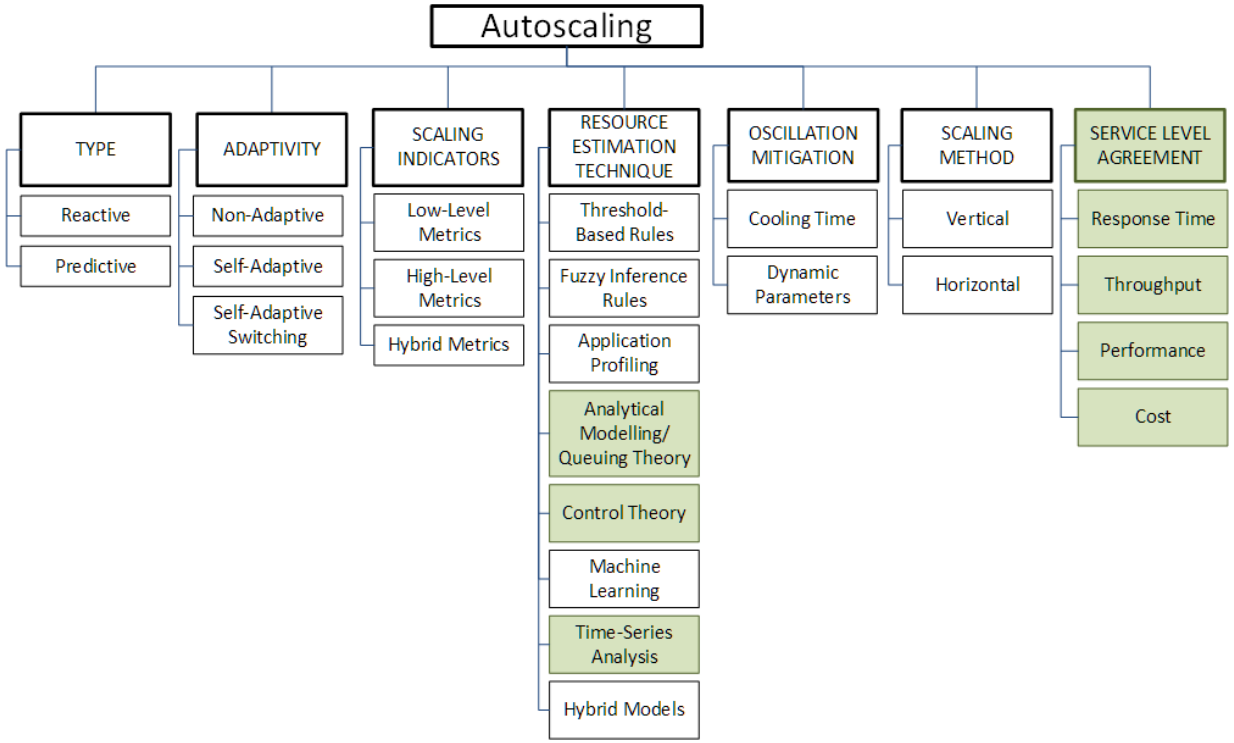


Fig. 2.4 Autoscaler Taxonomy (Adapted from [153]: Additions marked in green)

2.2.3 Applications of Autoscalers in Distributed Stream Processing Systems

1. **Real-Time Analytics**: In real-time analytics, SPSs must handle a continuous flow of data from various sources, such as social media, web applications, or server logs. These systems are required to process and analyse data in real-time, providing valuable insights for decision-making. Autoscalers can be utilised to dynamically adjust resources in response to fluctuating data rates and ensure that the processing systems maintain low latency and high throughput, even during peak loads. For example, an autoscaler can be employed to automatically scale a real-time dashboard application that displays user engagement metrics for a web platform.

2. **Internet of Things (IoT)**: IoT systems involve a large number of interconnected devices that generate massive volumes of data. Stream processing is critical in IoT applications, as it enables real-time processing and analysis of sensor data for applications like smart home automation, industrial monitoring, or healthcare. Autoscalers can help ensure that IoT data processing systems are capable of handling the variable data rates produced by the sensors and maintain the desired level of performance. For instance, an autoscaler can be used in a smart city application that processes and analyses data from traffic sensors to optimise traffic flow and reduce congestion.

3. **Finance**: In the finance domain, SPSs are essential for processing high-frequency data streams, such as stock prices, trading volumes, and news feeds. These systems are responsible for executing real-time analytics, like algorithmic trading or fraud detection, that demand low-latency processing and high throughput. Autoscalers can be used to dynamically allocate resources for these financial systems, ensuring that they can handle sudden surges in data rates, such as during market openings or periods of high volatility. For example, an autoscaler can be employed in a trading platform that automatically scales its resources to maintain low-latency order execution during periods of increased market activity.

4. **Log and Event Processing**: In modern DSPSs, monitoring and analysing log data are crucial for detecting anomalies, diagnosing issues, and ensuring system reliability. SPS are often used to process and aggregate log data from various components in real-time. Autoscalers can be utilised to adjust resources dynamically for log processing systems, ensuring that they can cope with variable log data rates and maintain the desired level of performance. For example, an autoscaler can be deployed in a log processing

Table 2.3 Overview of auto-scaler approaches in state-of-the-art literature (Extension of Table 1 found in [108] - number of autoscalers included increased from 13 to 22, and a more comprehensive legend provided).

Metrics: CPU: CPU; Mem: Memory; BW: Bandwidth; RT: Response Time; TP: Throughput; NS: Network Slack; QS: Queue Size; OR: Observed Rates; Con: Congestion; PT: Pending Tasks; L: Latency; ST; Service Time; IAT: Inter-Arrival Time; BP: Back-pressure; TPR: True Processing Rates; OPR; True Observed Rates; **Technique:** RB: Rule-Based; RL: Reinforcement Learning; TSA: Time-series Analysis; QT; Queuing Theory; CT: Control Theory; TB: Threshold Based; H: Heuristics; DFM; Data-flow Model; **Scaling Action:** S: Speculative; P: Predictive; SO: Single-Operator; MO; Multi-Operator; LS: Load Shedding; **Objective** RT: Response Time; RU: Resource Utilisation; C: Cost; WPA; Workload Prediction Accuracy; TP: Throughput; L: Latency; **Adjustment:** St: Static; D: Dynamic

Autoscaler	Metrics					Technique	Scaling Action		Objective			Adjustment
DM [177]	CPU	Mem	BW	RT	TP	RB	S	MO	RT	RU		D
ACCRS [163]	CPU	Mem	BW			RB	P	MO	RT	RU		St
FQL4KE [99]	Mem	RT	TP			RL	P	MO	RT	C		D
ECNN [94]	CPU	RT				TSA	P	MO	WPA			St
FSL/FQL [11]	RT	TP				RL	S	MO	RT			D
TIRAMOLA [182]	CPU	Mem	BW	RT	TP	RL	P	MO	C	L	TP	St
DC2 [59]	CPU	RT	TP			QT	P	MO	RT			D
EcoWare [16]	CPU	Mem	BW	RT	TP	CT	S	MO	RT			D
Borealis [34]	CPU	NS	QS			RB	LS	MO	L	TP		D
Stream Cloud [78]	CPU	OR				TB	S	SO	TP			D
Seep [33]	CPU					TB	S	SO	L	TP		D
IBM Streams [63]	Con	OR				TB	S	SO	TP			D
FUGU+ [87]	CPU	PT				TB	S	MO	L			D
Nephele [130]	L	ST	IAT			QT	P	MO	L			D
DRS [56]	ST	iat				QT	P	SO	L			D
Stela [195]	OR					TB	S	MO	TP			D
Spark Streaming [147]	PT					TB	S	MO	TP			D
Google Dataflow [4]	CPU	BP				H	S	SO	L	TP		D
Dhalion [54]	QS	POR	BP			RB	S	SO	TP			D
Pravega [151]	OR					RB	S	MO	TP			D
AuTraScale [199]	TPR					RL	P	MO	L	TP		D
DS2 [108]	TPR	TOR				DFM	P	MO	TP			D

system that monitors and analyses logs from a microservices-based application to detect anomalies or performance issues in real-time.

Benefits:

1. **Resource Utilisation Optimisation**: Autoscalers can automatically adjust the number of allocated resources, such as processing instances, according to the fluctuating workloads in the system. This dynamic resource management allows for efficient use of available resources, reducing resource wastage, and minimising operational costs.

2. **Improved Performance and Reliability**: By scaling resources in response to workload variations, autoscalers help maintain the desired level of performance, such as low latency and high throughput. This ensures that the SPS can cope with sudden surges in data rates and continue to provide real-time analytics, even during peak loads.

3. **Reduced Manual Intervention**: Autoscalers can automatically monitor system metrics and make scaling decisions, reducing the need for manual intervention. This not only saves time and effort for system administrators but also minimises the risk of human errors in resource allocation decisions.

4. **Increased Flexibility**: Autoscalers can be configured with various scaling policies and algorithms to suit the specific requirements of a stream processing system. This provides flexibility in choosing the appropriate scaling strategy for different use cases and application domains.

Limitations:

1. **Scaling Latency**: Scaling actions, such as launching new processing instances, might introduce some latency due to the startup time of the new instances or the time required to rebalance the workload. This can cause temporary performance degradation until the new resources are fully operational and integrated into the system.

2. **Algorithm Complexity**: Designing effective autoscaler algorithms can be complex, as they need to consider factors such as workload patterns, system metrics, and scaling costs. Finding the right balance between aggressive and conservative scaling strategies can be challenging, and suboptimal algorithms may lead to under- or over-provisioning of resources.

3. **Configuration Challenges**: Configuring autoscalers can be difficult, as it often requires fine-tuning various parameters, such as scaling thresholds, cooldown periods,

and scaling policies. An incorrect configuration might result in suboptimal scaling decisions or unnecessary oscillations between scaling actions, which can negatively impact system performance and resource utilisation.

4. **Dependency on Infrastructure**: The effectiveness of an autoscaler largely depends on the underlying infrastructure and its capabilities. For example, autoscalers in cloud-based environments may have more flexibility in resource provisioning compared to on-premises setups with limited resources. Additionally, autoscalers might be constrained by infrastructure-specific limitations, such as rate limits on API calls for launching or terminating instances.

2.2.4 Challenges and Research Directions

Current Challenges and Limitations of Autoscalers

1. **Accurate Prediction and Modelling of Workload**: One of the key challenges faced by autoscalers is the accurate prediction and modelling of workload patterns. Stream processing workloads can be highly variable, with sudden spikes or dips in data rates. Developing effective prediction models that can capture these variations and guide scaling decisions is a challenging task. Inaccurate predictions may lead to over- or under-provisioning of resources, resulting in increased costs or performance degradation.

2. **Effective Scaling Policies**: Designing effective scaling policies for autoscalers is a complex problem, as they need to balance responsiveness to workload changes with the potential costs and performance impact of scaling actions. Reactive scaling policies might be too slow to respond to sudden workload surges, while proactive policies could over-allocate resources in anticipation of workload increases that might not materialize. Finding the right balance between these approaches is challenging and depends on the specific characteristics of the SPS and its workload patterns.

3. **Coordination and Consistency**: Autoscalers need to coordinate scaling actions across distributed processing instances, ensuring that the entire system maintains consistent performance and resource utilisation. This can be challenging in large-scale deployments, where communication overhead and potential delays in propagating scaling decisions can impact the effectiveness of the autoscaler. Furthermore, maintaining consistency in stateful processing systems, where intermediate state information must be shared or migrated between instances, adds additional complexity to the coordination process.

4. **Impact of Scaling Actions on System Performance**: Scaling actions, such as launching new instances or redistributing workloads, can have a temporary impact on system performance due to startup times, rebalancing overhead, or data movement. Autoscalers need to account for these performance impacts when making scaling decisions and minimise their negative effects on the overall system performance.

5. **Integration with Other System Components**: Autoscalers need to interact with various components of the stream processing system, such as data sources, processing engines, and storage systems. Seamless integration with these components and their respective APIs is essential for efficient autoscaling. However, differences in API design, data formats, or communication protocols can pose challenges to the integration process and impact the effectiveness of the autoscaler.

6. **Adapting to Changes in System and Workload Characteristics**: SPSs and their workloads can evolve over time, with changes in data patterns, processing requirements, or infrastructure capabilities. Autoscalers need to be adaptive and robust to such changes, updating their prediction models, scaling policies, and resource management strategies accordingly. This requires ongoing monitoring and analysis of system performance and workload patterns to ensure optimal autoscaling decisions. For the sake of clarity, Point 1 above (**Accurate Prediction and Modelling of Workload**), focuses on the specific challenge of accurate short-term workload prediction and modelling, while here we take a broader view of adapting to long-term changes in system and workload characteristics. The former highlights the consequences of inaccurate predictions, such as resource misallocation and performance issues, while we now emphasise the need for ongoing monitoring, analysis, and adaptation of autoscaling components.

Ongoing Research in Autoscalers

1. **Scalability**: As DSPSs continue to grow in size and complexity, there is a need for autoscalers that can effectively manage resources and maintain performance at scale. Research in this area is focused on designing and implementing autoscaling algorithms that can efficiently manage large-scale deployments and adapt to changes in system size, resource availability, and workload patterns. This includes exploring novel approaches to workload prediction, resource allocation, and load balancing, as well as investigating the use of machine learning techniques to optimise autoscaling decisions.

2. **Fault Tolerance** [100]: Ensuring fault tolerance in autoscalers is essential for maintaining the reliability and robustness of distributed stream processing systems. Ongoing research is focused on designing autoscalers that can detect and recover from failures, such as processing node crashes, network partitions, or software bugs. This includes developing mechanisms for monitoring the health of processing nodes and detecting failures, as well as designing recovery strategies that can quickly restore system performance and resource utilisation. Researchers are also investigating the use of replication, checkpointing, and other fault-tolerance techniques to ensure the durability of intermediate processing state during scaling actions.

3. **Real-time Analytics** [36]: As the demand for real-time analytics and decision-making increases, there is a need for autoscalers that can effectively support these use cases in distributed stream processing systems. Research in this area is focused on developing autoscalers that can maintain low-latency processing while adapting to changes in workload patterns and resource availability. This includes exploring techniques for minimising the impact of scaling actions on system performance, such as incremental scaling, data partitioning, and load balancing. Researchers are also investigating the use of application-specific knowledge, such as query plans or data access patterns, to optimise autoscaling decisions and improve the efficiency of real-time analytics workloads.

4. **Machine Learning and AI-driven Autoscaling** [176]: To improve the accuracy of workload prediction and resource allocation, researchers are exploring the use of machine learning and artificial intelligence techniques in autoscalers. These approaches can help learn the patterns of the streaming workloads and adapt the autoscaling decisions accordingly. Reinforcement learning, for example, is being investigated as a method for training autoscalers to make optimal decisions based on system state and workload patterns.

5. **Heterogeneous Resource Management** [154]: As DSPSs increasingly involve diverse processing nodes and resource requirements, research is focused on developing autoscalers that can effectively manage these heterogeneous resources. This includes designing algorithms for allocating and scheduling resources based on the specific needs of different processing tasks, as well as investigating techniques for dynamically adjusting resource allocations in response to changes in workload patterns or processing requirements.

6. **Autoscaler Evaluation and Benchmarking** [172]: To assess the effectiveness of different autoscaling algorithms and strategies, researchers are working on developing evaluation frameworks and benchmarking tools for autoscalers in distributed stream processing systems. These tools can help compare the performance, resource utilisation, and cost-efficiency of different autoscalers, as well as identify areas for improvement and guide future research in this area.

2.3 Workload Generation and Modelling

Computer system workload modelling is a process that attempts to create a simple and general model, capable of generating synthetic workloads. These workloads should, in turn, be suitable for running simulations representative of those workloads a computer system may experience in a real-world setting. Creating workloads that are similar to those observed in practice forms the necessary foundation to carry out any type of meaningful performance evaluation efforts.

Performance evaluation is a vital step in the development and deployment of computer systems. It allows researchers and engineers to measure the effectiveness and efficiency of different designs and configurations, and to make informed decisions about how to improve performance. When building new systems, performance evaluation methods are implemented in order to compare different design options. This can include comparing different algorithms, architectures, and hardware configurations. For example, a researcher may compare the performance of a system that uses a distributed architecture to one that uses a centralised architecture, in order to determine which design is better suited for a particular application.

Similar evaluation approaches are also used to adjust the settings of existing systems. This can include tuning parameters such as CPU frequency, cache size, and memory configuration in order to optimise performance. This is especially important in large-scale systems where small changes in configuration can result in significant performance gains. For production use, performance evaluation can be used to determine the necessary capacity required for the system to handle the expected workload. This can include estimating the number of servers or other resources that will be needed to meet the expected demand, as well as determining how the system will handle unexpected spikes in traffic.

The ability to carry out meaningful performance evaluation plays a critical role in the development and deployment of computer systems in general, stream processing systems being no exception. It generates the necessary insight and provides information upon which

informed decisions can be made, thus allowing for the creation of efficient and effective systems that can meet the desired objectives, and minimise resource wastage.

There are three main factors that affect the performance of a stream processing system: (1) the design of the system, (2) the implementation of the system, and (3) the workload to which the system is exposed.

The design of the system refers to the architecture and overall structure of the system, the implementation of the system refers to the way the system is built and configured, and the workload refers to the tasks and requests that the system is expected to handle. All three of these factors must be taken into consideration when evaluating the performance of a computer system.

When evaluating complete systems, they may perform well for one workload, but not for another. There are many instances in which specific workload features have a significant impact on performance. However, it's important to note that not all workload features have the same effect. Sometimes, it is a single workload feature that has the greatest impact on performance. The challenge is that it is not always clear in advance which feature is the most critical, and even if it appears obvious, there is a chance of being mistaken. This makes the ability to test streaming systems using realistic workloads all the more important, hence the motivation to generate high-quality workload models that reliably capture all known workload features, including those that may be predicted to lack overall significance.

2.3.1 Background

Workload modelling plays a critical role in the design, deployment, and management of stream processing systems. In the context of stream processing, workload characterisation is the process of identifying and quantifying the key features of a stream processing workload, such as data arrival rates, data volume, processing complexity, and resource requirements. Workload characterisation provides a basis for developing workload models, which can be used to analyse and predict system behaviour, guide resource allocation decisions, and evaluate the performance and scalability of stream processing systems.

One of the main challenges in workload modelling for stream processing is dealing with the dynamic and potentially unpredictable nature of data streams. Data streams are continuous and unbounded sequences of data elements generated by sources such as sensors, log files, or social media feeds. There is a distinction to be made between fixed workloads and streaming workloads, of which we will focus purely on the latter:

- Fixed workloads are ones where the amount of data and the number of tasks to be processed are known in advance. The system is designed to handle a specific set of

tasks and the workload remains constant. An example of a fixed workload would be a batch processing system that processes a fixed number of records in a database.

- Streaming workloads are ones where the data is generated and processed in real-time, and the number of tasks and amount of data to be processed is not known in advance. The system is designed to handle a variable set of tasks and the workload is not constant. An example of a streaming workload would be a system that processes tweets or stock trading data in real-time.

2.3.2 Workload modelling methodologies

There are two prevalent approaches for using a recorded workload to perform analysis and allow a system design to be evaluated:

1. Using the traced workload as-is to run a simulation.
2. Constructing a model based on the trace and using the model for either analysis or simulation.

There are several advantages to using workload models over workload traces when evaluating the performance of a stream processing system. These include:

- Repeatability: Workload models can be used to repeat the same simulation multiple times, under statistically similar, but not identical conditions, which allows for more consistent and accurate performance measurements. For instance, different random number generator seeds can be used to generate multiple workloads. This would allow the computation of confidence intervals, whereas using a workload trace or log, only a single data point is available.
- Flexibility: Workload models allow for more flexibility in simulating different scenarios and workloads. This is particularly useful for evaluating the performance of a system under different conditions, such as varying data rates or different types of data.
- Control & Isolation: Direct measurement of the system's sensitivity to different parameters can be achieved by changing the values of model parameters individually, in order to understand the impact of each one, while maintaining the other parameters constant. Workload traces, as a general rule, can not be adapted for use in this manner.
- Simplicity & Generalisation: Workload models can simplify the process of evaluating the performance of a SPS by abstracting away the complexities of real-world workloads.

This also lends itself well to providing a generalisation and avoiding overfitting to a specific dataset.

Workload models can be broadly classified into two categories: descriptive and generative models.

Descriptive Models

Descriptive models are used to describe the characteristics and patterns of the workload as it is observed. These models capture the static properties of the workload, such as the distribution of requests, the frequency of requests, and the response time of requests. Descriptive models are useful for identifying patterns in the workload and for understanding how the workload behaves under different conditions. They do not attempt to understand the underlying processes that generate the workload, but simply provide a snapshot of the workload at a specific point in time.

Descriptive modelling typically involves creating a statistical summary of an observed workload. This summary includes all attributes of the workload, such as computation, memory usage, I/O behaviour, and communication. The goal of this summarisation is to capture the main characteristics of the workload. The longer the observation period, the more comprehensive the summary. For example, an entire year's workload can be summarised by analysing the records of all the jobs that ran on a given system during that year, and fitting distributions to the observed values of the different parameters.

A synthetic workload can then be generated according to the model by sampling from the distributions that constitute the model. Additionally, the model can be used directly to parameterise a mathematical analysis. This means that the model can be used as input for a mathematical analysis, to evaluate and predict the performance of the system under different workload conditions.

Generative Models

Generative models, on the other hand, focus on reproducing the actual processes that lead to the generation of the workload. This means that generative models attempt to emulate the underlying mechanisms that result in the workload, rather than simply describing it. These models are built based on the understanding of the system, workload, and the user behaviour. They are useful for simulating the workload and predicting how the workload will behave under different conditions. They can also be used to test the performance of a system under different workloads and to identify potential bottlenecks in the system.

One of the key advantages of using a generative approach for workload modelling is that it allows for manipulation of the workload. With a descriptive model, it is difficult to change the workload conditions as part of an evaluation. However, with a generative model, it is possible to modify the process that generates the workload in order to achieve the desired conditions, and the resulting workload will reflect these changes. For example, by modelling different types of file editing sessions, different file-size distributions can be generated that match the specific session being modelled. This ability to manipulate the workload through the generative approach makes it a powerful tool for evaluating and understanding the behaviour of a system under different workload conditions.

Descriptive versus Generative Models

In summary, the main differences between Descriptive and Generative workload modelling approaches are:

1. Purpose: Descriptive models summarise observed characteristics, while generative models reproduce workload generation processes.

2. Approach: Descriptive models provide a statistical summary, while generative models emulate underlying mechanisms.

3. Workload Understanding: Descriptive models do not consider workload generation processes, while generative models are built based on system and user behaviour understanding.

4. Manipulation: Descriptive models do not allow for workload manipulation, while generative models enable modification of the generation process.

5. Applications: Descriptive models are used for generating synthetic workloads and parametrising analyses, while generative models are used for simulating, predicting, and testing system performance.

2.3.3 Workload modelling in Stream Processing Systems

Overview of how workload modelling is applied to stream processing systems

The application of workload modelling to SPSs requires understanding and representing the characteristics of the workload that a system must be expected to handle, including the arrival patterns of the data streams, the nature and complexity of the processing required,

the interactions between different components of the system, and the system's requirements in terms of performance, availability, and reliability.

Workload modelling plays an essential role in designing, deploying, and managing SPSs. By capturing the characteristics of data streams, processing complexity, and resource requirements, workload models provide valuable insights into system behaviour and performance and can enable informed decisions that improve performance, efficiency, and reliability, while also providing a foundation for automation and intelligent adaptation to changing conditions.

Benefits:

1. **Resource optimisation:** Workload modelling helps to optimise resource allocation by guiding decisions about CPU, memory, and network capacity assignments. By understanding the processing requirements and resource demands, system designers can allocate resources more effectively, leading to better performance and cost-efficiency.

2. **Performance prediction and evaluation:** Workload models enable the prediction and evaluation of system performance under different conditions, such as varying data arrival rates, processing complexity, and resource constraints. This allows for proactive performance tuning and helps to identify potential bottlenecks and scalability issues.

3. **Autoscaling support:** Workload modelling is an essential component of autoscaling algorithms, which dynamically adjust resource allocations and processing parallelism in response to changes in workload patterns or system performance. By providing accurate workload models, autoscalers can make better resource allocation decisions and maintain performance and resource utilisation targets.

4. **Improved system design:** Workload modelling provides insights into the processing complexity and resource requirements of stream processing tasks, which can inform system design decisions. This can lead to the development of more efficient and scalable stream processing systems.

Limitations:

1. **Dynamic and unpredictable data streams:** Workload modelling techniques, when applied in a synthetic environment, often struggle to accurately mimic the intricate and unpredictable nature of real-world data streams. These real-world workloads exhibit highly dynamic and complex behaviour, with significant variations in data arrival rates, volume, and distribution over time. Capturing and representing these complex dynamics in synthetic workload models poses a significant challenge. Synthetic workload models often rely on simplified assumptions and abstractions, which

may not adequately represent the full spectrum of behaviors observed in real-world scenarios. This limitation can lead to inaccuracies in the generated workloads, resulting in a mismatch between the modelled behaviour and the actual characteristics of the real-world data streams. Consequently, the insights and decisions derived from synthetic workload models, particularly those related to resource allocation and system optimisation, may not accurately reflect the performance and resource requirements of the system when subjected to real-world workloads. This discrepancy can result in suboptimal resource provisioning, inefficient utilisation of system resources, and potential performance degradation when the system is deployed in a production environment.

2. **modelling complex processing tasks:** SPSs often involve complex processing tasks, such as windowing, aggregation, and pattern detection. Accurately modelling the computational requirements and resource demands of these tasks can be challenging, particularly when considering the interactions between multiple tasks and operators within a processing pipeline.

3. **Scalability of workload models:** As SPSs grow in scale and complexity, the corresponding workload models may also become more complex and resource-intensive. Developing scalable workload models that can effectively represent large-scale systems while maintaining reasonable computational overhead is an ongoing challenge.

4. **Adapting to evolving workloads:** Workloads in SPSs can evolve over time, as new data sources are added, processing requirements change, or system configurations are updated. Workload models must be adaptable to these changes to maintain their accuracy and relevance, which may require continuous monitoring, model updates, and adaptive modelling techniques.

5. **Integration with autoscaling algorithms:** Workload models are a critical component of autoscaling algorithms, which dynamically adjust resource allocations and processing parallelism in response to changes in workload patterns or system performance. Ensuring that workload models provide accurate and timely input to autoscaling algorithms, while avoiding excessive overhead, is a challenge that needs to be addressed.

6. **Model validation and evaluation:** Validating the accuracy and effectiveness of workload models can be difficult, as it often requires comparing model predictions with actual system performance under a range of conditions. Developing appropriate validation and evaluation methodologies for workload models in SPSs is an important area of ongoing research.

2.3.4 Challenges and Research Directions

Ongoing research in this area aims to address the challenges and limitations of current workload modelling techniques, while exploring new methods and tools for more accurate and efficient modelling. Some key ongoing research directions in workload modelling for SPSs are:

1. **Machine learning-based modelling approaches** [134]: Machine learning techniques, such as neural networks, clustering, and regression models, are being investigated for their potential to improve workload modelling accuracy and adaptability in stream processing systems. These methods can potentially capture complex relationships between input data characteristics, processing requirements, and resource usage more effectively than traditional model-based approaches.

2. **Online workload modelling and adaptation** [43]: As data stream characteristics and processing requirements evolve over time, workload models must adapt to maintain their accuracy and relevance. Research in online workload modelling focuses on developing methods for continuously updating and refining models based on observed system behaviour and performance, ensuring that models remain responsive to changing workloads and system configurations.

3. **modelling uncertainty and variability** [17]: Workload models should account for the inherent uncertainty and variability in data stream characteristics, processing requirements, and system performance. Probabilistic modelling techniques, such as Bayesian networks and Markov models, are being explored for their ability to represent and reason about uncertainty in workload models, enabling more robust and reliable resource allocation decisions.

4. **Scalable modelling techniques** [69]: As SPSs grow in scale and complexity, the corresponding workload models must also scale to accommodate larger numbers of processing tasks, operators, and resources. Research in scalable modelling techniques aims to develop methods for efficient and accurate workload representation, even in large-scale and highly parallel systems.

5. **Model validation and evaluation** [105]: Assessing the accuracy and effectiveness of workload models is critical for ensuring their reliability and utility in practice. Ongoing research in model validation and evaluation seeks to develop robust methodologies for comparing model predictions with observed system behaviour, identifying sources of modelling error, and refining models to improve their accuracy and reliability.

6. **Integration with autoscaling and resource management algorithms:** Workload models play a key role in informing autoscaling and resource management decisions in stream processing systems. Research in this area focuses on developing techniques for more effective integration of workload models with resource management algorithms, enabling more dynamic and responsive resource allocation and system performance optimisation.

2.4 Benchmarking and Evaluation

Benchmarking refers to the process of measuring and comparing the performance of a system, application, or component against a set of predefined metrics, standards, or reference implementations. Benchmarking and evaluating stream processing systems involve assessing the performance, scalability, fault tolerance, and other essential characteristics of these systems under various workloads, configurations, and operating conditions. This process aims to compare different stream processing solutions, identify their strengths and weaknesses, guiding the selection of the most suitable system for a given use case, and informing the development of future improvements and optimisations.

SPSs are designed to handle continuous, high-volume, and time-sensitive data streams, such as those generated by IoT devices, social media platforms, or financial transactions. Therefore, the benchmarking and evaluation of these systems must take into consideration the unique requirements and challenges posed by real-time data processing.

Benchmarking and evaluating SPSs is of paramount importance for various reasons, including:

1. **Informed decision-making:** With a plethora of SPSs available, it can be challenging for organisations to choose the most appropriate one for their specific needs. By conducting comprehensive benchmarks and evaluations, organisations can gain insights into each system's performance, scalability, fault tolerance, and other relevant characteristics. This information allows them to make more informed decisions when selecting the best-suited SPS for their particular use cases and requirements.

2. **Continuous improvement:** Benchmarking and evaluating SPSs enable developers and researchers to identify areas where improvements can be made. By measuring performance and other key characteristics, they can detect bottlenecks, inefficiencies, and limitations in the system, paving the way for further optimisations and enhancements. Continuous evaluation and iteration ensure that SPSs evolve and adapt to the ever-changing requirements of real-time data processing.

3. **Identification of best practices:** Through the benchmarking and evaluation process, researchers and practitioners can identify best practices for designing, implementing, and deploying stream processing systems. By comparing various systems and their performance under different conditions, they can discern the most effective strategies, techniques, and approaches for building robust, high-performing, and scalable stream processing solutions.

4. **System optimisation:** Benchmarking and evaluating SPSs help organisations optimise their existing systems for better performance, resource utilisation, and cost efficiency. By comparing their current system with alternative solutions or configurations, they can determine the most effective ways to enhance their system's capabilities, reduce resource consumption, and minimise costs. This optimisation process may involve adjusting the system's settings, modifying its architecture, or even switching to a different stream processing solution altogether.

5. **Facilitating research and development:** The process of benchmarking and evaluating SPSs plays a vital role in driving research and development in this domain. By identifying current challenges, limitations, and research gaps, the evaluation process can inspire new research directions and innovative solutions. It also fosters a healthy competition among developers and researchers, motivating them to advance the state of the art in stream processing technology.

6. **Industry standardisation:** Benchmarking and evaluating SPSs contribute to the standardisation of performance metrics, methodologies, and best practices in the industry. Standardised benchmarks enable fair and meaningful comparisons among different systems, promoting transparency and collaboration among developers, researchers, and users. These standards also facilitate the adoption of SPSs in various application domains, as organisations can rely on well-established benchmarks and evaluation results to make informed decisions and ensure the success of their real-time data processing initiatives.

2.4.1 Background

The historical development of benchmarking and evaluating stream processing systems can be traced back to the early days of stream processing research and development. Over the years, various benchmarking methodologies and tools have emerged to address the specific requirements and challenges associated with stream processing systems.

1. **Early stream processing systems:** The first wave of stream processing systems, such as TelegraphCQ [35], Aurora [34], and STREAM [174], emerged in the early 2000s. These systems aimed to address the increasing need for real-time data processing and analysis. However, at this stage, there was limited focus on benchmarking and evaluation, as the emphasis was on developing the foundational concepts and architectures for stream processing.

2. **Introduction of benchmarking methodologies:** As SPSs evolved and gained traction, researchers began to recognise the need for standardised benchmarks to compare and evaluate these systems. In the mid-2000s, the Linear Road Benchmark [179] was proposed as one of the first attempts to provide a comprehensive benchmark for evaluating stream processing systems. Linear Road Benchmark simulated a traffic management scenario with varying data rates and complexity to test the performance, scalability, and fault tolerance of stream processing systems.

3. **Adoption of data processing benchmarks:** In the late 2000s and early 2010s, researchers started to adapt existing data processing benchmarks, such as TPC-H and TPC-DS [150], for the evaluation of stream processing systems. While these benchmarks were originally designed for traditional data processing systems, they were extended to accommodate the unique characteristics of stream processing, including continuous queries, windowed operations, and real-time analytics.

4. **Development of domain-specific benchmarks:** As SPSs found applications in various domains, such as social media analytics, IoT, and finance, domain-specific benchmarks emerged to address the unique requirements and challenges of these applications [150, 80]. For example, the Yahoo! Streaming Benchmark was developed to evaluate the performance of SPSs in the context of real-time advertisement analytics, while the DEBS Grand Challenge series focused on various application scenarios, such as smart grid management and social network analysis.

5. **Emergence of micro-benchmarking and custom benchmarks:** In the mid-2010s, researchers began to explore micro-benchmarking techniques to evaluate the performance of individual components and features of stream processing systems [41, 74]. These micro-benchmarks enabled fine-grained performance analysis and optimisation. Moreover, custom benchmarks were developed by organisations to address their specific requirements and use cases, allowing them to evaluate and optimise SPSs for their unique needs.

6. **Recent developments:** In recent years, there has been a growing interest in the development of comprehensive and extensible benchmarking frameworks for stream processing systems. Tools such as the Data Accelerator for Streaming (DAS) and the Benchmarking-as-a-Service (BaaS) framework have emerged, offering a more flexible and modular approach to benchmarking and evaluation [41, 120, 23]. These frameworks allow users to design and execute custom benchmarks, tailoring the evaluation process to their specific requirements and use cases.

2.4.2 Benchmarking and evaluation methodologies

Overview of different methodologies for benchmarking and evaluating stream processing systems

A variety of methodologies and tools have been developed to facilitate the benchmarking process. The categories listed below are not mutually exclusive and there is in part, of course, overlap between them.

1. **Synthetic Benchmarks:** Synthetic benchmarks involve the creation of artificial datasets and workloads to test the performance and capabilities of stream processing systems. These combine closely with, and relate closely to, synthetic workloads. These benchmarks use simulated data generators and predefined query sets to assess system performance under different conditions, such as varying data rates, query complexity, and system configurations.

2. **Adapted Data Processing Benchmarks:** These benchmarks modify existing data processing benchmarks like TPC-H and TPC-DS to evaluate stream processing systems. They extend traditional benchmarks to accommodate the unique characteristics of stream processing, including continuous queries, windowed operations, and real-time analytics.

3. **Domain-specific Benchmarks:** Domain-specific benchmarks are designed to evaluate SPSs in specific application domains, such as social media analytics, IoT, finance, or healthcare. They use real-world datasets and scenarios to test the system's ability to process and analyse data streams effectively within the target domain.

Comparison of Different Methodologies

Benchmarking methodologies for SPSs can be compared based on factors such as accuracy, scalability, and real-time performance. In this section, we provide a comparison of the

different methodologies discussed earlier, highlighting their strengths and weaknesses in these aspects.

Synthetic Benchmarks:

- **Accuracy:** Synthetic benchmarks can provide a controlled environment for evaluating system performance, but their accuracy is limited by the degree to which the generated data and workloads represent real-world scenarios. The artificial nature of synthetic benchmarks may not fully capture the complexity and variability of real-world data streams and processing tasks.
- **Scalability:** These benchmarks can be easily scaled to test system performance under varying data rates and processing loads, providing valuable insights into system capacity and resilience.
- **Real-time Performance:** Synthetic benchmarks can measure real-time performance effectively, as they can simulate high-velocity data streams and evaluate the system's ability to handle real-time data processing requirements.

Adapted Data Processing Benchmarks:

- **Accuracy:** These benchmarks provide a more accurate representation of real-world data processing workloads compared to synthetic benchmarks. However, they may not fully capture the unique characteristics and challenges of stream processing, limiting their accuracy in evaluating stream processing systems.
- **Scalability:** Adapted benchmarks can be scaled to test system performance across different data rates and processing loads, although their scalability might be constrained by the original benchmark's design.
- **Real-time Performance:** While these benchmarks can evaluate real-time performance to some extent, they might not accurately assess stream processing systems' capabilities, as they are not specifically designed for stream processing use cases.

Domain-specific Benchmarks:

- **Accuracy:** Domain-specific benchmarks offer the highest accuracy in evaluating stream processing systems, as they use real-world datasets and scenarios specific to the target domain. However, their applicability may be limited to the domain they were designed for.

- **Scalability:** These benchmarks can be scaled to accommodate different data rates and processing loads within the domain, providing valuable insights into system performance and capacity.
- **Real-time Performance:** As these benchmarks are tailored to specific application domains, they can accurately evaluate the real-time performance of SPSs in the context of the target domain.

Micro-benchmarks:

- **Accuracy:** Micro-benchmarks provide a fine-grained evaluation of individual system components, allowing for accurate identification and analysis of bottlenecks or inefficiencies. However, they might not capture the overall performance and interactions between different components in a complete system.
- **Scalability:** These benchmarks can be scaled to evaluate the performance of individual components under varying conditions. However, their scalability is limited to the component being tested.
- **Real-time Performance:** Micro-benchmarks can assess the real-time performance of individual components, but they may not provide a complete picture of the system's real-time capabilities.

Custom Benchmarks:

- **Accuracy:** Custom benchmarks can provide highly accurate evaluations of stream processing systems, as they are designed based on the organisation's specific requirements and use cases. However, their accuracy and applicability might be limited to the organisation and scenarios they were designed for, making them less useful for comparing different systems or generalising their results.
- **Scalability:** These benchmarks can be scaled to test system performance under varying data rates and processing loads specific to the organisation's requirements. This ensures that the scalability assessment is relevant to the organisation's real-world needs.
- **Real-time Performance:** Custom benchmarks can accurately evaluate the real-time performance of stream processing systems, as they are tailored to the organisation's specific real-time data processing requirements. However, their focus on the organisation's unique needs might limit their usefulness in comparing different systems or assessing their performance in other contexts.

2.4.3 Benchmarking and evaluation tools

Tools for Benchmarking Stream Processing Systems

Data Generators: Data generators, such as Databricks Labs Data Generator [1] and StreamGen [77], are tools used to create synthetic data streams for benchmarking purposes. These tools allow users to generate datasets with specific characteristics, such as data distribution, volume, and velocity, to evaluate the performance of SPSs under various conditions.

Benchmark Suites: Benchmark suites, such as the Linear Road Benchmark [12], Yahoo! Streaming Benchmark [37], and DEBS Grand Challenge series [79], provide standardised query sets and performance metrics for evaluating stream processing systems. They enable comparisons across different systems and configurations, helping organisations identify the best solution for their needs.

Benchmarking Frameworks: Benchmarking frameworks, like the Data Accelerator for Streaming (DAS) [122] and Benchmarking-as-a-Service (BaaS) [121] framework, provide a modular and flexible approach to benchmarking and evaluation. These frameworks allow users to design and execute custom benchmarks, tailoring the evaluation process to their specific requirements and use cases. They often offer various tools and features for generating datasets, defining workloads, and analysing performance metrics.

Benefits and limitations of using benchmarking and evaluation

Benefits

1. **Performance Optimisation**: Benchmarking and evaluation allow for the identification of performance bottlenecks, leading to targeted improvements in the SPSs for each specific domain.

2. **Comparability**: By comparing different SPSs in various domains, organisations can make informed decisions about the most suitable system for their use case, considering factors such as processing speed, fault tolerance, and scalability.

3. **Cost Efficiency**: Benchmarking can help organisations understand the cost-performance trade-offs of different stream processing systems, enabling them to choose solutions that deliver the best value for their investment.

4. **Tailored Solutions**: Evaluating SPSs in different domains can lead to the development of more specialised solutions that address the unique requirements and challenges of each domain, resulting in more effective and efficient systems.

5. **System Evolution**: Benchmarking and evaluation can provide valuable feedback to system developers, driving continuous improvement and innovation in the field of stream processing.

Limitations

1. **Benchmark Suitability**: Not all benchmarks or evaluation methods are equally suitable for every domain. Some benchmarks may not accurately reflect the specific characteristics or requirements of certain domains, leading to suboptimal comparisons or misleading results.

2. **Implementation Complexity**: Designing and implementing benchmarking experiments for SPSs can be complex, especially in domains with unique requirements, large-scale data processing needs, or complex data structures.

3. **Data Sensitivity**: Some domains, such as healthcare or finance, involve sensitive data that may be subject to strict privacy and security regulations. This can create challenges for benchmarking and evaluation, as it may not be possible to use real-world data or share results publicly.

4. **Benchmark Overemphasis**: Focusing too much on benchmark performance can lead to an overemphasis on optimising for specific metrics, potentially neglecting other important aspects of stream processing systems, such as ease of use, flexibility, or integration capabilities.

5. **Evolving Workloads**: In many domains, workloads and data streams can evolve rapidly, making it difficult to develop benchmarks that remain relevant over time. Regularly updating benchmarks to reflect these changes can be resource-intensive and time-consuming.

2.4.4 Challenges and Research Directions

Benchmarking and evaluating SPSs is an active area of research, with a focus on developing more accurate, reliable, and comprehensive methodologies to assess the performance, scalability, and fault tolerance of these systems. Several key topics of ongoing research and trends in this field include:

1. **Domain-specific benchmarks** [22]: Researchers are developing new benchmarks specifically tailored to different application domains, such as IoT, finance, social media, and healthcare. These domain-specific benchmarks aim to provide more accurate

and relevant performance assessments by incorporating the unique characteristics, data structures, and processing requirements of each domain.

2. **Multidimensional evaluation** [175]: There is a growing interest in adopting multi-dimensional evaluation techniques that consider various aspects of SPSs beyond performance, such as resource consumption, ease of use, and flexibility. This holistic approach to evaluation enables more comprehensive comparisons and better-informed decision-making when choosing a SPS for a particular use case.

3. **Adaptive and self-tuning benchmarks** [88]: As stream processing workloads evolve over time, static benchmarks may become less relevant or informative. Researchers are exploring adaptive and self-tuning benchmarking methodologies that can automatically adjust to changes in data streams and workload characteristics. This approach aims to provide more accurate and up-to-date performance assessments while reducing the manual effort required to maintain and update benchmarks.

4. **Real-time and continuous benchmarking** [22]: Traditional benchmarking approaches often involve running a set of predefined experiments and comparing the results. However, this approach may not be suitable for stream processing systems, which are designed to operate in real-time and continuously adapt to changing conditions. Researchers are investigating real-time and continuous benchmarking techniques that can evaluate the performance of SPSs as they operate, providing more timely and actionable feedback.

5. **Cross-platform comparisons**: With the increasing number of available stream processing platforms, there is a need for robust benchmarking methodologies that can facilitate cross-platform comparisons. Researchers are working on developing benchmarks and evaluation techniques that can be applied across different stream processing systems, taking into account variations in programming models, data processing semantics, and system architectures.

6. **Integration of machine learning and artificial intelligence** [61]: Machine learning and artificial intelligence techniques have the potential to significantly enhance the process of benchmarking and evaluating stream processing systems. Researchers are exploring the use of machine learning algorithms to automate the generation of synthetic workloads, predict system performance under various conditions, and optimise the configuration of SPSs for specific workloads or use cases.

Chapter 3

Methodology

3.1 Introduction

Our methodological approach and paradigm follows the over-arching principles and practices outlined in the Design Science Research Methodology (DSRM) approach [146]. The Design Science (DS) process includes six steps:

1. Problem Identification and Motivation: One must know both what specific problem is being tackled and why it is worth spending the effort to find a solution.
2. Definition of the Objectives for a Solution: Clearly define the desired outcomes and determine what makes a solution better than existing ones.
3. Design and Development: Create a "research artefact" that embeds a research contribution in its design.
4. Demonstration: Demonstrate that the artefact can be used to solve instances of the problem through experimentation, simulation, or another appropriate activity.
5. Evaluation: Test the effectiveness of the artefact in offering a solution to the problem.
6. Communication: Draw conclusions, communicate the problem and its importance, and present the solution's utility, novelty, and scientific rigour.

Following a sound methodology provides a structured approach to designing and conducting research efforts, helping to ensure a systematic and rigorous process. In the computer science domain, where technological progress and general rate of change can be rapid, this forms a vital foundational framework for developing new and innovative solutions to complex problems. Moreover, a good research methodology guarantees replicability and

verifiability of results, allowing for the building upon existing research and for validating the findings of a study. It has enabled me to communicate my findings effectively to the broader research community, an important aspect for advancing knowledge and for facilitating the use of research findings to inform practical applications. An example of this can be seen in the research paper published at DEBS 2023; a replication package containing 40,000 workloads traces collected as part of the work, all code and workloads necessary to reproduce our findings was created and made available for download via an open-source, public GitHub repository.

3.2 Research Systems and Requirements

Our overarching research questions all concern themselves with (1) DSPSs and aspects related to their performance and behaviour when (2) combined with an auto-scaling system, and (3) faced with dynamic, unpredictable incoming workloads.

3.2.1 DS2 Autoscaler

We look to utilise DS2 [108] as the base experimental autoscaler on which to carry out the required experiments. DS2 is an automatic scaling controller for distributed streaming dataflows. It operates on a performance model that assumes operator instances repeatedly perform three activities in sequence: deserialisation, processing, and serialisation. When an operator instance is scheduled for execution, it pulls records from its input, deserialises them, applies its processing logic, and serialises the results, which are then pushed to the output. This model fits all types of operators in most modern streaming dataflow systems, including Heron [118], Flink [29], and Timely [28].

DS2 obtains rate measurements of each operator by lightweight instrumentation, which is already present in many streaming systems. It measures the useful time of an operator's timeline and determines the true rate of each operator. Based on these measurements, DS2 infers the required increase in parallelism for each operator to keep up with the output rate. It identifies the optimal level of parallelism for each operator in the dataflow on the fly, while the computation executes, based on real-time performance traces and maintains a changing provisioning plan, i.e., the number of resources allocated to each operator. The optimal parallelism for each operator is estimated assuming perfect scaling, i.e., the true processing and output rates change linearly with the number of instances. When the "perfect scaling" assumption holds, DS2 estimations correspond to bounds and the model enjoys certain properties. For example, a scale-up decision will not result in over-provisioning and the

estimated optimal number of instances for an under-provisioned operator is always less than or equal to the minimum required to keep up with the target rate.

DS2 converges to configurations that exhibit no back-pressure (and thus keep up with the source rates) while minimising resource usage. For a given dataflow, fixed input rate, and initial configuration, DS2 identifies the optimal parallelism regardless of whether the job is initially under or over-provisioned. It identifies the lowest parallelism that can keep up with the source rate whereby further increasing the parallelism would not significantly improve latency and would waste resources, while lower parallelism would limit the observed source rate.

One of the key features of DS2 is its execution model independence. The system can be applied to streaming systems regardless of their execution model, making it a versatile solution for a wide range of applications. A streaming system's "execution model" refers to the underlying architecture, processing semantics, and runtime behaviour that define how data is processed and how computation is performed within the system. The DS2 autoscaler has been implemented atop different stream processing engines: Apache Flink, Timely Dataflow, and Apache Heron, and it has shown to be capable of accurate scaling decisions with fast convergence, while incurring negligible instrumentation overheads.

DS2 Configuration Inputs

The DS2 scaler offers several configuration options that can be helpful to avoid potential under/over-shooting and operator parallelism value oscillation. These are:

- *Policy Interval Rate* (PIR) - Defines the frequency with which metrics are gathered and the policy invoked.
- *Activation Time* (AT) - Number of consecutive policy decisions considered by the scaling manager before issuing a scaling command.
- *Warm-Up Time* (WUT) - Number of consecutive policy intervals (epochs) to ignore at system start-up time and after a re-configuration.

Figure 3.1 shows a timeline of interaction between the PIR, AT and WUT. Upon system start-up the auto-scaler undergoes a "warm-up time" during which a period of no scaling recommendations or activities are carried out (A - B). Upon completion of the warm-up period, the system then begins to analyse the incoming system metrics, requiring the completion of a certain minimum number of epochs before making a scaling recommendation is considered (B - C). Following that minimum number of epochs, DS2 then ingests metrics from each new epoch, and is able to recommend and enact a scaling activity as soon as it

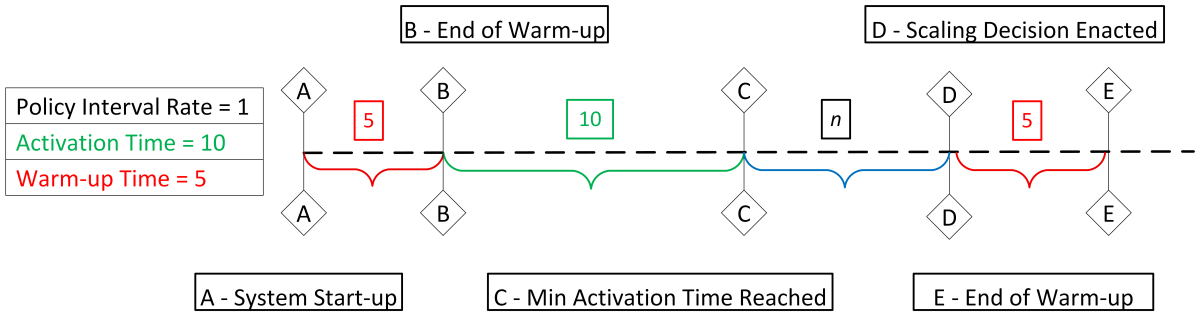


Fig. 3.1 DS2 Configuration Variables Interaction Timeline

deems necessary (C - D). Once a scaling activity has been enacted, DS2 then enters a new warm-up time (D - E) (an exact repetition of that seen at A - B), which aims to allow the system to reach steady state before subsequent scaling decisions are considered. The logic then repeats as described above.

A comprehensive example of DS2 in action is provided by Kalavri et al. [108].

3.2.2 Apache Flink

Although there exist both a number of conceptual foundations underpinning all stream processing systems and multiple shared high-level characteristics, we have to select a single DSPSs to act as a test-bed system for our experiments. Apache Flink, as introduced in Section 2.1.4, has been chosen to act as this test-bed system.

Aside from Flink's adaptability to our needs, the DS2 autoscaler chosen to act as our base experimental autoscaler has conveniently been integrated with Flink as described in [108]; a `MetricsManager` module which is responsible for gathering, aggregating, and reporting policy metrics for DS2's use was implemented in Flink.

3.2.3 Dynamic Workloads

Performance evaluation involves the analysis of system designs and implementations, with the aim of drawing comparisons that allow for educated decisions as to which design or implementation to employ in a particular circumstance. It is, however, often not solely system design and implementation which dictate how a system will perform, but rather may be influenced by the incoming workload. It is known that different workload characteristics can significantly affect streaming system performance. When used in the experimental evaluation process, workloads differing in their characteristics may lead to discrepant conclusions in both sequential and parallel systems [129].

3.2.4 Characteristics of Streaming System Workloads

A study of Facebook's in-memory key-value caching layer found workloads to be bursty, and follow a daily cycle with traffic spikes [15]. Prior studies show that disk, file system, network, and web traffic all exhibit some common temporal properties such as burstiness, self similarity, long range dependence, and daily cyclicality [148]. Pitchumani et al. [148] concluded temporal patterns can be classified into three kinds of arrival processes:

1. **Poisson Process**: A Poisson process is a simple and widely used stochastic process for modelling arrival times. Requests can be modelled as a Poisson process if the request inter-arrival times are truly independent and exponentially distributed. It is often observed, however, that streaming systems face arrival streams that are highly variable (mean < variance; over-dispersion), while in specific cases systems have to deal with almost deterministic arrivals (variance < mean; under-dispersion) [86]. These situations would violate the assumptions that the request inter-arrival times are truly independent and exponentially distributed.

2. **Self Similar Process**: Self similarity means the series looks similar to itself at different time scales and has *invariance* with respect to scaling across all time frames. If invariance is taken to mean "exact identity" then it is called *deterministic* self-similarity; if invariance is taken to mean "identical statistical properties", then it is called *statistical* self-similarity. Self similar workloads typically include bursts of increased activity, and similar looking bursts appear at many different time scales, ranging from a few milliseconds to minutes and hours [125]. Self similarity has been observed in WWW traffic [40], Ethernet local area network (LAN) traffic [125], file-system traffic [76] and also in disk-level I/O traffic [73], [157]. A self-similar traffic model leads to more realistic evaluations of required buffer space and other parameters [51].

3. **Envelope-guided processes**: [76] showed that high-level file system events exhibit *self similar* behaviour, but only for short-term time scales of approximately under a day. Envelope-guided processes reflect the fact that at longer time frames, many workloads exhibit clear daily cyclicality and general periodicity. For example, [158] observed that Internet backbone traffic exhibits both daily and weekly periodic patterns, as well as a longer-term trend. In addition, there are shorter timescale stochastic fluctuations superimposed on these patterns. They modelled this traffic by separating it into a regular, predictable component and a random component.

Workload Arrival Models

To demonstrate generality across diverse workload types and time-frames we incorporate two of the three above-mentioned arrival rate processes within the workload generation models, along with a stylised sine wave function and a monotonic step function:

Poisson Process

$$P(t) = \begin{cases} 1 - e^{-\lambda t} & \text{, if } t > 0 \\ 0 & \text{, otherwise} \end{cases} \tag{3.1}$$

where: t = inter-event time
λ = rate

Envelope-Guided Process

For Envelope-guided process based data, a selection of sine wave functions will determine the arrival rate per time interval (unmodulated and amplitude, frequency, and phase modulated). The actual inter arrivals will be generated from a secondary distribution, such as the Poisson distribution [148]. Details of the sine wave functions are presented in Section 3.2.4.

Sine Wave Function

The sine wave function is presented in Equation 3.2a, along with the amplitude modulated sine wave function in Equation 3.2b. Figures 3.2 and 3.3 show visual examples of both frequency and phase modulated waves respectively, while they are also represented in Equations 3.2c and 3.2d. Using a sine wave function to model streaming system workloads is particularly useful for predicting system load under cyclic user behaviour patterns. The sine wave function can capture the cyclical patterns of demand or usage that occur over time, such as daily or weekly peaks in user activity or data flow rates.

- Amplitude (A) can represent the maximum deviation of the workload from its average value, capturing the intensity of peak usage periods.
- Angular Frequency (ω) can reflect the cyclical nature of the workload, such as 24-hour cycles for daily patterns or longer cycles for weekly patterns.
- Phase shift (φ) can adjust the model to align with when the peaks actually occur during the cycle (e.g., peak usage during evening hours).

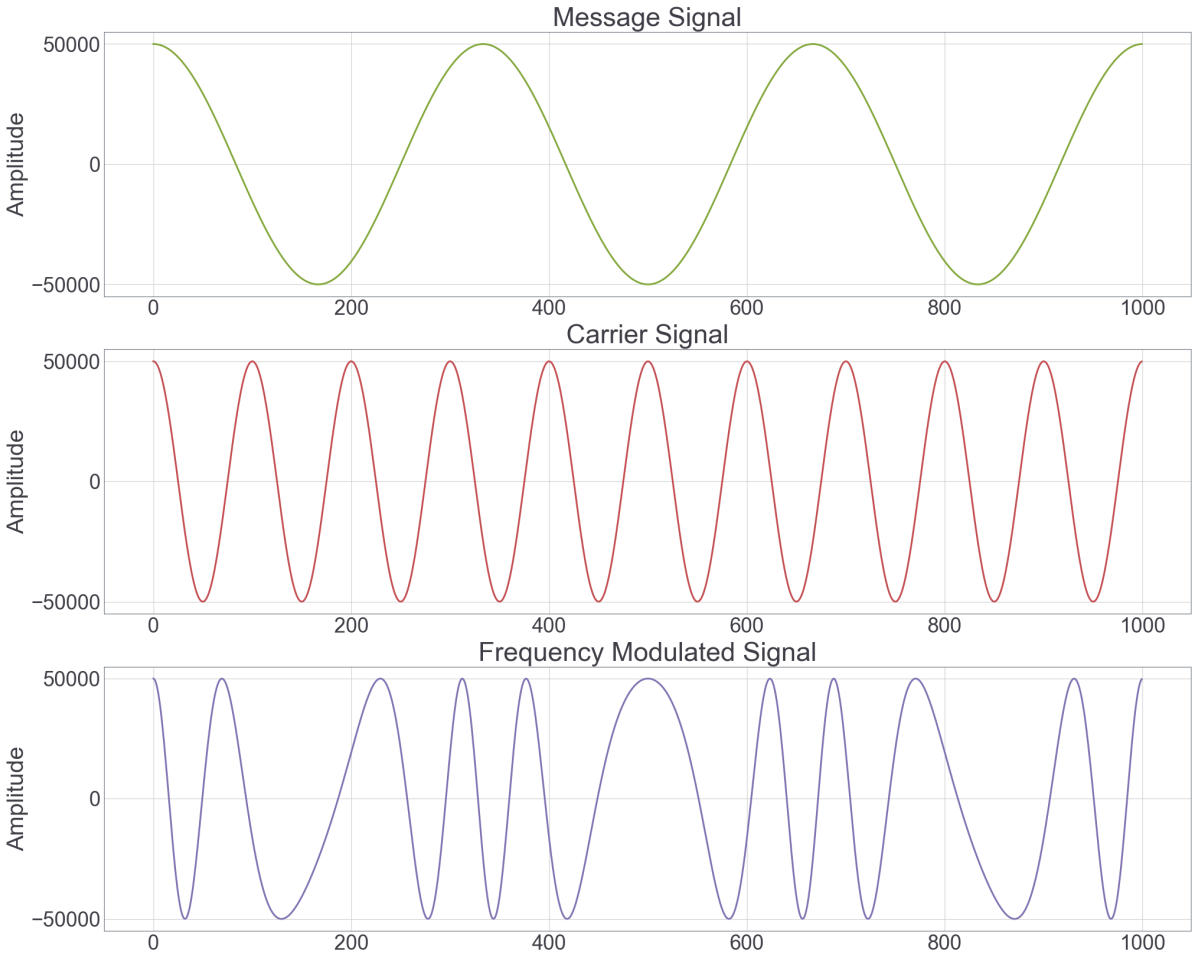


Fig. 3.2 Frequency Modulated Sine Wave

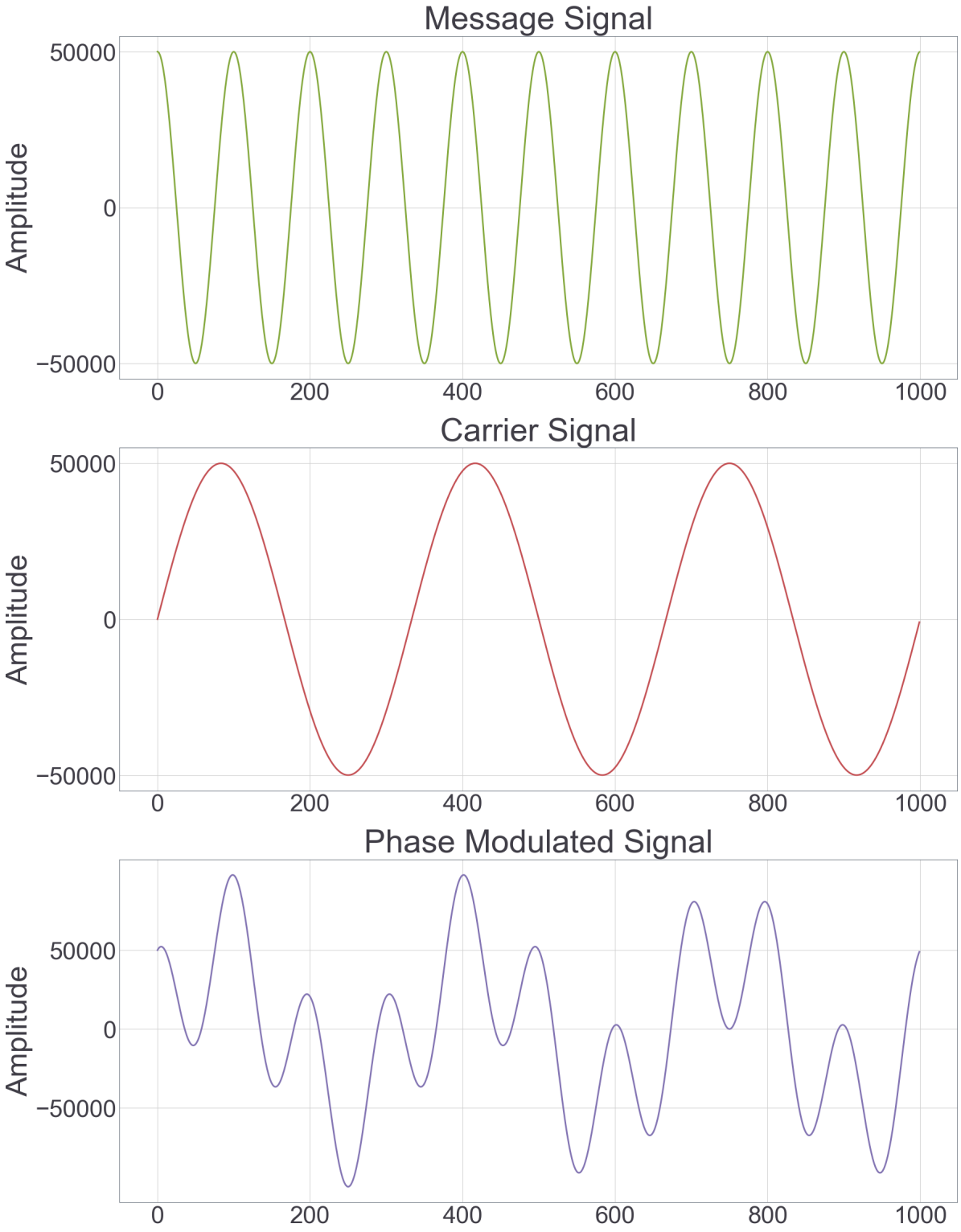


Fig. 3.3 Phase Modulated Sine Wave

- Vertical Shift can be applied to represent the baseline level of the workload, ensuring the model accounts for the average load on the system outside of peak times.

$$Y(t) = A\sin(\omega t + \varphi) \tag{3.2a}$$
$$AM(t) = A_c \sin(\omega_c t) m \cos(\omega_m m t + \varphi) \tag{3.2b}$$
$$FM(t) = A_c \cos(\omega_c t + m \sin \omega_m t) \tag{3.2c}$$
$$PM(t) = A_c \sin(\omega_c t + \cos(\omega_c t + m\omega_m t) + \varphi_c) \tag{3.2d}$$

where: t = time
A = amplitude
ω = angular frequency
φ = phase shift
A_c = carrier amplitude
m = modulation index
ω_c = carrier angular frequency
ω_m = messenger angular frequency
φ_c = carrier phase shift

Monotonic step function

The arrival rate increases by a set amount every time step for real numbers $n >= 0$, α_i, intervals A_i and indicator function χ_A of A.

$$\chi_A = \begin{cases} 1 & \text{, if } t \in A \\ 0 & \text{, if } t \notin A \end{cases} \tag{3.3}$$

$$F(t) = \sum_{i=0}^{n} \alpha_i \chi_{A_i}(t) \tag{3.4}$$

where: $n >= 0$
α_i = real numbers
A_i = intervals
χ_A = indicator function of A

Implementations for each of these arrival processes have been made available within the DEBS 2023 published research paper replication package which can be found here. This includes implementations of each `SourceFunction` class, an implementation of a `WordCount` workload leveraging each source function. These source functions are directly compatible with the WordCount implementation provided as part of DS2

Wordcount Topology

The system topology used in the experiments consists of a three-step Word Count topology with three unique operators: Source Operator, Splitter Operator, and Count Operator. A characterisation of each operator is as follows:

1. Source Operator:
    - Input: The Source Operator does not have any input from other operators in the topology. It generates the input data stream for the entire pipeline.
    - Functionality: The Source Operator is responsible for generating sentences of a specific length by randomly selecting words from a provided set of English words. It emits these sentences as the input data stream for the subsequent operators in the topology.
    - Output: The Source Operator outputs a stream of sentences, where each sentence is composed of randomly selected words from the provided set. The length of the sentences is determined by a configurable parameter.
    - Parallelism: The number of Source Operators in the topology can be varied to simulate different workload characteristics and to test the system's ability to handle varying input rates.
2. Splitter Operator:
    - Input: The Splitter Operator receives the stream of sentences generated by the Source Operator.
    - Functionality: The Splitter Operator is responsible for splitting each received sentence into individual words. It takes each sentence from the input stream and tokenises it into separate words.
    - Output: The Splitter Operator outputs a stream of individual words, which are then sent to the Count Operator for further processing.
    - Parallelism: The Splitter Operator can be parallelised to distribute the workload of splitting sentences into words across multiple instances, enabling faster processing of the input data stream.
3. Count Operator:
    - Input: The Count Operator receives the stream of individual words from the Splitter Operator.

- Functionality: The Count Operator is responsible for counting the occurrences of each unique word in the input stream. It maintains a running count for each word encountered and updates the count whenever a word is received.
- Output: The Count Operator outputs the final word counts, representing the frequency of each unique word in the processed data stream. The output can be in the form of a key-value pair, where the key is the word and the value is the corresponding count.
- Parallelism: The Count Operator can be parallelised to distribute the counting workload across multiple instances. This is particularly useful when dealing with large vocabularies or high-volume data streams.

The interaction between the operators in the topology follows a linear flow:

1. The Source Operator generates sentences and emits them as a stream.
2. The Splitter Operator receives the sentences from the Source Operator, splits them into individual words, and emits the words as a stream.
3. The Count Operator receives the individual words from the Splitter Operator, maintains a running count for each unique word, and outputs the final word counts.

3.3 Threats to Validity

Here, we introduce the limitations of this work, and highlight threats to validity arising from these [47, 193].

L1 Single streaming platform The ~40,000 workload traces used within Chapter 4 are all obtained from Apache Flink [29].

L2 Single autoscaler The experimental results in this thesis consider a single state-of-the-art autoscaler, DS2 [108].

L3 Single workload The experimental results used a single Wordcount workload, leveraging four arrival processes.

L4 Reconfiguration Cost Our experimental results cannot fully capture the interplay between reconfiguration cost (e.g. reconfiguration time) and autoscaler behaviour.

We now consider the implication of these limitations in terms of *construct, internal* and *external* validity.

Construct Validity We measured key metrics in relation to the autoscaler; average level of parallelism, the average size of a change in parallelism, the cumulative size and total number of these changes. Our experiments do not currently cater for reconfiguration costs which exceed the length of a single autoscaler evaluation period (**Limitation L4**). Further experimentation to demonstrate the performance of models within operational environments are required to measure the impact on lack of reactivity, insensitivity and synchronisation, and to quantify the impacts of oscillation (§ 4.2).

Internal Validity The experimental work we have presented makes use of DS2 [108] as the exemplar autoscaler, and also adopts its notions of *true* and *observed* processing rates. These provide the benefit of being able to see not only the number of records processed by operators within the topology, but the amount of *useful* vs *idle* time the operators spend. Consequently, this enables an autoscaler to understand the relationship across the topology, and make rescaling decisions at multiple stages of the topology simultaneously. However, it has been shown that equivalent metrics can be obtained readily from many contemporary streaming systems [108] so this does not limit the impact of this work.

External Validity Our data considers workload traces collected using a single streaming system, Flink (**Limitation L1**). Furthermore, we elected to generate our results using a Word-count workload (**Limitation L3**) which, in conjunction with representative arrival processes (§ 4.4.1), was the most parsimonious model to demonstrate the autoscaler behaviours. While we used traces from a single platform, the traces contain information disclosed by all major streaming systems, ensuring our approach can be applied across a breath of streaming systems.

We outline our lightweight assumptions for the capability of a target autoscaler, and the metrics required from a system to make our approach practicable (§ 4.4).

We have applied our approaches to a single state-of-the-art autoscaler, DS2 (**Limitation L2**). We have reason to believe the DS2 approach naturally provides more resilient handling of autoscaler behaviours than more conventional approaches, due to the use of the DS2 model of true and observed rates (c.f. Internal Validity). Our findings show strong results for DS2, and we have reason to believe we may see even greater results for situations *a)* where systems use "noisy" metrics such as CPU utilisation, and *b)* systems which adopt rule-based, threshold or heuristic based approaches.

Addressing the weaknesses outlined above, and adding generality to your results could involve several steps:

1. Expand the Platform and Autoscaler Scope: The research relies on a single streaming platform (Apache Flink) and a single autoscaler (DS2). Expanding the scope to include additional platforms and autoscalers would test the generality of your findings and possibly uncover new insights applicable across different systems.

2. Incorporating a Variety of Workloads with Varied Characteristics: This involves experimenting with different types of workloads beyond the Wordcount example used in your study. It encompasses introducing workloads with different computational requirements, data arrival rates, and patterns (e.g., bursty, cyclic, and unpredictable workloads). The objective is to ensure the autoscaling strategy is robust and effective across a diverse set of operational environments and workload demands.

3. Reconfiguration Cost: The current experimental results do not fully capture the interplay between reconfiguration cost (e.g. time), and autoscaler behaviour. Investigating the impact of reconfiguration costs in greater depth could provide insights into optimising autoscaler performance while minimising resource consumption and operational delays.

4. Further Experimentation: Further experimentation within operational environments is suggested to measure the impact on lack of reactivity, insensitivity, and synchronisation, and to quantify the impacts of oscillation. This could involve real-world deployments or more sophisticated simulation environments that better capture the dynamics of streaming systems under varying conditions.

5. Cross-platform and Cross-autoscaler Generalisability: To enhance the generality, one can conduct cross-platform and cross-autoscaler studies. This involves validating autoscaling approaches against various streaming systems and autoscalers to assess their effectiveness and adaptability across different environments.

Chapter 4

On Improving Streaming System Autoscaler Behaviour

Related Publications

The work presented in this chapter appeared in the following publication: Stuart Jamieson and Matthew Forshaw (2023). On Improving Streaming System Autoscaler Behaviour using Windowing and Weighting Methods. In: Proceedings of the 17th ACM International Conference on Distributed and Event-Based Systems (DEBS). Association for Computing Machinery, 2023. New York, NY, USA. DOI: 10.1145/3583678.3596886. URL: https://doi.org/10.1145/3583678.3596886.

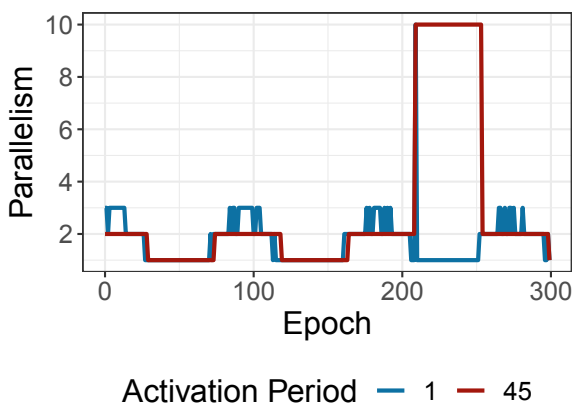
4.1 Introduction

Performant distributed stream processing systems (DSPSs) are essential to process large volumes of high-velocity data in a reliable and timely fashion. These systems commonly experience highly variable, bursty and unpredictable workloads [167]. These workloads typically display strong time-of-day and time zone correlation, flash crowd behaviour [170], cyclicality and periodicity [85]. Such incoming workloads can make it difficult to plan a system's capacity requirements and ensure an SLA is fulfilled, while also keeping costs low. If capacity is set to manage expected peak load, performance will be maintained but costs will rise accordingly as resources lay idle for significant periods of time. If capacity is set to manage average expected load, costs will be curtailed, but performance will degrade when faced with above average loads.

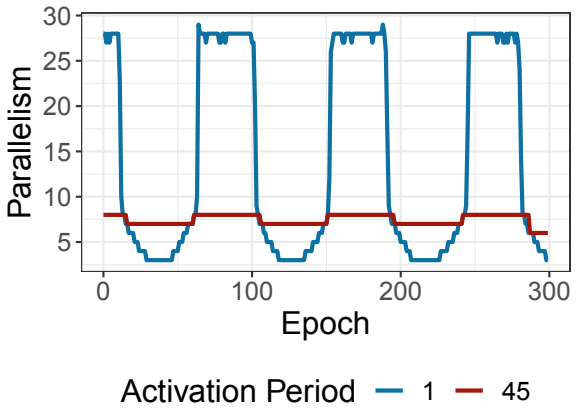
Recent years have seen the emergence of myriad streaming systems, e.g. Storm [181], Spark [162] and Flink [29]. With these systems have come efforts to develop DSPSs with the ability to scale system resources in an *on-demand* manner through the use of an *autoscaler*. The aim of an autoscaler is to dynamically adapt the resources assigned to a system, depending on the input workload, to maintain an acceptable trade-off between the required performance and the associated costs of providing the resources. Autoscaler are prone to suffer from undesired behaviours [185]. Examples of these are under- or over-provisioning, oscillation and chasing behaviour [132].

Autoscalers can manage and deploy system resources dynamically, and in an automated fashion. Commonly, such autoscalers are governed by tunable parameters/thresholds with the non-trivial task of choosing those values being left up to the user. Autoscaler system behaviour can be highly sensitive to the choice of such input values, resulting in high levels of volatility, and therefore uncertainty regarding the outcome of any particular choice of such values. While overall uncertainty of outcome is considered undesirable in a business operating environment, certain case specific behaviours at the autoscaler level itself can clearly be determined to be undesirable also, from a system operator's perspective.

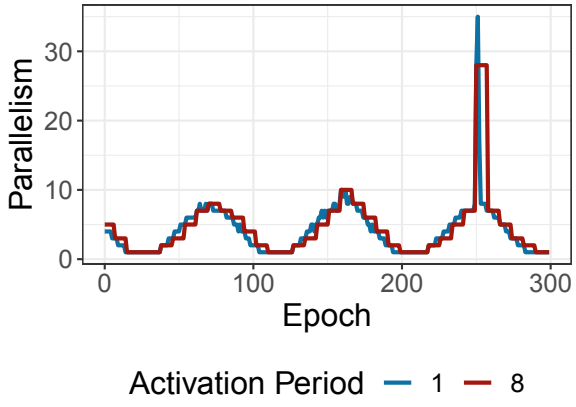
In this chapter we empirically study undesirable behaviours experienced by a state-of-the-art controller (Section 4.2) and contribute a categorisation of autoscaler mechanisms (Section 4.3). We explore, for realistic workloads (Section 4.4.1) the feasibility of applying moving average models (Section 4.4.2). We establish methods (Section 4.4.4) to systematically evaluate these models, and demonstrate the potential to augment existing autoscalers to mitigate at least 90% of undesirable extreme parallelism shifts observed and significantly reducing scaling behaviour volatility (Section 4.5). We make available a replication package containing 40,000 workload traces collected as part of this work, all code and workloads to reproduce our findings (Section 4.6). Finally we conclude and offer suggestions of areas for future research (Section 4.7).



(a) Lack of Reactivity



(b) Insensitivity



(c) Extreme Parallelism Shift

Fig. 4.1 Illustrative examples of Autoscaler failure categories. Blue lines represent the instantaneous parallelism suggestions from our exemplar autoscaler, DS2, configured with an activation period of 1. Red lines shows the parallelism suggestions for the auto-scaler with a specific value of tunable parameter activation period, which exhibits undesirable behaviour. Full details of the workloads to recreate these experiments are available in the replication package (§ 4.6).

4.2 Categories of Autoscaler Failure

Autoscalers seek to provide the SASO properties; stability, accuracy, short settling time, and not overshooting [2, 108]. Autoscalers which lack stability, make sub-optimal decisions, or do not settle quickly, are considered undesirable. We empirically evaluate the behaviour of the state-of-the-art controller, DS2 [108], for a variety of representative workloads. We focus primarily on the impact of activation period, the number of consecutive policy decisions considered by the autoscaler before issuing a re-configuration.

Our findings identify additional failure categories beyond those identified by SASO. Their grounding in experimental results, based on a state-of-the-art controller for representative workloads, demonstrates the importance of these factors.

Under or Over- provisioning: results in resources operating below their capacity and incurring excess cost, or SLA violation due to insufficient resources. In many real-world instances, some controlled level of over-provisioning of resources is effected to cope with minute workload fluctuations and is considered desirable.

Oscillation: manifest when scaling decisions are enacted in quick succession, leading to rapid changes in configuration. Oscillations are detrimental to system performance due to the performance impact of reconfiguration, which can harm latency and throughput for periods of seconds to minutes. In extreme circumstances, these oscillations may represent transitions between under- and over-provisioning, leading to instability and SLA violation. The acquisition and subsequent release of resources represents resource wastage, and in cloud deployments can be financially costly.

Volatility: Highly volatile autoscaler behaviour across a relatively small range of chosen input parameter values, in terms of both average parallelism levels and average parallelism change size, and in terms of average and cumulative magnitude of scaling decisions enacted throughout an individual workload.

Lack of Reactivity (Figure 4.1a): Reactivity being the speed at which scaling actions are taken to correct previously inaccurate decisions, i.e., the speed at which an inaccurate decision is overwritten and a new attempt at an accurate decision is made. Measured as a combination of time passed and magnitude of deviation between the current state and the optimal/correct state.

Insensitivity (Figure 4.1b): the speed at which new information regarding the system environment (e.g. resource failure, or changes in offered load) is assimilated and acted upon.

Synchronisation: Due to the use of a particular input parameter value (e.g. for activation period), the autoscaler can be "out-of-sync" with the dynamic workload. That is, the autoscaler will cause the operators to be at high or increasing levels of parallelism while the incoming workload rate is low/decreasing, and vice-versa.

Extreme parallelism shifts (Figure 4.1c): incur substantial overhead costs, require substantial scale out and impact system performance while they are being enacted. Extreme shifts are characterised as 1) rapid, 2) large and 3) quickly retraces its movement. DS2 allows the selection of certain input parameters that should guard against this behaviour (i.e. activation period and warm-up time). Setting a larger value for activation period or warm-up time helps guard against enacting a scaling decision based upon a large, one-off suggested change in parallelism, which is then quickly followed by a suggestion to revert back to (close to) the previously seen level. However, if the extreme scaling suggestion coincides with the end of an activation period or warm-up time window, the scaling decision will be enacted. It will then remain at the extreme level of parallelism until the next time period the scaler is free to enact a further scaling decision (i.e., once the next activation period and warm-up time thresholds have been met). Typically, the higher the activation period, the fewer times such behaviour will occur, but the higher will be the impact of each occurrence as the extreme level of parallelism is maintained for a longer period.

Lack of Behavioural Robustness: Finally, we saw a lack of robustness regarding how scaling behaviours manifest across a range of input parameter values. We see instances of large deviations in style of behaviour (e.g. insensitivity switching to lack of reactivity switching to operating in an out-of-sync manner) across the same workload when the activation period is varied across a relatively small range. This refers to the type of failure behaviour, rather than the *Volatility* of the autoscaler's behaviour.

4.3 Background

Building upon the concepts introduced in the literature review (Chapter 2), recent works have further explored the challenges and implications of autoscaler parameterisation and performance evaluation in streaming systems. For a comprehensive background we refer the reader to [31] and [186]. Wang *et al* [190] extend the discussion on the trade-offs between auto-scaling systems and manual expert tuning, providing insights into the behaviour of production autoscalers in cloud environments.

The use of non-parametric goodness-of-fit models to quantify streaming system robustness has been investigated by [97, 96]. These studies delve into the categorisation of scaling decision volatility and its impact on system performance, introducing new methods for assessing the robustness of streaming systems.

Straesser *et al* [173] contribute to the benchmarking and evaluation methodologies discussed in Section 2.4.2, offering practical guidelines for communicating autoscaler performance under diverse workload patterns. Their work aims to assist practitioners in evaluating

and comparing autoscaler performance, building upon the foundational concepts presented in the literature review.

4.4 Preliminaries and Model

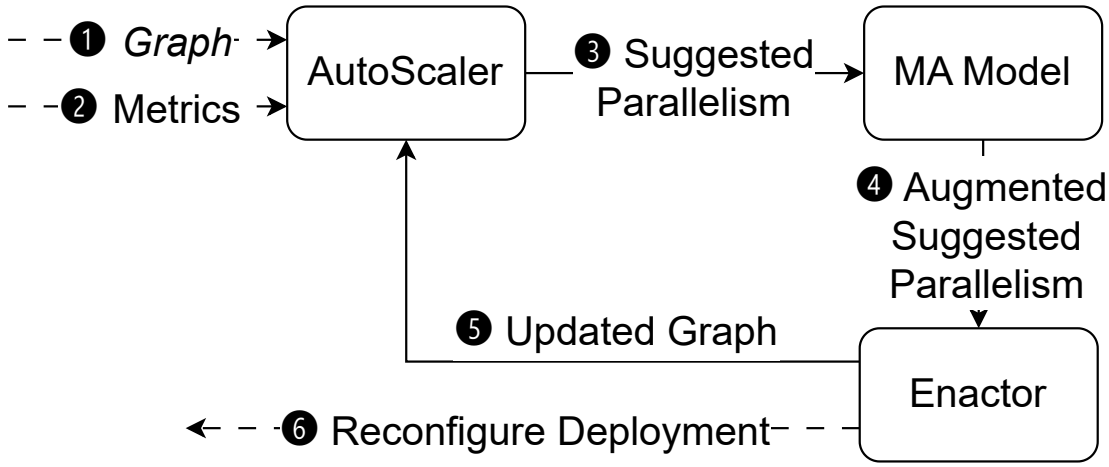


Fig. 4.2 Logical diagram showing the integration of MA models into a typical autoscaler deployment. Dashed lines represent data arriving from, or decisions flowing to, components outwith our system.

Here we outline the conceptual model of our chosen autoscaler (DS2) and moving average logic. Figure 4.2 shows the high-level architecture of the integration with the moving average models.

Streaming System Model: We model an exemplar streaming workflow as a directed acyclic graph $G = (V, E)$ where:

- V is a set of vertices representing operators within the topology (including data sources, and sinks), each with property p representing the current parallelism level for that operator.
- E is a set of edges representing the connection between logical operators.

The graph is dynamic, evolving over time in response to failures, or changes enacted through autoscaling or manual intervention.

Autoscaler Model: We model our autoscaler as a black-box, accepting as its parameters the graph $G = (V, E)$, and a stream of metrics from the streaming operators in V, or from the compute resources (e.g. CPU/memory usage). The operator emits tuples $e = (t, o_{id}, p)$ where

at time t, $o \in V$ denotes the operator within the system, and p the suggested parallelism level. The autoscaler stores as its state t_{last}, the time step at which the last rescaling suggestion was emitted. Our model permits autoscalers which produce zero or more tuples per time step, allowing us to model autoscalers capable of scaling single [54] or multiple [108] operators per time period.

Autoscalers commonly introduce *guards* (often as tunable threshold parameters) to mitigate several of the behaviours we explore in this chapter. For example, they may enforce a minimum time period ν between successive reconfigurations ($t - t_{last} \leq \nu$).

Enactor: We model an *enactor* taking a stream of tuples (t, o_{id}, p) from the autoscaler, and is responsible for enacting these changes. Multiple tuples received at t are combined to provide a new G'. We give flexibility with a non-zero reconfiguration duration, after which an updated graph $G' = (V', E')$ updates the autoscaler state.

Generalisability: Our approach is agnostic to autoscaler approach. We expect it to work particularly well in situations where systems; *a)* use "noisy" metrics such as CPU utilisation, and/or *b)* rule-based, threshold or heuristic approaches. Our approach reduces the number of rescaling decisions and their latency and throughput impacts, through a) reduced oscillations, b) mitigating costly-to-enact extreme parallelism shifts. Our approach is complementary to efforts to reduce reconfiguration complexity [92], manage state efficiently [107] and state-aware enactment of rescaling decisions [42].

Implementation: Our proposed approach is agnostic to workload and operator type. It relies only on the light assumptions outlined here, and requires only information which is commonly available at runtime in major stream processing engines. Throughout the remainder of the paper we experiment with DS2 as a robust, best-in-class autoscaler, making it a challenging system to evaluate against. We anticipate greater benefits for other autoscalers (Section 3.3).

We apply moving average models to the suggested parallelism values provided by the autoscaler. This is most appropriate for DS2, allowing us to leverage its recursive approach to consider parallelism values for all operators simultaneously. It would also be possible to apply the moving average models to the stream of metric tuples ahead of the autoscaler.

4.4.1 Summary of Workloads

We use the most parsimonious workload capable of replicating autoscaler failure behaviours, a three-step *Word Count* topology, coupled with realistic arrival processes. *Source Operators* emit sentences of a specified length, the *Splitter Operators* split them into individual words, and the *Count Operators* count the number of times each word occurs. Unlike benchmarks

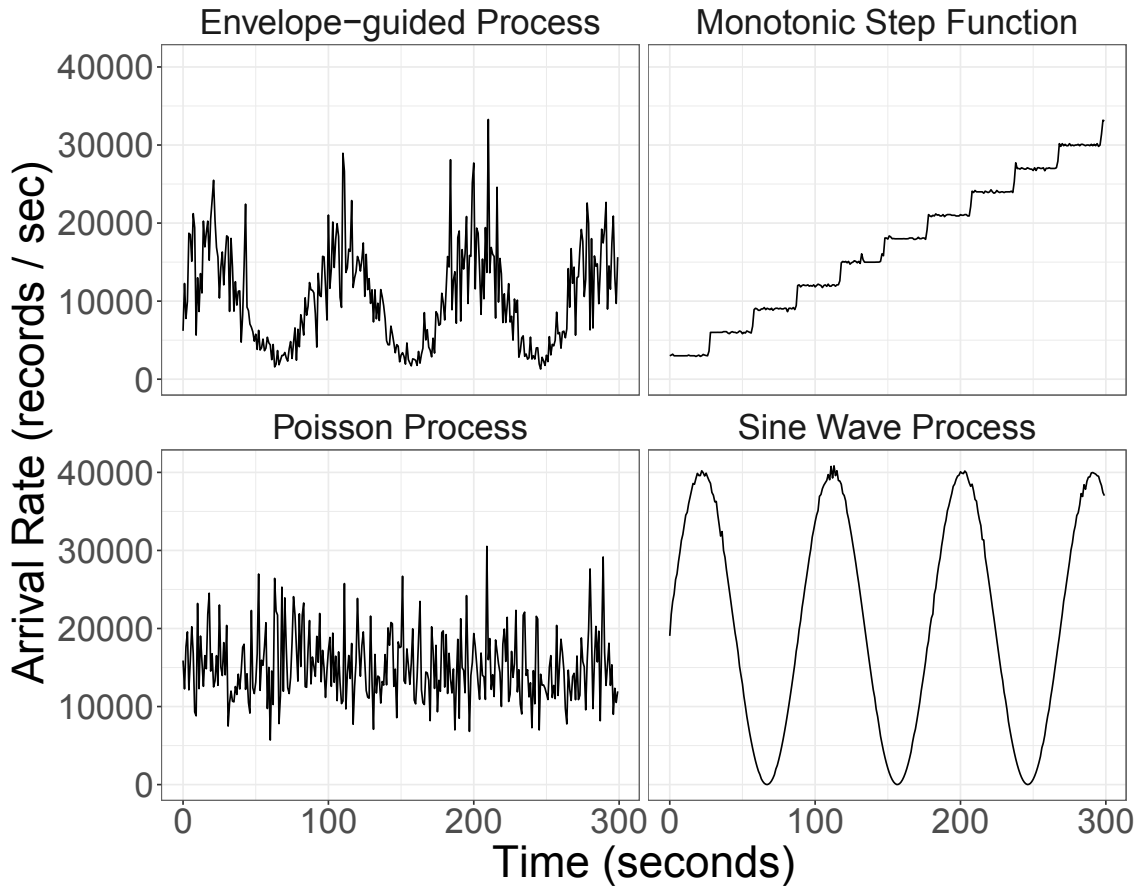


Fig. 4.3 Examples of four workload generation functions: Poisson process, sine wave process, envelope-guided process, and monotonic step function, illustrating different patterns and characteristics of workload arrivals over time.

which commonly use uniform arrival rates, we use four classes of arrival process to evaluate models' ability to overcome challenges of workload variability.

Poisson processes (Equation 3.1): Our arrival rate at t was modelled with a Poisson distribution with different λ values.

Sine wave (Equation 3.2): Functions determine arrivals; unmodulated $Y(t)$ and amplitude $AM(t)$, frequency $FM(t)$, and phase modulated $PM(t)$.

Envelope-guided process: In our case the regular, predictable component is modelled by a selection of sine wave functions which determine the workload arrival rate per time interval (Equation 3.2). Interarrivals were generated from the Poisson distribution, modelling the stochastic component (Equation 3.1). Burstiness is determined by a secondary distribution.

Monotonic Step Function (Equations 3.3 and **??**): the arrival rate increases by a set amount every time step.

4.4.2 Summary of Moving Average Models

Moving Average (MA) models offer us a principled approach to react to changing system state, while incorporating valuable historical information across a *window period* of configurable length. Desirable properties for MA models are *a)* smoothness, *b)* insensitivity to noise, *c)* they do not lag behind the original time series. In our work, we evaluate a number of MA models, as outlined below, for window length k and n observed values.

Simple MA (SMA) [155] weights equally in window period k.

$$SMA_k = \frac{1}{k} \sum_{i=n-k+1}^{n} x_i \tag{4.1}$$

Linearly Weighted MA (WMA) [155] applies linearly decreasing weights, with the greatest weight given to recent values, w_i weight assigned to value at time i.

$$WMA_k = \sum_{i=n-k}^{n} (x_i \times w_i) \Big/ \sum_{i=1}^{k} w_i \tag{4.2}$$

where: x_i = value of underlying time series at time i
k = window size
n = total number of observed values
w_i = weight assigned to value at time i

Exponentially Weighted MA(EMA) [95] applies a non-uniform weighting, so that recent value is weighted more heavily. Weights use a smoothing factor based upon the exponential

function α.

$$EMA_k = \frac{x_k + (1-\alpha)x_{k-1} + (1-\alpha)^2 x_{k-2} + \cdots + (1-\alpha)^k x_0}{1 + (1-\alpha) + (1-\alpha)^2 + \cdots + (1-\alpha)^k} \tag{4.3}$$

where: x = value of underlying time series
k = window size
α = smoothing factor

Double Exponential MA (DEMA) [155] and **Triple Exponential MA(TEMA)** [166] are composite MAs atop EMA with window size k. To improve smoothness DEMA runs an MA on itself, but this operation increases lag, so to counter this problem it uses the so called "twicing" technique [184]. $TEMA$ applied EMA three times.

$$DEMA_k = (2 \times EMA_k) - (EMA(EMA_k)) \tag{4.4}$$

$$\begin{aligned} TEMA_k = (3 \times EMA_k) - (3 \times EMA(EMA_k)) \\ + (EMA(EMA(EMA_k))) \end{aligned} \tag{4.5}$$

where: k = window size
EMA = Exponentially Weighted MA

T3 Moving Average (T3MA) [180] is calculated by taking the weighted sums of a simple, a double, and a triple EMA. This smoothing technique allows the model to produce curves more gradual in nature than "ordinary" moving averages and with smaller lag, making it smoother and more responsive. However, it bears the disadvantage of a tendency to overshoot the underlying values as it attempts to realign itself to current values.

$$\begin{aligned} GD_{k,v} &= EMA_k \times (1+v) - EMA(EMA_k) \times v \\ T3_k &= GD(GD(GD_k)) \end{aligned} \tag{4.6}$$

where: k = window size
GD = Generalised DEMA
v = volume factor

Sine Weighted Moving Average(SINWMA) [155] is a weighted average, based on motivation that values of a particular time series fluctuate following some unknown wave. As a

model, the Sine wave is used to adjust value weights.

$$SINWMA_k = \frac{\left(\sum_{i=1}^{5} \sin\left(\frac{i\pi}{6}\right) x_{k-i+1}\right)}{\left(\sum_{i=1}^{5} \sin\left(\frac{i\pi}{6}\right)\right)} \tag{4.7}$$

where: x_i = value of underlying time series at time i
k = window size

Arnaud Legoux Moving Average (ALMA) [13] removes small time series value fluctuations and enhances the trend by applying a moving average twice, one from left to right and one from right to left. It is inspired by the use of Gaussian Filters in image processing, it uses a Gaussian distribution shifted with an offset so that it's not evenly centred on the window but biased towards the more recent time periods. The offset is adjustable to enable the trade-off between smoothness and responsiveness. The second parameter is the sigma (σ) parameter which changes the shape of the filter making it wider (with a larger σ) or more focused (with a smaller σ).

$$ALMA_k = \frac{1}{NORM} \sum_{i=1}^{k} x_i e^{-\frac{(i-O)^2}{\sigma^2}} \tag{4.8}$$

where: x_i = value of underlying time series at time i
k = window size
O = offset
σ = filter range

Kaufman's Adaptive Moving Average (KAMA)[112] was developed by Perry Kaufman, Kaufman's Adaptive Moving Average is a moving average designed to account for underlying noise or volatility in a time series. KAMA will closely follow observed values when the swings are relatively small and the noise is low. KAMA will adjust when the swings in value widen and follow the observed values from a greater distance. This trend-following indicator can be used to identify the overall trend, time turning points and filter movements in time series data.

The *Efficiency Ratio (ER)* is used as a mechanism to sense speed of change and choppiness of the observed data values. The ratio divides the underlying time series net movement by the total movement (the sum of each of the individual moves taken as a positive num-

ber). It can also be considered a ratio of the time series direction to its volatility. The ER is sometimes named *generalised fractal efficiency*.

A *scaled smoothing constant* is then calculated by mapping the ER onto a range of trend speeds. If the trend speed is converted to a smoothing constant approximation using $sc = {}^{2}/_{(N+1)}$, then the slowest speeds have the smallest values and the formula for the scaling constant becomes:

$$ssc = ER \times (sc_{fast} - sc_{slow}) + sc_{slow} \tag{4.9}$$

Finally, the scaled speed value is squared to retrieve the smoothing constant as follows $c = sc \times sc = sc^2$. The smoothing constant is calculated every time period and used in the EMA formula; this becomes the KAMA:

$$KAMA_k = KAMA_{i-1} + c \times (x_i - KAMA_{i-1}) \tag{4.10}$$

Midpoint (MIDPOINT) [104] the calculated average of values, is assigned to the midpoint of the time period window.

$$MIDPOINT_k = \frac{(\max(x_1, x_2, \ldots, x_i)) + (\min(x_1, x_2, \ldots, x_i)) \times}{2} \tag{4.11}$$

Wilder's Moving Average (RMA) [192] is similar to the EMA with the difference that the RMA uses a smoothing factor of $\alpha = {}^{1}/_{n}$ instead of $\alpha = {}^{2}/_{(n+1)}$ as is the case for the EMA. This alternative smoothing factor results in the RMA responding more slowly to changes in the observed values when in comparison.

$$RMA_{k,i} = \begin{cases} SMA_{k,i} & \text{if } RMA_{k,i-1} = 0 \\ RMA_{k,i-1} + \frac{1}{k}(x_i - RMA_{k,i-1}) & \text{otherwise} \end{cases} \tag{4.12}$$

where: n = total number of observed values
x_i = value of underlying time series at time i
k = window size
SMA = Simple Moving Average

Symmetric Weighted Moving Average (SWMA) [104] sets the weights applied to individual values, based on a symmetric triangle.

$$SWMA_k = \sum_{j=-t}^{t} a_j \times SWMA_{k+j} \begin{cases} a_j & = a_{-j} \\ \sum_{j=-t}^{t} a_j & = 1 \end{cases} \tag{4.13}$$

where: x_i = value of underlying time series at time i
k = window size
c_i = coefficient of statistic at time i

Triangular Moving Average (TRIMA) [104] is a variation on the Simple Moving Average. Rather than giving equal weight to all days in the period, the shape of the weights are triangular and the TRIMA gives progressively more weight to days in the middle of the period. It is equivalent to a double smoothed simple moving average.

$$TRIMA_k = \frac{(SMA_1 + SMA_2 + ... + SMA_n)}{n} \tag{4.14}$$

where: k = window size
SMA = Simple Moving Average

Zero Lag MA (ZLMA) [45] adds a momentum factor to reduce lag in the average so that it can track current prices more closely. It achieves this by positively weighting recent prices in the window period while adding negative weights on old prices.

$$\begin{aligned} \beta = 0, \text{ for } n > 1 \text{ else } 2 \\ ZLMA_k = \alpha(x_{k-1} + \beta \times (x_k - ZLMA_{k-1}(x_{k-1})) \\ + (1-\alpha) \times ZLMA_{k-1}(x_{k-1}) \end{aligned} \tag{4.15}$$

where: x_i = value of underlying time series at time i
k = window size

We wish to establish an intuition as to the properties of each MA model and how we anticipate they will relate to the categories of autoscaler failure outlined in (Section 4.2). Figure 4.4 highlights that our models occupy a continuum between smoothness and responsiveness. We anticipate methods such as Simple, Linearly Weighted, and Wilder's Moving Average will exhibit smooth behaviour, and will address oscilation, volatility and behavioural

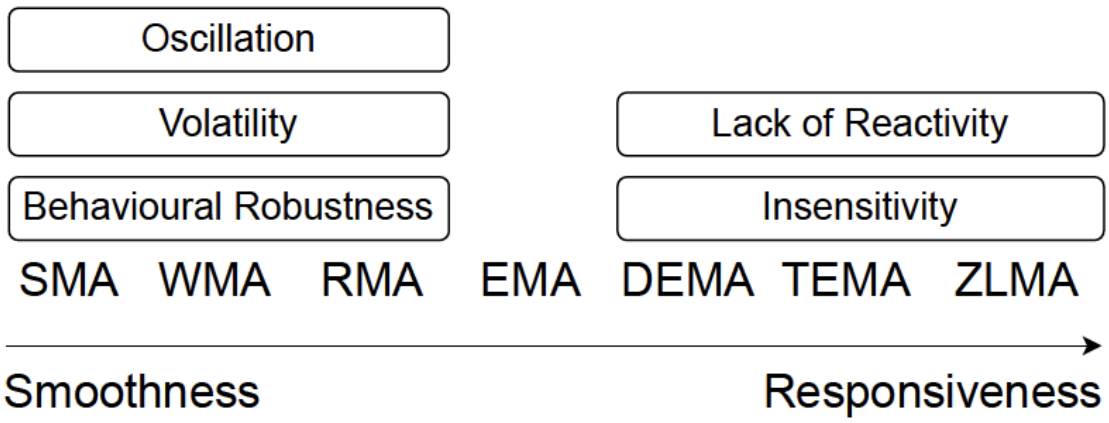


Fig. 4.4 Interplay between moving average models on categories of autoscaler failure (§ 4.2)

robustness. Meanwhile, moving average models designed to respond to recent values, such as Zero Lag (ZLMA) and Triple-Exponential Moving Average (TEMA) we expect to overcome non-reactivity and insensitivity.

4.4.3 Experimental Environment

The evaluation used Apache Flink 1.4.1 [29] and ran on an X570 AORUS ULTRA with 12-core AMD Ryzen 9 3900X and 64GB RAM.

Each of our workload traces (39,000 with extreme parallelism shifts and 400 chosen for volatility analysis), were run through DS2 in *offline* mode to provide our "ground truth" values. To generate our "ground truth" values, we use DS2 with an activation period of one. Setting the activation period to one means that DS2 is configured to make parallelism adjustments at the highest possible frequency, i.e., at every time step or epoch. By using an activation period of one, we aim to achieve "instantaneous optimal levels of parallelism" for the given workload ($DS2_{act1}$). This means that DS2 is expected to adapt the parallelism of the system in real-time, based on the immediate workload requirements, without any delay or aggregation of metrics over multiple time steps. The use of DS2 with an activation period of one serves as a baseline or reference point for evaluating the system's behaviour and performance under ideal conditions, where the autoscaler can instantly adapt to workload changes. We perform further experiments across *a)* a range of activation periods, *b)* applying a range of moving average models, for varying window periods.

4.4.4 Summary of Comparison Metrics

We evaluate the efficacy of each moving average model with respect to three metrics; accuracy, bias and smoothness.

Accuracy [156]: the absolute difference between "ground-truth" values and values with a smoothing model applied. We take the instantaneous estimates of optimal operator parallelism from DS2 with an activation period of one ($DS2_{act1}$). Accuracy for an MA with window size n at time t is calculated as shown in Equation 4.16. The average accuracy for the entire dataset is given by Equation 4.17.

$$a_t = |X_t - ma_t^n| \tag{4.16}$$

$$a^{ma^n} = \frac{1}{k}\sum_{t=1}^{k}|X_t - ma_t^t| \tag{4.17}$$

for accuracy a_t and value X_t at time t, ma_t is the moving averaged series at time t, for window size n. Smaller values are better.

Bias: the tendency to observe a positive or negative difference between the value of the underlying "ground-truth" and the value of the MA, across an entire data set, in similar fashion to Equation 4.17, without converting to absolute values. This represents the bias of a model towards under vs over-provisioning of resources.

Smoothness [156]: the tendency of the model to propose contradictory scaling decisions (Equation 4.18). Characteristically similar to the second derivative of the time series. A smaller smoothness value signifies the model has a lower tendency to oscillate, and less likely to follow a scaling action with one in the opposite direction.

$$smo^{ma^n} = \frac{1}{k}\sum_{t=3}^{k}|ma_t^n - 2ma_{t-1}^n + ma_{t-2}^n| \tag{4.18}$$

4.4.5 Model Ranking and Selection

Table 4.1 Moving Average Model Ranking Table

		Wins					Avg Rank					Rank		
Name	Symbol	P5	P10	P20	P50	Total	P5	P10	P20	P50	Avg. Rank	Wins	Avg Rank	Combined Rank
Triple Exponential MA[166]	TEMA	17	17	21	22	77	4.4	5.3	4.6	3.5	4.5	1	1	1
Linearly Weighted MA[155]	WMA	11	11	7	2	31	3.5	3.6	4.9	6.0	4.6	3	2	2
DS2[108]	DS2	14	16	17	10	57	6.7	7.7	7.8	8.0	7.6	2	8	3
Exponentially Weighted MA[95]	EMA	6	7	4	4	21	3.8	4.2	5.1	5.2	4.6	5	3	4
Double Exponential MA[155]	DEMA	8	5	4	7	24	5.1	6.1	6.1	4.5	5.5	4	4	5
Zero Lag MA[45]	ZLMA	7	6	2	4	19	5.3	5.9	6.5	5.4	5.8	6	6	6
Wilder's MA[192]	RMA	2	2	5	8	17	5.7	5.8	5.7	5.5	5.6	7	5	7
Arnaud Legoux MA[13]	ALMA	0	0	0	2	2	8.4	7.8	6.6	5.8	7.3	12	7	8
Simple MA[155]	SMA	0	0	2	2	4	8.0	7.2	7.4	8.2	7.7	11	9	9
Tilson MA[180]	T3MA	1	2	1	1	5	7.4	7.8	8.5	7.5	7.8	8	10	10
Kaufman's Adaptive MA[112]	KAMA	1	1	1	2	5	9.2	9.9	9.2	8.5	9.2	8	12	11
Midpoint[104]	MIDPOINT	0	0	3	2	5	9.2	9.0	9.3	9.5	9.2	8	13	12
Sine Weighted MA[155]	SINWMA	0	0	0	1	1	8.9	8.8	9.2	9.3	9.0	13	11	13
Symmetric Weighted MA[104]	SWMA	0	0	0	0	0	10.7	10.5	10.9	9.7	10.5	14	14	14
Triangular MA[104]	TRIMA	0	0	0	0	0	11.5	11.5	11.9	10.6	11.4	14	15	15

To evaluate each moving average model, we use the estimated optimal parallelism values suggested by the autoscaler, across a range of activation periods. The logic is that one should compare moving average models with similar smoothness values, setting that as a base characteristic. We aim to identify moving average models with both a small accuracy value and a small smoothness value, as that would correspond to a model that is both stable and remains close to "optimal". We now compare models of similar smoothness and identify those which perform better, as measured by accuracy. Firstly, for each workload, and activation period, we select all models with the same or better smoothness. Using the EMA model as the example, with $n \in (5, 10, 20, 50)$ and smoothness S:

$$g = \arg\min_{g} S_g^{EMA} \leq S_n^{DS2} \tag{4.19}$$

We then calculated the accuracy scores for all selected models, identified the window period parameters for each model that produced the smallest accuracy score, and ranked the models according to their best accuracy score. Two approaches were then used to create a set of interim results, which are presented in Table 4.1:

1. Winner selection based on voting. The winning model, at each window period, for each workload was that with the smallest accuracy score. The most often winning model across all window periods and workloads was selected.

2. Winner selection based on aggregated rankings. For each workload, for each window period, the models were ranked by accuracy and assigned an integer value based on that rank. Rank values for each model were then averaged across all workloads and all window periods, and the model with the smallest average rank value was selected.

The final "Combined Rank" was calculated by normalising "Total Wins" and "Average Rank" to the range 0 to 1 with the largest "Total Wins" value, and the smallest "Average Rank" value, assigned the normalised value of 1. For each model, these two scores were averaged and ordered to assign a "Combined Rank".

The justification for this approach is based on the following considerations:

1. Equal Importance: By assigning equal weight to both "Total Wins" and "Average Rank", the combined metric attributes both components equal importance in assessing the overall performance of the moving average models. This implies that the ability of a model to consistently outperform others (as measured by "Total Wins") and its average ranking across all scenarios (as measured by "Average Rank") are considered equally valuable in determining its overall effectiveness.

2. Balancing strengths and weaknesses: Each component of the combined metric captures a different aspect of the models' performance. "Total Wins" focuses on the frequency of a model being the best performer, while "Average Rank" takes into account the models' relative position across all scenarios. By combining these two components with equal weight, the metric balances the strengths and weaknesses of each model, preventing a model that excels in one aspect but significantly underperforms in the other from being overly favoured or penalised.

3. Robustness and generalisability: Giving equal weight to both components helps to identify models that exhibit robust and consistent performance across various scenarios. A model that achieves a high "Total Wins" score but has a poor "Average Rank" may be overly specialised or sensitive to specific conditions. Conversely, a model with a good "Average Rank" but low "Total Wins" may be a strong performer overall but rarely outperforms others. By considering both components equally, the combined metric favours models that strike a balance between being the best performer in some cases and maintaining a consistently high ranking across all scenarios.

The top six performing MA models (as shown in in Table 4.1) were selected for consideration and application when attempting to alleviate any of the autoscaler failures selected for analysis. Due to being a foundational element in the construction of a number of the selected models, the SMA model was also included, making seven short-listed models in total (those presented in Section 4.4.2).

4.5 Findings and Results

The remainder of our analysis centres around two key areas. First we present MA models' effectiveness in ameliorating the impacts of extreme parallelism shifts (Section 4.5.1). We then systematically explore the volatility of the methods under consideration (Section 4.5.2).

4.5.1 Extreme Parallelism Shift

We aim to identify extreme parallelism shifts which display the characteristics described in Section 4.2. These shifts are particularly costly to enact, due to their substantial resource requirements and the performance impact during the extended time they take to enact.

We then investigate the effectiveness of applying a range of MA models in order to mitigate such shifts. Efforts are made to also quantify the sensitivity of each MA model with regard to its effect on the extreme shift behaviour, while varying the window period value

input. Some MA models may display changes in levels of effectiveness and sensitivity that are directly proportional to the magnitude of a shift (i.e., they may work better at mitigating the effects of relatively large extreme shifts, than they do small ones), while others may be inversely proportional.

Analysis was carried out on 39,000 workload traces (split evenly among the four workload functions presented in Section 4.4.1), containing extreme parallelism shifts that exceeded eight standard deviations in size above the average parallelism change of that individual workload. To identify and locate instances of what we would consider extreme parallelism shifts, we wish to isolate moves which are 1) rapid, 2) large and 3) retraces its movement in a similar fashion.

- *Rapid* will be captured by measuring changes across a pre-defined, short rolling window, within a set period of time.
- *Large* will be captured by selecting changes larger than a pre-defined number of standard deviations above the mean.
- *Retracement* identifies if the parallelism level returns within a threshold distance of an initial starting value.

Once occurrences of extreme shifts are located, one-by-one we apply our selection of MA models to the output of $DS2_1$, iteratively across a range of window period values. At each iteration, we measure the effect of the MA model on the magnitude and duration of the extreme shift. We also record the window period value that results in the move reaching a size below the threshold needed to be considered large. We also verify that the duration of the move (i.e., the time between initial move and retracement) is not significantly altered.

Figure 4.5 (left) shows the distribution of extreme parallel shift sizes for the test workloads (in terms of standard deviations above the mean for each workload in question). The results of applying the 90th percentile window period value are shown in Figure 4.5 (right) (note the independent y-axis scales). An extreme parallelism shift is considered one that 1) is larger than eight standard deviations in magnitude above the mean for a particular workload, and 2) retraced to its starting range within four time steps/epochs.

By setting the threshold at eight standard deviations, we are focusing on exceptionally rare and extreme events that are highly unlikely to occur under normal circumstances, allowing for a more targeted analysis of the system's response to these exceptional events. Such a high threshold ensures that the identified parallelism shifts are truly anomalous and not just random fluctuations or minor deviations from the average behaviour.

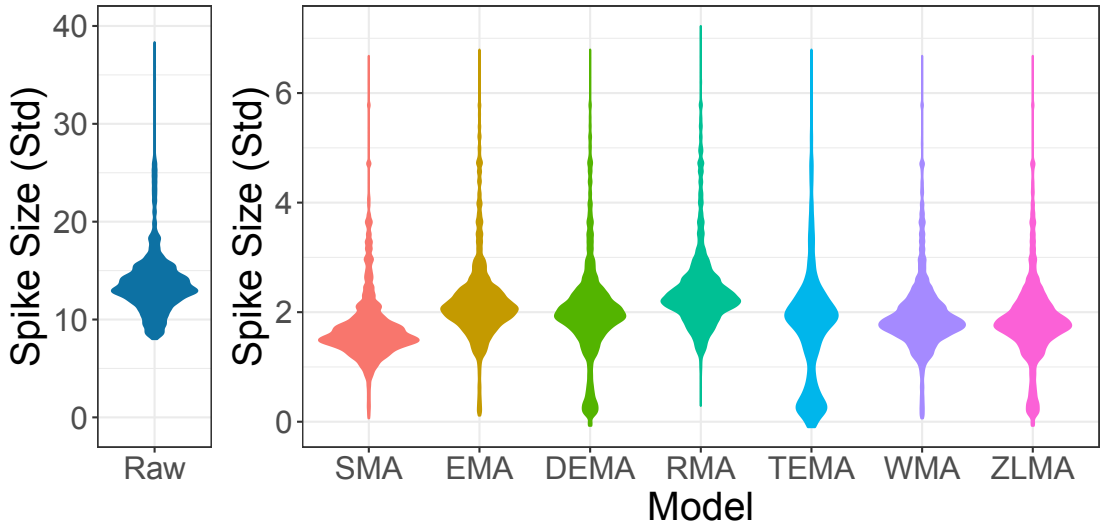


Fig. 4.5 Distribution of Extreme Parallelism Shifts for raw workload traces (left) and for each MA model (right).

The retracement window of four time steps/epochs is chosen to identify parallelism shifts that are quickly determined to be "incorrect" or unjustified by the autoscaler. In these cases, the autoscaler initially proposes an extreme change in parallelism but soon realises that this decision is not based on any legitimate or defensible reason. As a result, it quickly retraces the parallelism level back to its original range within a short period. By focusing on extreme parallelism shifts that are quickly retracted, this definition helps to identify instances where the autoscaler's behaviour may be erratic or unreliable, leading to unnecessary resource allocation changes and potentially impacting the system's performance and stability.

It's important to note that the specific values of eight standard deviations and four time steps/epochs are not absolute and may vary depending on the characteristics of the workload, the system architecture, and the goals of the analysis. The choice of eight standard deviations as the threshold magnitude and four time steps/epochs as the retracement window is based on the assumption that shifts of that size are truly extreme, and that a well-functioning autoscaler should be able to recognise and correct its unjustified decisions within a relatively short period. However, both values may be adjusted based on the characteristics of the autoscaler, the workload, and the system under study. Researchers may fine-tune these values to better capture the autoscaler's responsiveness and the timescales at which unjustified parallelism shifts are typically retracted, as well as to suit their specific requirements and the observed behaviour of the system under study.

We can see that the data consisted of parallelism shifts ranging between eight and 40 standard deviations, with the vast majority ranging between ten and 20 standard deviations.

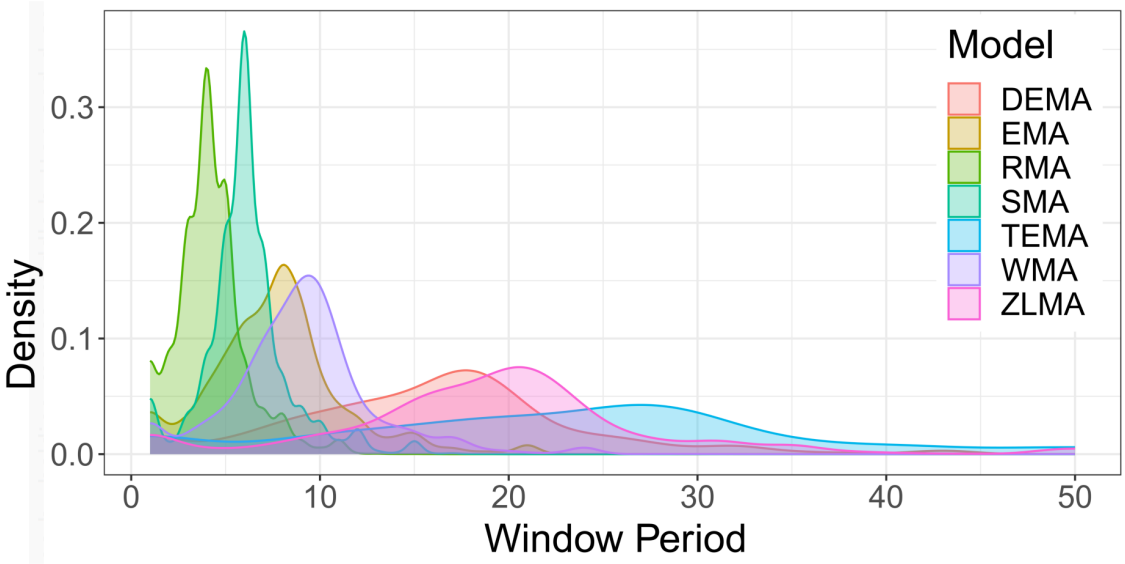


Fig. 4.6 Distribution of smallest window periods to mitigate extreme parallelism shifts.

Figure 4.6 displays the distribution of the minimum window period size required by each moving average model to successfully mitigate the impact of extreme parallelism shifts, considering all tested workload traces. The mitigation point is defined as the moment when the magnitude of the parallelism shift is reduced to within three standard deviations above the mean parallelism change for that specific workload.

In other words, for each moving average model and workload combination, we determine the smallest window period value that effectively brings the extreme parallelism shift under control. A shift is considered "under control" or "mitigated" when its size falls below a threshold of three standard deviations above the average parallelism change observed in that particular workload.

By presenting the distribution of these minimum window period values, Figure 4.6 provides insights into the relative effectiveness of each moving average model in handling extreme parallelism shifts across various workload scenarios. Models with smaller minimum window period values can be seen as more efficient in mitigating extreme shifts, as they require shorter window sizes to bring the shifts under control.

Figure 4.7 shows; 1) the average values for smoothness and accuracy at each window value, when applied across the complete set of workload traces and 2) the inherent trade-off between smoothness and accuracy. Table 4.2 shows, for each model, the average values for smoothness, accuracy and bias, when applied using the specific window period identified

Table 4.2 Smoothness & Accuracy: η_{90} Window Period

MA Model	η_{90} Window Period	Smoothness	Accuracy	Bias
RMA[192]	6	0.48	2.61	0.05
SMA[155]	9	0.42	2.71	-0.01
EMA[95]	12	0.46	2.76	0.00
WMA[155]	14	0.40	2.71	-0.02
DEMA[155]	25	0.43	3.03	-0.10
ZLMA[45]	29	0.44	3.26	-0.19
TEMA[166]	38	0.43	3.27	-0.21

as the 90th percentile value from the distributions shown in Figure 4.6. Figure 4.5 shows the distribution of extreme parallel shift sizes across the workloads, after a particular model has been applied using the 90th percentile window period.

Figures 4.5 and 4.6 allow us to analyse the general effectiveness of an MA model in mitigating extreme one-off parallelism shifts. Furthermore it helps reveal any potential underlying relationships that exist between that general level of effectiveness and both the window size values and the magnitude of the extreme shifts.

With increasing window period values, so to does the MA models' effectiveness in mitigating the effects of extreme parallelism shifts. Increasing a model's window period increases the smoothing effect it has, as it assigns less weight to - and becomes less influenced by - each suggested parallelism value. This is supported by Figure 4.6 which shows that once we reach a window period value of 38, all moving average models tested successfully mitigated the impact of at least 90% of the extreme parallelism shifts found in the test dataset (we refer to this as the 90th percentile window period value). The models reduced the size of these shifts, bringing them down to a level that falls within three standard deviations from the average parallelism change. In simpler terms, the moving average models were able to handle and control the vast majority (90% or more) of the extreme parallelism shifts present in the test data, by employing a window period value of 38 or lower. By applying these models, the magnitude of the shifts was decreased to a range that is considered reasonably close to the typical or average parallelism change observed in the dataset. This range is defined as within three standard deviations from the mean. The ability of all models to mitigate at least 90% of the extreme shifts demonstrates their effectiveness in managing and smoothing out the impact of these significant and sudden changes in parallelism. By reducing the magnitude of the shifts to a level closer to the average, the models help to

stabilize the system's behaviour and minimize the potential disruptions caused by extreme parallelism changes.

Table 4.2 displays the 90th percentile window period value required to achieve this 90% extreme parallelism shift mitigation for each of the MA models tested. We observe that this threshold window period value varies significantly across the range of MA models (ranging from six to 38).

The results of applying the 90th percentile window period value are shown in Figure 4.5 (note the independent y-axis scales). All models are able to reduce the magnitudes of the extreme shifts to below two standard deviations, save for the EMA and RMA models which result in magnitudes between 2 to 2.5 standard deviations.

Figure 4.7 shows a clear trade-off between smoothness and accuracy, both at the window period level and the model itself. The rankings for accuracy are the inverse of those for smoothness. The model that displays the best accuracy, displays the worst smoothness (TEMA) and vice versa (RMA). WMA holds the average rank for both smoothness and accuracy. However, there appear to be some models that offer a better trade-off between their accuracy and smoothness for a particular window value.

The WMA model offers the most beneficial accuracy/smoothness trade-off at the level where 90% of shifts were mitigated. The WMA model displays the best level of smoothness out of the models tested, while displaying the joint second best level of accuracy. WMA displayed no significant level of bias. Furthermore, the far right plot in Figure 4.7 shows that the convex accuracy-smoothness "trade-off frontier" for the WMA model dominates all other models tested, confirming the WMA to be the model of choice.

We note that the accuracy-smoothness "trade-off frontier" for the TEMA, ZLMA and DEMA models appear to the right of, and thus are dominated by, the other models tested. It is these same three models which appear in Table 4.2 with the three highest η_{90} values (i.e., the smallest window value which successfully mitigates at least 90% of the undesirable extreme parallelism shifts observed). They are also the only three models which display any noticeable bias, with values of -0.10, -0.19 and -0.21, between 2-4x the magnitude of that shown by the RMA model (0.05) as the next closest value. The above results indicate the TEMA, ZLMA and DEMA models to be a poor choice, relative to the other models tested.

4.5.2 Volatility

We now explore volatility with respect to the following metrics: *a)* the average level of parallelism, *b)* the average size of a change in parallelism, *c)* the cumulative size of changes in parallelism, relative to the average level of parallelism observed to that point in time, *d)* the total number of changes in the level of parallelism.

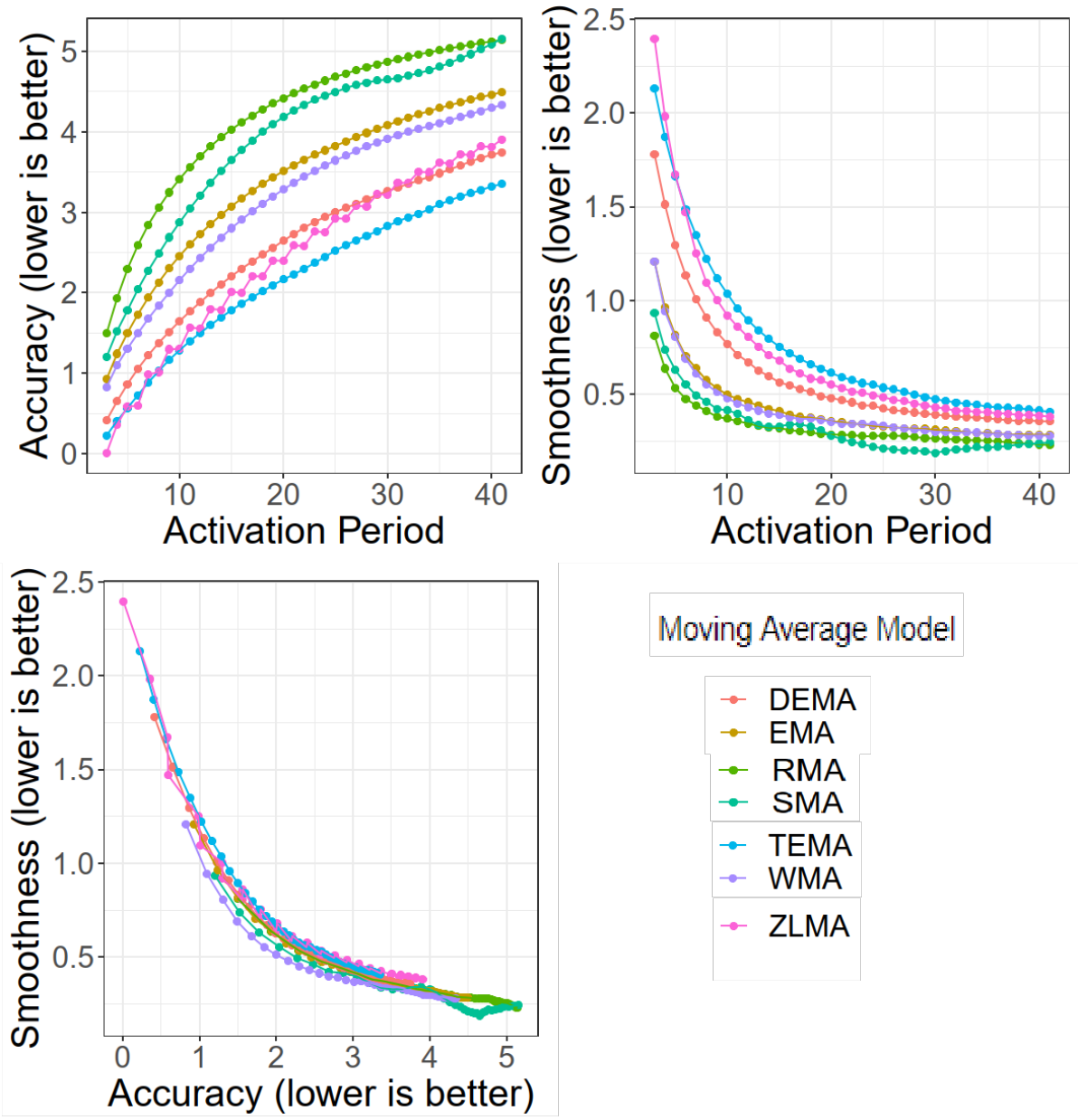


Fig. 4.7 Performance of moving average models with respect to accuracy (top left) and smoothness (top right) and the accuracy-smoothness trade-off (bottom left).

Table 4.3 Volatility of autoscaler metrics for each MA model.

	Mean (%)				Median (%)			
MA Model	No. of Chng	Cum Rel Size Chng	Avg Para	Avg Chng Size	No. of Chng	Cum Rel Size Chng	Avg Para	Avg Chng Size
DEMA [155]	6.36	6.6	0.23	1.77	4.62	5.15	0.19	1.69
DS2 [108]	9.5	75.2	14.28	57.21	8.76	41.07	10.2	42.2
EMA [95]	8.26	8.17	0.12	1.31	5.34	5.56	0.11	1.23
RMA [192]	9.86	9.14	2.67	1.61	5.87	5.68	1.41	1.57
TEMA [166]	7.86	8.61	0.66	1.74	6.56	7.14	0.39	1.84
WMA [155]	8.21	8.02	0.13	1.5	5.87	5.84	0.1	1.55
ZLMA [45]	9.34	9.51	0.39	4.13	8.02	7.19	0.35	3.96

Stratified random sampling was carried out with each strata representing a specific workload generation function. This provided an even mix of 400 traces across all four workload generation functions, creating a sample that best reflects the population being studied (Section 4.4.1). The above metrics of interest were recorded and their volatilities calculated.

Figure 4.8 shows the values recorded for each of the metrics of interest, as we vary activation / window periods. For brevity we show the results for a single workload, whose behaviour is representative of that across all experiments. Therefore, Figure 4.8's sub-figures represent a particular metric's values, for a particular model, for a specific workload trace, when applied using a range of differing activation/window period values, (e.g. Figure 4.8 (top left) shows the observed average parallelism values for each model, for a specific workload trace, when applied using differing activation/window period values, across a range of 2 to 100). Our MA models exhibit preferable behaviour with respect to average parallelism, average parallelism change size, and cumulative parallelism change size. DS2 shows fluctuating values of average parallelism as its activation period increases, highlighting a challenge in correctly parametrising this value in the presence of fluctuating workloads. In contrast, DS2 shows improved behaviour for the number of parallelism changes enacted.

Figure 4.9 shows the distribution of each metric of interest for each MA model, across all 400 workload traces. DS2 is more dispersed than any of the MA models. For the number of parallelism changes, we see a central tendency higher than the MA models. For the other three metrics, there appears to be no central tendency, signifying a close to random relationship across differing workloads.

Table 4.3 shows the mean and median volatility for each model and metric, across all workloads tested. Each metric's volatility distribution exhibits positive skew, with median values smaller than means. The values for DS2's metrics' volatility, as implied by Figure 4.9, are confirmed to be significantly higher than those for any of the MA models tested (save for the number of parallelism changes metric volatility). This signifies that all models are successful in reducing the volatility, and therefore uncertainty, regarding the behaviour of

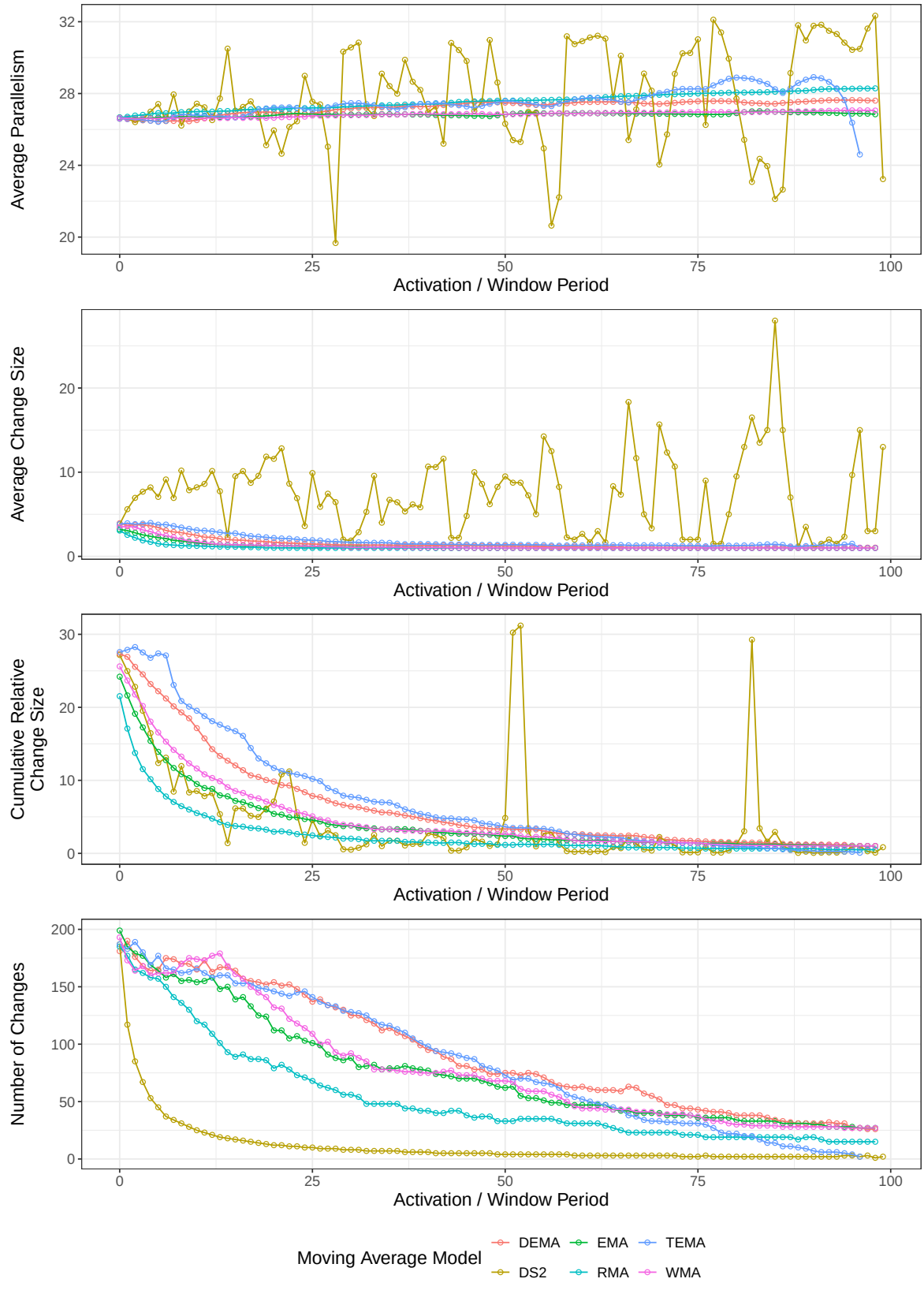


Fig. 4.8 Impact of activation/window period, with respect to the average parallelism (top left), average change size (top right), the number of parallelism changes (bottom left) and cumulative relative change size (bottom right).

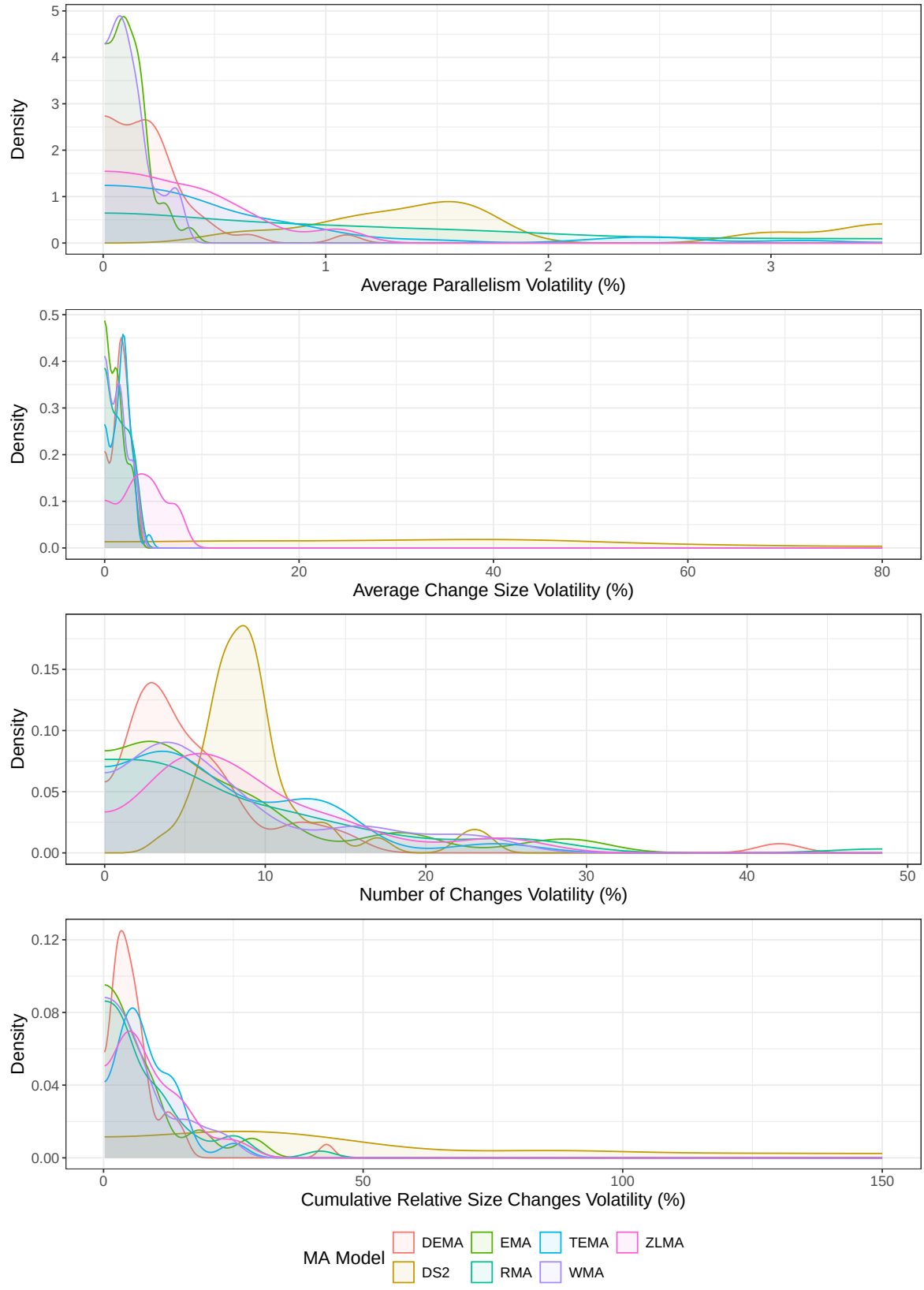


Fig. 4.9 Distribution of metrics volatility for all models, with respect to the volatility of average parallelism (top left), average change size (top right), the number of parallelism changes (bottom left) and cumulative relative change size (bottom right).

DS2 across differing workloads, with possible fundamental differences in their underlying characteristics and generative process.

The EMA and WMA models are most successful in reducing the volatility of 1) the average parallelism level and 2) the average change size. The DEMA models appears most successful with regards to 1) the numbers of parallelism changes enacted and 2) the cumulative relative size of those changes across a workload trace. It is noted that the RMA model, while performing similarly to other models across most metrics, its value for the volatility of the average parallelism value, while significantly lower than that for DS2, remains significantly higher than the other MA models.

4.5.3 Summary of Findings

We have examined several moving average models, as applied to the parallelism values proposed by a state-of-the-art autoscaler. Relating our findings back to the categories of failure in Section 4.2:

Extreme Parallelism Shifts: We have demonstrated the potential for moving average models to mitigate over 90% of extreme parallelism shifts (Section 4.5.1)

Volatility: We show that moving average models can reduce volatility of suggested parallelism values (Section 4.5.2). In particular, we confirm our expectations (Section 4.4.2) that smoother models are most successful in mitigating volatility.

Under- and over-provisioning: Our Bias measure (Table 4.2) shows our approaches realise these benefits without causing under-/over-provisioning. The parallelism suggestions from the models are consistent with the instantaneous parallelism suggestions from our state-of-the-art autoscaler.

Further work is required to explore the remaining failure behaviours of autoscalers within a production environment. In particular, re-sharding and the transfer of snapshot state can lead to reconfigurations which cannot be completed within the decision interval of the autoscaler. Further experimentation will allow us to understand the interplay between model performance and reconfiguration time. These implications are explained further in Section 3.3.

4.6 Replication Package

We are committed to ensure the broader community benefit from our work. We provide a replication package [1] comprising; *a)* 40,000 workload traces collected through our experimentation, *b)* computational notebooks supporting the analysis of the measurements, *c)* executable scripts and a practitioners' annex to support the replication of these findings across other workloads and target systems.

4.7 Conclusion

We have demonstrated the strong potential for using moving average models to enhance the robustness and accuracy of autoscaler decision-making in the face of workload variability. Through empirical evaluation, we demonstrate additional classes of undesirable autoscaler behaviour, beyond those captured by commonly-used SASO properties. We show applying MA models to autoscaler output can:

1. Successfully mitigate instances of undesirable, extreme parallelism shifts, with different models displaying differing characteristics and levels of effectiveness.

2. Reduce the number of rescaling decisions and their latency and throughput impacts, through a) reduced oscillations, b) mitigating costly-to-enact extreme parallelism shifts.

3. Significantly reduce autoscaler uncertainty and volatility when faced with dynamic and variable incoming workloads.

Our work highlights several interesting avenues of future research. Firstly, we see the potential for an ensemble approach to provide the distinct benefits of several MA models. Secondly, we see potential to incorporate these measures to inform other operating parameters for autoscalers. For example, to increase contingency multipliers for scaling decisions in periods of uncertainty, or introduce MA models dynamically when dynamic and variable incoming workloads are detected. Finally, we may incorporate estimates on rescaling duration (e.g. accounting for state size) as a mechanism to optimise the choice of window size.

[1] https://www.github.com/MattForshaw/DEBS23_WindowingAndWeighting/

Chapter 5

Measuring Streaming System Robustness Using Non-parametric Goodness-of-Fit Tests

Related Publications

The work presented in this chapter appeared in the following publication:
Jamieson, S., Forshaw, M. (2023). Measuring Streaming System Robustness Using Non-parametric Goodness-of-Fit Tests. In: Gilly, K., Thomas, N. (eds) Computer Performance Engineering. EPEW 2022. Lecture Notes in Computer Science, vol 13659. Springer, Cham. DOI: 10.1007/978-3-031-25049-1_1. URL: https://doi.org/10.1007/978-3-031-25049-1_1.

5.1 Introduction

Due to the high penalties for system performance degradation, streaming system operators may seek to accept lower average performance in return for an increase in robustness to a wide variety of operating conditions [48]. This raises the question of how robustness is to be quantified and measured. We evaluate several robustness metrics based on non-parametric goodness-of-fit tests. Our aim is to identify emerging best practice which could then inform future performance analysis research considering robustness quantification in distributed systems.

A robust system is one that can maintain performance under a wide variety of operating conditions, or in the face of various uncertainties in the operating environment. Robustness

is the persistence of certain specified features, despite the presence of perturbations in the system's environment [48].

Our work is most closely related to [48], which presents a robustness metric based upon the Kolmogorov-Smirnov (KS) statistic, formalised and presented in Section 5.2.1, (although we go on to include analysis of multiple, additional metrics). The suitability of KS arises from the fact that it can be used in any analytical model for which the Cumulative Distribution Functions (CDFs) of the disturbance and performance can be estimated. It can also be applied to the ECDFs of actual system data. Given a set of performance observations, if the system is robust, the ECDF of the performance metric under "normal" conditions $F(x)$, should be very similar to the ECDF of the performance metric with perturbations applied $F^*(x)$. The closer the functions agree, the more robust the system.

To analyse the persistence of relevant specified features, e.g., system latency, data must be collected across a range of environments, i.e., situations whereby certain disturbances are applied and those without. The results can then be compared and conclusions drawn as to the magnitude of effect each disturbance has on the chosen performance metric. When measuring data from complex systems, one often has no a priori knowledge regarding the underlying distribution function from which the data originate. To avoid making assumptions as to the underlying distribution, non-parametric tests, as opposed to parametric tests, are most appropriate when comparing two data samples for goodness of fit.

Parametric tests, such as the t-test or Analysis of Variance (ANOVA), make specific assumptions about the underlying distribution of the data. These tests typically assume that the data follows a normal distribution, and that the variance of the groups being compared is equal. When these assumptions are not met, the results of parametric tests can be unreliable or misleading.

Non-parametric tests, such as the KS test or the CVM test, do not make strict assumptions about the distribution of the data. These tests are based on the ranks or relative positions of the observations rather than their actual values. As a result, non-parametric tests are more robust and flexible when the data does not follow a specific distribution or when the assumptions of parametric tests are violated.

Although appropriate for use in our case, the KS statistic also displays certain characteristics which suggest it may not be the optimal approach in certain circumstances. For example, while the KS statistic is robust to outliers, it is known that the values of the KS statistic itself are not equally sensitive to movements along its own probability distribution. The KS statistic is most sensitive to situations where ECDFs differ in a global fashion near the centre of the distribution, but is relatively insensitive to differences at the extremes [8]. This is due to values converging to zero and one, at the extremes of the ECDFs.

This poses two problems; firstly, real-world computing systems are known to experience "heavy-tailed" distributions of metric values when experiencing performance disturbances or degradation [84]. Secondly, observations near the right tail of the ECDF, by definition, represent more extreme values of the underlying metric than those observed nearer the median. More extreme values result in more extreme impacts upon the system. The insensitivity of the KS statistic in the tail regions goes against this logic by implicitly giving less weight to these extreme observations than they deserve.

There exist numerous other non-parametric goodness-of-fit tests from the same family of tests as the KS test, each created to better cope with different sample sizes or circumstances. We propose incorporating a number of these statistical tests, along with the KS statistic on the basis that the resulting values either include additional information not provided by the KS statistic alone, or overcome a particular weakness or assumption within the other tests.

Our analysis shows that different tests will produce differing results to provide a more complete picture of the (1) level and (2) characteristics of a streaming system's robustness. A streaming system's robustness can be characterised by its reaction to various levels of disturbance, rather than to a single level. We relate a robust system to one that shows a size and rate of degradation that is in line with the size and rate of change of the applied disturbances. If this is the case then the test statistic value will follow a predictable, near-linear path in the face of changes in the size of disturbance. Vice versa we relate a sensitive, non-robust system to one that shows large, possibly non-linear changes in the chosen test statistic in relation to the applied disturbance.

5.2 Summary of Test Statistics

Within this section, we formalise notation for each test statistic under evaluation.

5.2.1 Kolmogorov-Smirnov

The KS test [168] belongs to the supremum class of Empirical Distribution Function (EDF) statistics and is based on the cumulative probability distribution of data. This class of statistics is based on the largest vertical difference between the two distributions being compared. The *one-sample KS test* is used to compare a sample distribution with a reference probability distribution to decide if the sample comes from a population with that specific distribution.

The one-sample KS test δ quantifies the distance between the sample distribution and reference distribution, using Equation 5.1.

$$\delta = \sup_{-\infty<x<\infty} \left| F(x) - F^*(x) \right| \tag{5.1}$$

where: δ = KS statistic
$F(x)$ = Reference ECDF (performance metric under "normal" conditions)
$F^*(x)$ = Sample ECDF (performance metric with perturbations applied)

The *two-sample KS test* is used to compare two sample distributions to decide whether they were drawn from the same (but unknown) population. calculates the distance between the ECDFs of the two samples, at each unit of the scale, (e.g. at each time point for latency distributions), where the value of interest is the maximum vertical distance identified. The value of interest is the maximum vertical distance between the two CDFs because it quantifies the largest discrepancy between the observed and expected distribution values over the entire range of data. This maximum distance is a critical measure because it captures the most significant deviation between the two distributions being compared, regardless of where that deviation occurs.

For the two-sample KS test, the null hypothesis is that both sample distributions come from the same underlying distribution. If the KS statistic $\delta_{nn'}$, shown in Equation 5.2, is larger than the critical value, the null hypothesis is rejected.

$$\delta_{nn'} = \sqrt{\frac{nn'}{n+n'}} \sup_x |F_n(x) - F_{n'}(x)| \tag{5.2}$$

where: $\delta_{nn'}$ = Two-sample KS statistic
$F_n(x)$ = ECDF of data size n
$F_{n'}(x)$ = ECDF of data size n'

Due to the insensitivity of the KS statistic to differences in the tails of a distribution, a weighting function ψ is applied to adjust it, creating the WKS δ_w, in order to assign higher weights to larger values in the right tail of the ECDF [48]. This is shown in Equation 5.3.

$$\delta_w = \delta_{nn'} \psi(x) \tag{5.3}$$

where: δ_w = Weighted KS statistic
$\psi(x)$ = Weighting function

The weighting function applied needs to account for the underestimation of the KS statistic in the right tail, but not overly so. Previous experimentation carried out by [48] suggests choosing the function shown in Equation 5.4 to apply in the computation of δ_w. This function is based on the fact that the quantity $F(x)(1-F(x))$ is at a maximum where $F(x) = 0.5$, and the weight is only applied when δ occurs to the right of the median, otherwise $\psi(x) = 1$.

$$\psi(x) = -\ln(F(x)(1-F(x))) \tag{5.4}$$

5.2.2 Cramér-von Mises

The *Cramér-von Mises* (CVM) statistic [136] belongs to the quadratic class of EDF statistics. This class of statistics is based on the squared difference between the two distributions being compared. Specifically, the CVM defined as the integrated squared difference between the EDFs of the two samples being compared. If the value of T, calculated as shown in Equation 5.5, is larger than the tabulated values, the hypothesis that the two samples come from the same distribution can be rejected.

$$T = \left[\frac{NM}{(N+M)}\right] \int_{-\infty}^{\infty} [F_N(x) - G_M(x)]^2 \, dH_{N+M}(x) \tag{5.5}$$

where:
T = Cramér-von Mises statistic
$F_N(x)$ = ECDF for the first sample
$G_M(x)$ = ECDF for the second sample
$H_{N+M}(x)$ = ECDF for the two samples together
N = first sample size
M = second sample size

While the KS statistic can be insensitive to distributions with similar means that display multiple *cross-overs* in their ECDFs, the CVM statistic retains its sensitivity in this situation as it measures the sum of squared differences, not just the differences themselves. However, the CVM statistic is insensitive to differences in the tails of the distributions, in similar fashion to the KS statistic.

5.2.3 Anderson-Darling

The *Anderson-Darling* (AD) test [8] belongs to the quadratic class of EDF statistics and was developed originally for detecting sample distributions' departure from normality. The two-sample AD test states the null hypothesis that the two samples come from the same

continuous distribution and is rejected if the AD statistic is larger than the corresponding critical value. The two-way AD statistic is calculated as per Equation 5.6.

$$AD = \frac{1}{mn} \sum_{i=1}^{n+m} (N_i Z_{(n+m-ni)})^2 \frac{1}{iZ_{(n+m-1)}} \tag{5.6}$$

where: AD = Anderson-Darling statistic
Z_{n+m} = ECDF for the two samples together
n = first sample size
m = second sample size

Analysis carried out by [49] compared the AD test with the KS test and found the AD test more powerful when comparing two distributions that vary in shift only, in scale only, in symmetry only, or that have the same mean and standard deviation but differ in the tail ends only.

5.2.4 Epps-Singleton

The *Epps-Singleton* (ES) test [50] compares the empirical characteristic functions (ECFs) of two samples, rather than the observed distributions. The p-value of the ES test gives the probability of falsely rejecting the null hypothesis that both samples have been drawn from the same population. The ECF is defined as the Fourier transform of the distribution function, as shown in Equation 5.7. The ECF is known to completely characterise any distribution and can be used to derive its moments, so contains more information than just a single measure such as the mean or variance.

$$\phi_{n_k}(t) = \int_{-\infty}^{\infty} e^{itx} dF_{n_k}(x) = n_k^{-1} \sum_{m=1}^{n_k} e^{itX_{km}} \tag{5.7}$$

where: ϕ = ECF
t = a real number
i = $\sqrt{-1}$
n = sample size
X_{km} = m-th observation in sample k

[50] compared the ES test with the AD, CVM and KS tests resulting in the following conclusions [71]; (a) Apply the ES test if using discrete data, (b) the KS test usually has a lower power than the ES test if using continuous data, and (c) sometimes the AD and CVM tests can have a higher power than the ES test.

5.3 Methodology

This experiment aims to measure changes in streaming system performance, in the face of disturbances, i.e., changes in the characteristics of the incoming workloads. A range of workloads were generated and run using Flink [30]. Latency metrics were selected as our proxy with which to represent performance, or any degradation thereof. Having recorded the latency observations across the different workload simulation runs, the results were analysed and transformed to allow an indication of how robust the system was to the changing workloads.

5.3.1 System

The evaluation was carried out using Apache Flink 1.4.1 [30]. Experiments ran on an X570 AORUS ULTRA with AMD Ryzen 9 3900× 12-Core Processor and 64GB RAM, running Microsoft Windows 10 Pro. Data collection was integrated with SLF4J in a Docker container.

The workloads were run through a three-step *Word Count* topology with three unique operators (as shown in Fig. 5.1). In our topology, the *Source Operators* emit sentences of a particular word-length by randomly selecting from a provided set of English words. The sentences are sent to the second stage of the topology, where the *Splitter Operators* split them into individual words. These words are, in turn, sent forward to the third stage of the topology, the *Count Operators*. At this stage, a count is kept of the number of times each word was encountered.

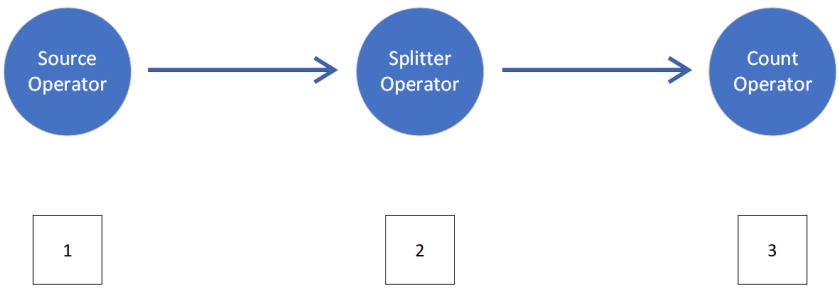


Fig. 5.1 Three-Step Word Count Topology

5.3.2 Summary of Workloads

We design our workloads by specifying the workload generation function and the input variables which are to be varied to simulate the necessary disturbances. We create a range of simulations by varying the inputs to the chosen workload function, using a *one-factor-at-a-time* (OFAT) experiment design approach [55].

The workload generation function is a sinusoidal wave (Equation 3.2a) where a is amplitude, ω is angular frequency, ϕ is phase shift and δ is vertical shift.

$$Y(t) = A\sin(\omega t + \varphi) \qquad \text{(3.2a revisited)}$$

where: t = time
A = amplitude
ω = angular frequency
φ = phase shift

	Input Chosen to Vary			
Input Variable	Source Operators	Amplitude	Sentence Size	Angular Frequency
Source Operators	{5,6...15}	10	10	10
Splitter Operators	10	10	10	10
Count Operators	10	10	10	10
Amplitude	10,000	$\{1000k \mid k \in \{5,6...15\}\}$	10,000	10,000
Angular Frequency	0.006978	0.006978	0.006978	$\{\frac{1}{120k}2\pi \mid k \in \{5,6...15\}\}$
Sentence Size	100	100	$\{10k \mid k \in \{5,6...15\}\}$	100
Vertical Shift	20,000	20,000	20,000	20,000

Table 5.1 OFAT workload design

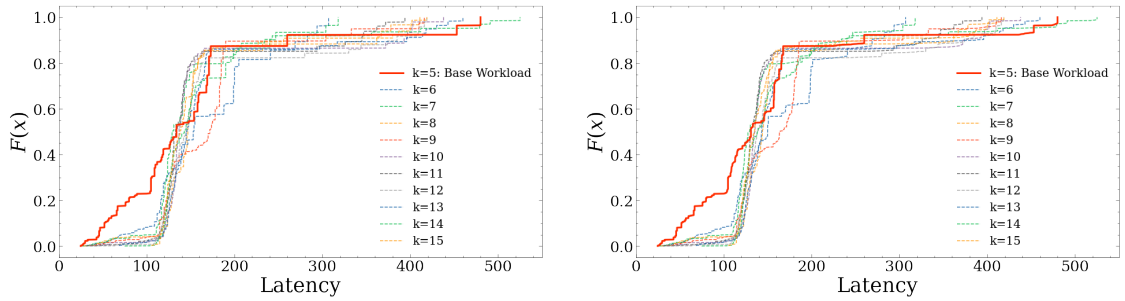
The input variables to be varied are: (1) The number of source operators generating the incoming workload, (2) the amplitude of the sine wave workload generation function vs its vertical distance, (3) The angular frequency of the sine wave workload generation function, and (4) The sentence size on the Nexmark Word Count [183] workload logic. The workload variations, driven by the OFAT approach, are shown in Table 5.1. 11, equally spaced levels were chosen for each input value to be varied, resulting in 44 simulation runs carried out in total.

Our choice of a sine wave to form the basis of our workload generation function is due to its simplicity and the ease with which its parameters can be varied. The sine wave function is also well understood, and the effects of changing its parameters are well documented. These characteristics make it an excellent choice for simulating disturbances in a controlled manner.

5.4 Results and Discussion

5.4.1 Source Operator Variability

Figure 5.2 shows the ECDFs of the max (5.2a), 99th (5.2b), 95th (5.2c) and 50th (5.2d) percentile latencies for each Flink simulation, for which the number of workload source operators were varied. There are noticeable deviations in the left tail for the three highest percentile latencies while, although all latency percentile ECDFs have extreme right tails, significant deviations between the ECDFs within those extreme right tails only occur for the max and 99th percentile latencies.



(a) Max Latency

(b) 99th Percentile Latency

(c) 95th Percentile Latency

(d) 50th Percentile Latency

Fig. 5.2 ECDFs of percentile latency values with variable source operators

Furthermore, as the latency percentiles increase, the extreme right tail values account for larger proportions of the observed values, i.e. the maximum values for each percentile are not significantly different, but we see a significantly different number of observations at or near those maximum values. For example, using a 250ms latency value as the threshold, we observe proportions of total observations above that value of; circa 1% for 50th percentile latencies, circa 3-4% for 95th percentile latencies, circa 10-12% for 99th percentile latencies, and circa 15% for max percentile latencies.

Figure 5.2 also shows that the higher percentile latencies (max and 99th) display significantly more instances of ECDF cross-overs, than do the ECDFs for the lower percentile

Statistic	Max					p99					p95					p50				
	WKS	KS	AD	ES	CVM	WKS	KS	AD	ES	CVM	WKS	KS	AD	ES	CVM	WKS	KS	AD	ES	CVM
WKS	1.0	0.9	0.88	0.35	0.9	1.0	0.91	0.92	0.46	0.94	1.0	1.0	0.95	0.91	0.95	1.0	1.0	0.94	0.82	0.95
KS		1.0	0.94	0.49	0.91		1.0	0.95	0.52	0.92		1.0	0.95	0.91	0.95		1.0	0.94	0.82	0.95
AD			1.0	0.55	0.98			1.0	0.59	0.95			1.0	0.98	1.0			1.0	0.85	1.0
ES				1.0	0.53				1.0	0.58				1.0	0.98				1.0	0.84
CVM					1.0					1.0					1.0					1.0

Table 5.2 Test stat correlation across latency %tiles w. variable source operators

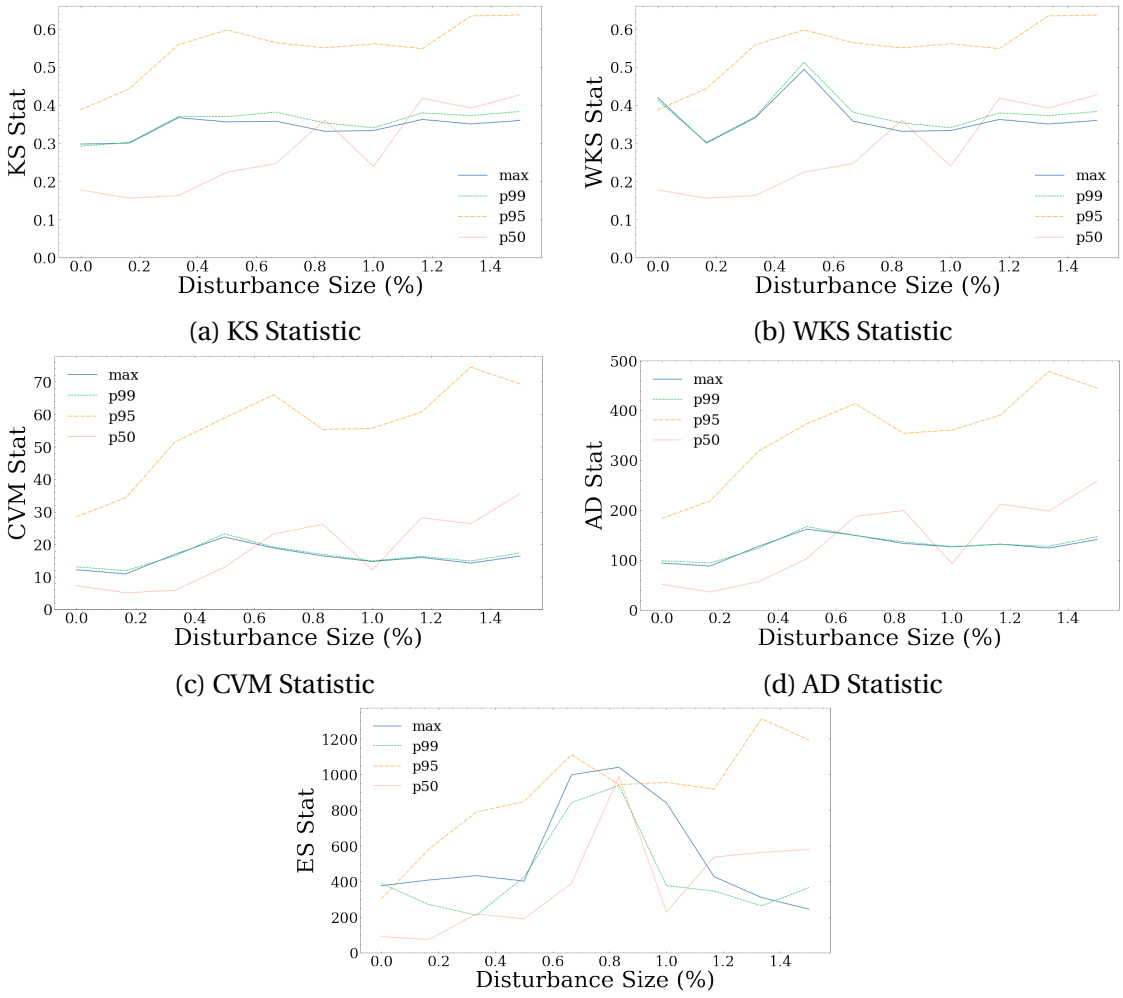
latencies (95th and 50th). As the latency percentile in question increases, it becomes more important to capture any differences in the right tail of the distribution as those differences become larger, along with retaining sensitivity to any instances of ECDF cross-over.

Table 5.2 shows that the 50th and 95th percentile latency results display higher overall levels of inter-statistic correlation than those observed for the max and 99th percentile latencies. This is in keeping with expectations; the more sensitive a statistic is to non-locational, non-global differences in the distributions, (e.g., the more sensitive it is to differences at the extremes or to multiple ECDF cross-overs), the more it should begin to differ, in this case, from other statistics which are less sensitive to these characteristics.

The WKS statistic values, shown in Figure 5.3b, suggest that the system is less robust to changes in the number of source operators, in regard to higher latency percentiles, than do the KS statistic values shown in Figure 5.3a. The WKS statistic values however suggest the system is as equally robust to changes in the number of source operators, in regard to the lower percentile latencies as do the KS statistic values. This appears to show the WKS statistic working as intended to afford more attention to values occurring in the right tails. A system experiencing high values of high percentile latency observations represents a system already under potential stress, or near the bounds of its "stable operating environment" and could be considered more likely to lack robustness.

The CVM and AD statistics (Figures 5.3c and 5.3d), represent the system as being significantly more robust in the max and 99th percentile latencies than for the 95th percentile latency. The WKS statistic (Figure 5.3b) shows a markedly smaller difference in robustness between the 95th and the higher percentile latencies. This is expected behaviour as the CVM statistic is known to be insensitive to deviations at the tails, which is the case here for the higher percentile latencies. This would result in a CVM statistic value which deems the robustness level for the various percentile latencies to be more similar to that shown by the KS statistic (Figure 5.3a), as it does not differentiate as markedly between ECDFs with and without larger deviations in the tails, as does the WKS statistic.

The behaviour of the AD statistic is less expected. The AD statistic is said to retain sensitivity to deviations in distributions, even when those deviations occur in the extreme



(a) KS Statistic

(b) WKS Statistic

(c) CVM Statistic

(d) AD Statistic

(e) ES Statistic

Fig. 5.3 Robustness of System to Variability in Number of Source Operators

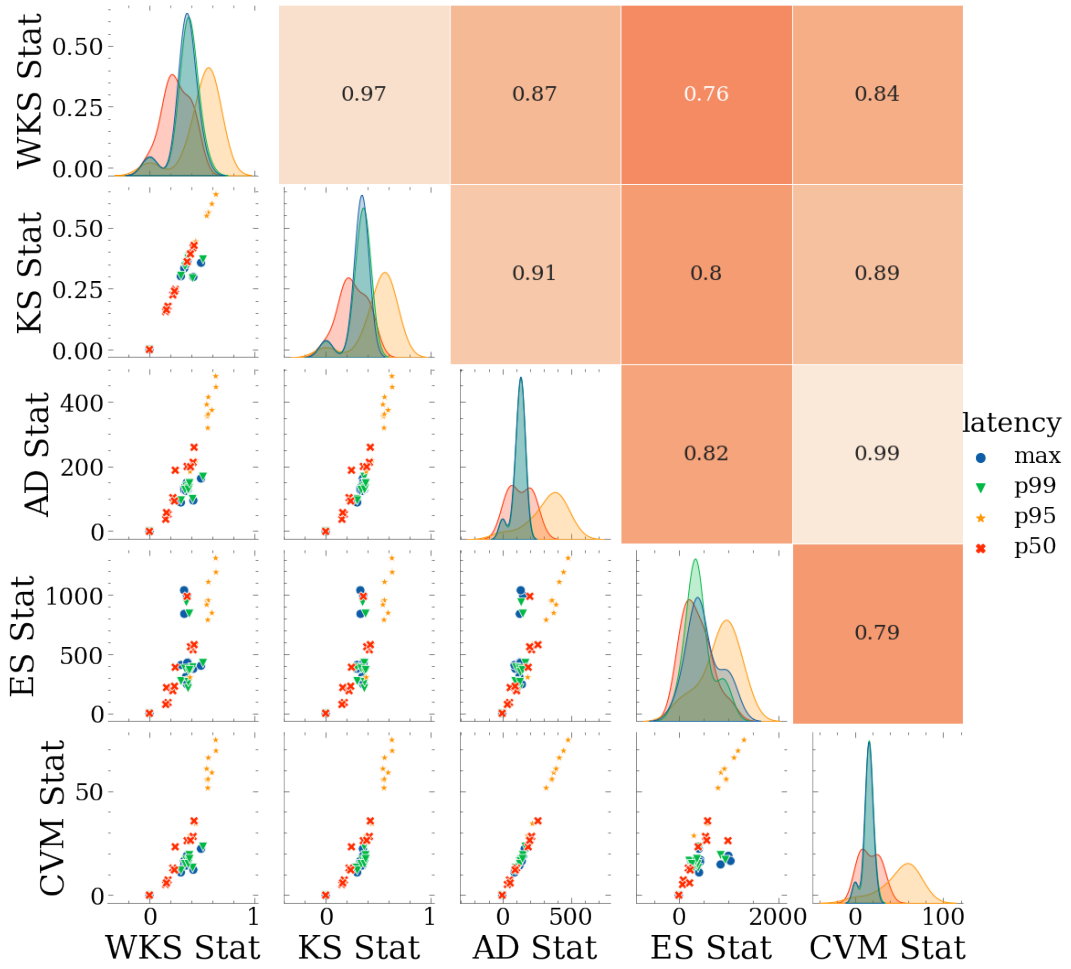
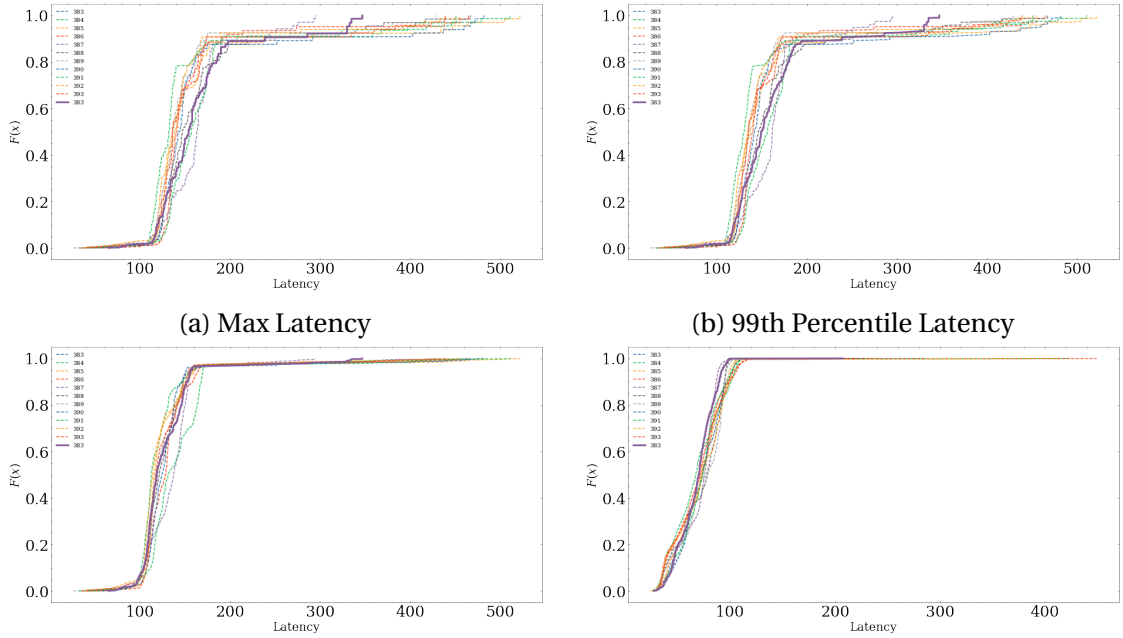


Fig. 5.4 A pair plot comparing the relationship and correlation between various test statistics (TS) across different latency percentiles when the number of source operators is varied, revealing patterns and dependencies among the test statistics.



(a) Max Latency

(b) 99th Percentile Latency

(c) 95th Percentile Latency

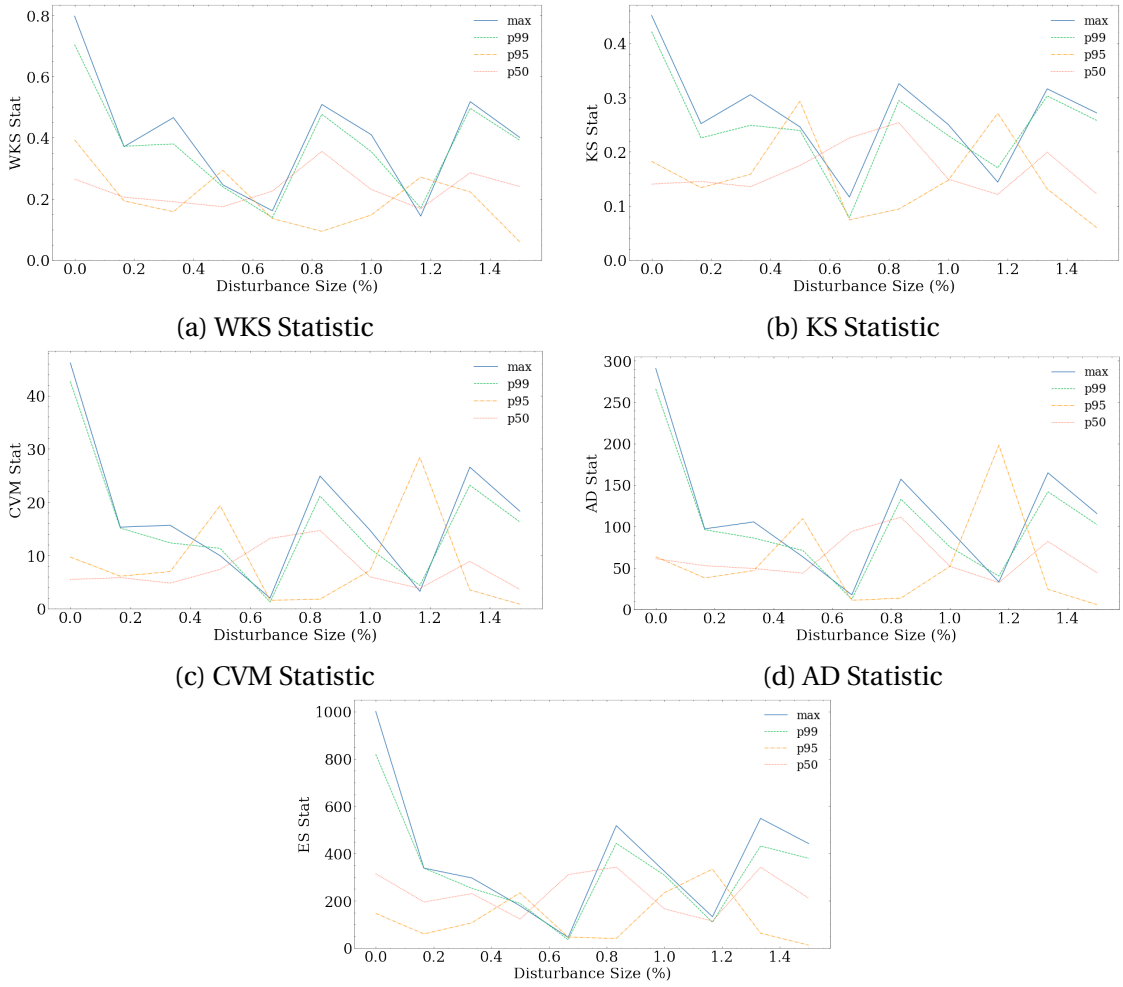
(d) 50th Percentile Latency

Fig. 5.5 ECDFs of percentile latency values with variable frequency

tails. For this reason, one might expect the AD statistic to show a difference in system robustness levels that is more similar to that shown by the WKS statistic, i.e. it would capture the lack of robustness implied by any deviations occurring at the extremes of the ECDFs, and deem the system to be less robust in terms of the max and 99th percentile latencies, therefore shifting their statistic values higher and closer to those of the 95th percentile latency. Considering this, whereby the AD statistic is more similar to the KS than the WKS statistic, we may deem the AD statistic to be potentially under-estimating the sensitivity of the system's high percentile latency distributions to changes in the number of source operators, (i.e. over-estimating the levels of robustness).

The ES statistic suggests an even smaller difference between the system robustness across different percentile latencies, with Figure 5.3e displaying a number of observed instances whereby the max and 99th percentile latency distributions are less robust than the 95th percentile latency distributions to certain disturbances in the number of system source operators. This is in contrast to the KS, WKS, CVM and AD statistics which represent the 95th percentile latency distributions to be less robust than the higher percentile latencies across the entire range of disturbances to the number of source operators within the system. The ES statistic also appears to suggest a substantially lower level of system robustness overall for the higher percentile latencies, when compared to the CVM and AD statistics. Figure 5.3e

displays a non-linear, somewhat erratic relationship for the max, 99th and 50th percentile latencies.



(a) WKS Statistic

(b) KS Statistic

(c) CVM Statistic

(d) AD Statistic

(e) ES Statistic

Fig. 5.6 Robustness of System to Variability in Frequency

Figure 5.3 shows that the various test statistics agree more when considering the robustness of the system for lower percentile latencies compared to higher percentile latencies. This suggests that the statistics may capture differences in the distributions of higher percentile latencies and extreme values differently than when capturing the effects of less extreme values and differences in lower latency percentile distributions.

Most of the test statistics, except for the ES statistic, indicate that the system is more sensitive to initial, smaller disturbances (especially for higher latencies) and becomes less sensitive to larger disturbances. For example, the max and 99th percentile latency test

statistic values reach their maximum level of deviation after disturbance sizes of 75-100% and then level out.

The lack of robustness in higher percentiles to initial disturbances (i.e. disturbances in the lower half of the tested value range) could be because these latency percentiles already represent extreme values observed when the system is under strain. If a system is already strained and operating at the extremes of its "normal" stable range, even a small disturbance can cause a significant reaction and degradation. The system may then become less sensitive to larger disturbances because the performance can only deteriorate so much before the system fails.

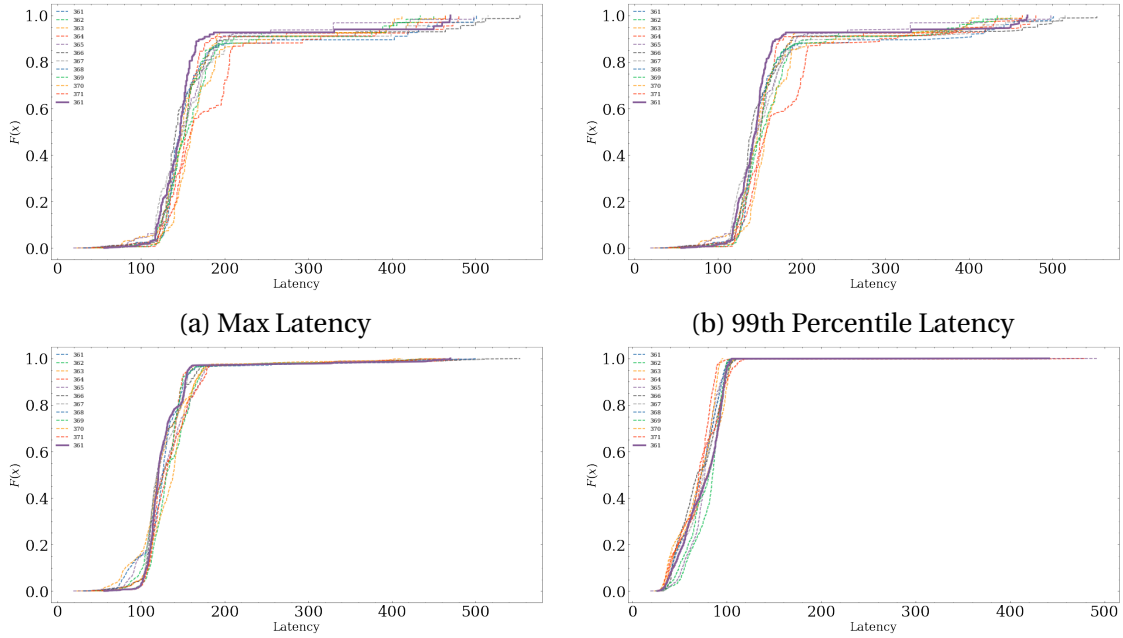
Among the test statistics, the 95th percentile latency appears to be the least robust metric. This might be because the values in the 95th percentile are located relatively deep in the right tail of the distribution, where the underlying characteristics begin to change. For example, Extreme Value Theory (EVT) may govern this region rather than the Central Limit Theorem (CLT), but further investigation is needed to confirm this.

Figure 5.4 shows that the ES statistic generally has a positive, linear relationship with the other statistics when considering the 50th and 95th percentile latencies. However, Table 5.2 indicates that for the max and 99th percentile latencies, the ES statistic values have much lower correlation with the other statistics' values. This may suggest that the ES statistic is more sensitive to differences between extreme right-tail values, as it continues to vary for higher percentile latencies even when the other statistics begin to remain almost static.

5.4.2 Frequency Variability

Similarly to when the number of source operators is varied, when the angular frequency input to the Sinewave workload generation function is varied, the resulting ECDFs, shown in Figure 5.5, display significant deviations in the right tails, when considering the higher percentile latencies (max and 99th). In contrast, however, there appear to be no significant deviations located in the left tail for any of the percentile latencies.

The WKS statistic values, shown in Figure 5.6a, suggest that the system is less robust to changes in angular frequency across all latency percentiles than do the KS statistic values. Rather than the WKS statistic weighting function affecting the higher percentile latencies' values more noticeably than those of the lower percentile latencies, we see more of a linear, upward shift for all values, as shown in Figure 5.6a versus Figure 5.6b. This linear shift maintains the nature of the relationship between the WKS statistic values and the KS statistic values across the range of angular frequency disturbances tested. This relationship between the test statistic values for the differing latency percentiles across various levels of disturbance in angular frequency appears to hold true across the remaining test statistics



(a) Max Latency

(b) 99th Percentile Latency

(c) 95th Percentile Latency

(d) 50th Percentile Latency

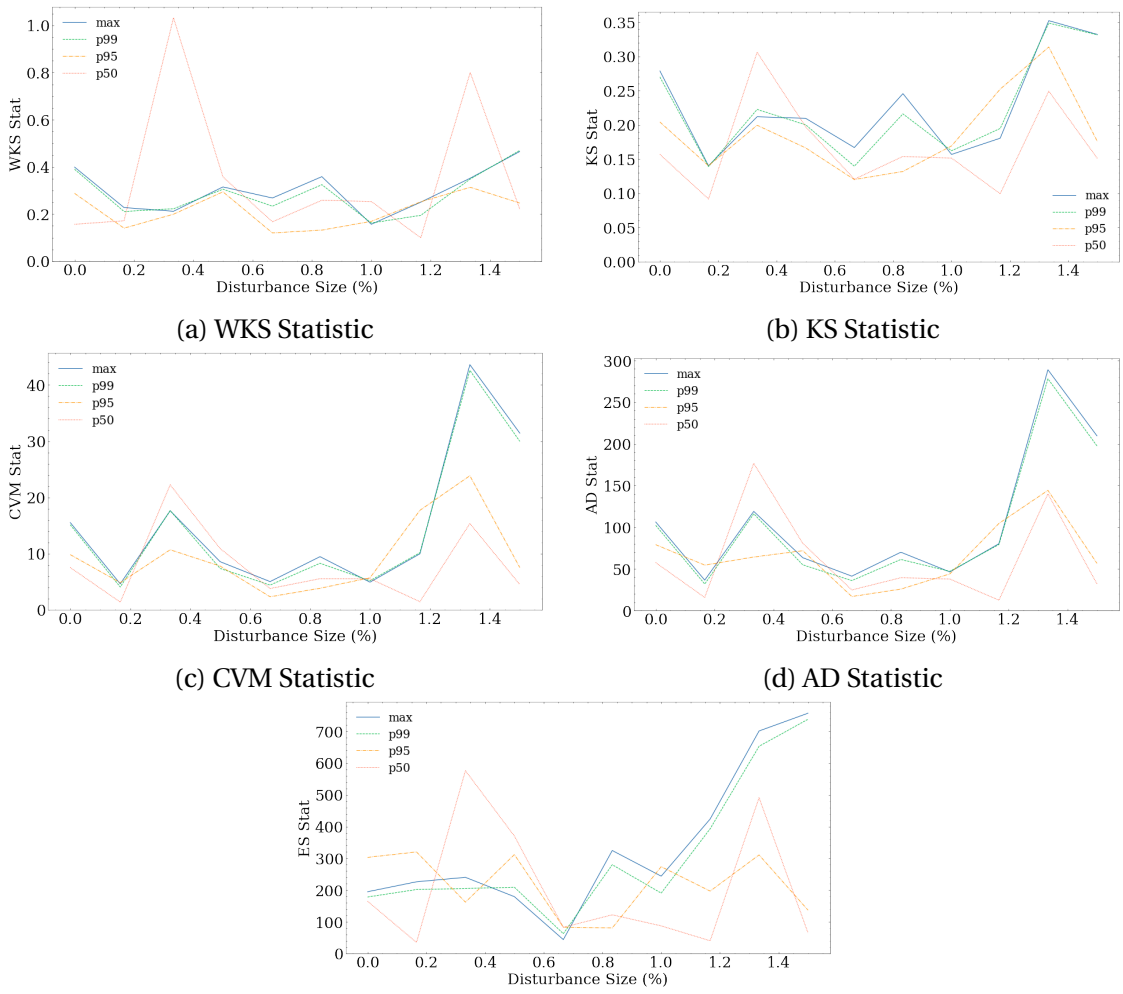
Fig. 5.7 ECDFs of percentile latency values with variable amplitude

(Figures 5.6c 5.6d 5.6e). All statistics suggest that the system is least robust to small changes in the angular frequency when considering the max and 99th percentile latencies. As the size of the disturbance increases, the levels of robustness converge across the different percentile latency values.

5.4.3 Amplitude Variability

We found the correlation between the WKS statistic and other (non-KS) statistics to be significantly lower for the higher percentile latencies when the amplitude of the sine wave workload generation function is varied, compared to when the number of source operators is varied. This appears to manifest due to a larger proportion of the KS statistic values being located to the right of the median of the ECDFs, shown in Figure 5.7, when the amplitude is varied, therefore being subject to the weighting function. In this instance, in general, the distributions differ most to the right of the median, albeit not to an extreme level. The weighting function applied to the KS statistic, described in Equation 5.4, is non-linear so reduces the levels of correlation once applied.

Figure 5.8 shows that when the amplitude is varied, the CVM, AD and ES, shown in Figures 5.8c, 5.8d and 5.8e respectively, exhibit large increases in the test statistic value for the higher percentile latencies (max and 99th) as the disturbance size reaches the maximum



(a) WKS Statistic

(b) KS Statistic

(c) CVM Statistic

(d) AD Statistic

(e) ES Statistic

Fig. 5.8 Robustness of System to Variability in Amplitude

test range. This may signify a disturbance which causes the system to operate near the limit of its stable operating environment.

5.4.4 Sentence Size Variability

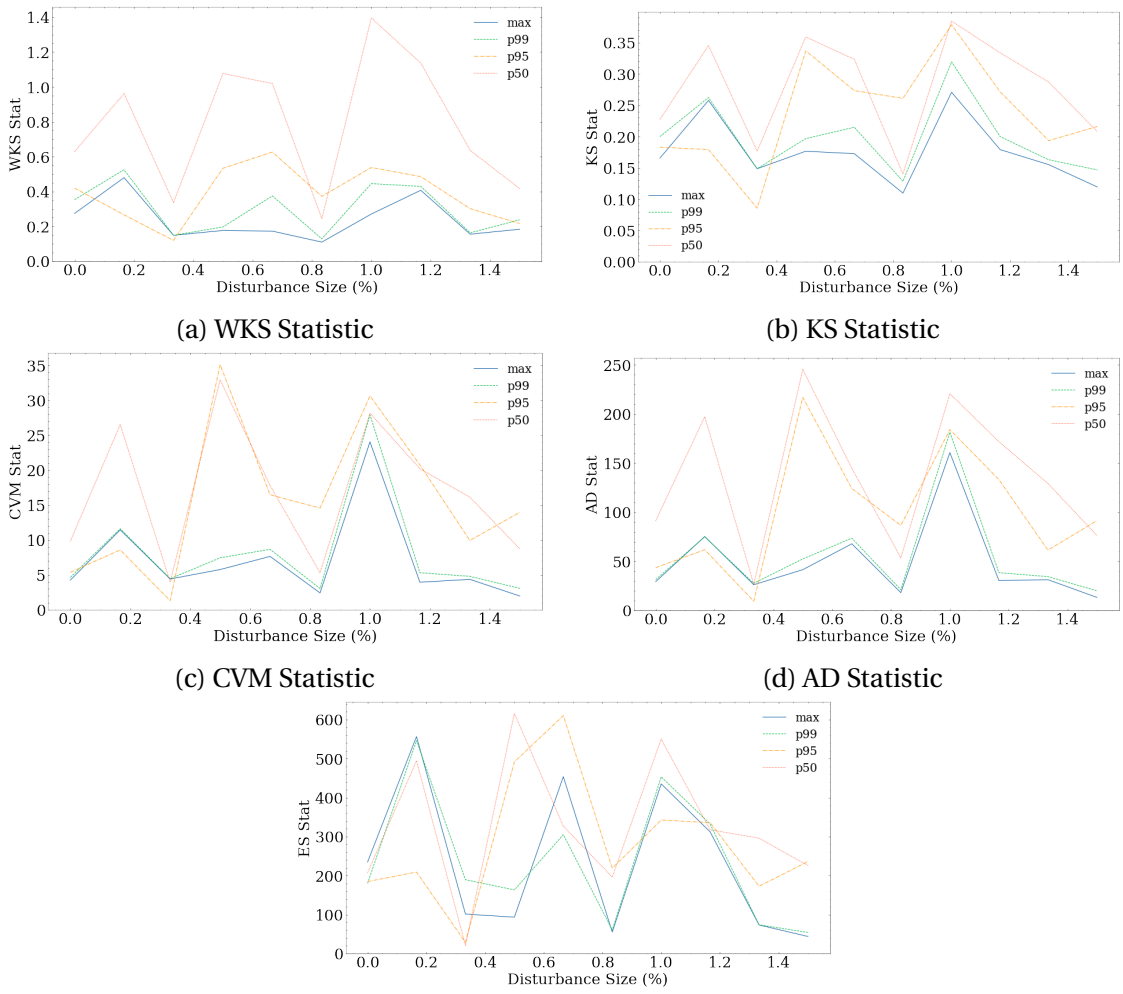
When the sentence size is varied, we observe the resulting ECDFs of the percentile latencies as displaying long right tails, with significant deviations within. Similarly to when the frequency input value is varied, the ECDFs show minimal deviations in the left tail. In this instance, it is noticeable that there also appear significant deviations between the ECDFs not only within the right tails, but within the main bodies of the distributions, albeit occurring to the right of the median. This large number of occurrences of maximum deviations to the right of the median has the same effect on the correlations between the WKS and other (non-KS) statistics, as seen for the higher percentile latencies when varying amplitude, as shown in Table 5.3.

Figure 5.9 appears to suggest a far lower level of system robustness across all the test statistics, and across all percentile latencies, when the workload sentence size is varied, as when compared to the case whereby the number of workload source operators is varied. All statistics display a non-linear relationship between the size of the change in test statistic value and the size of disturbance.

5.4.5 Combined Insight

From these experiments, we have seen how a single metric such as the KS statistic, whether weighted or not, may struggle to properly convey as much insight regarding a streaming system's level of robustness as could be conveyed through the application of multiple goodness-of-fit test statistics. When concerned with less extreme values and lower percentile latencies, the selection of test statistics presented here show strong overall similarities in their behaviour and in their interpretation of the system robustness. However, as the values under scrutiny become more extreme, whether from being contained within a higher percentile latency distribution, or by being located further towards the right tail of that distribution, the test statistics presented begin to diverge in their levels and their relative behaviours.

This should be viewed positively; the different stats tend to differ more when faced with more extreme values within the distribution, which can have greater importance. For example, while the Anderson Darling statistic seemed to show a lack of sensitivity to a situation with multiple ECDF cross overs occurring within the far right tail of a distribution, the ES statistic showed clear indications of a suggested lack of robustness when faced with the same set of inputs.



(a) WKS Statistic

(b) KS Statistic

(c) CVM Statistic

(d) AD Statistic

(e) ES Statistic

Fig. 5.9 Robustness of System to Variability in Sentence Size

Statistic	Max					p99					p95					p50				
	WKS	KS	AD	ES	CVM	WKS	KS	AD	ES	CVM	WKS	KS	AD	ES	CVM	WKS	KS	AD	ES	CVM
WKS	1.0	0.78	0.43	0.76	0.44	1.0	0.87	0.65	0.9	0.63	1.0	0.89	0.81	0.9	0.76	1.0	0.95	0.93	0.86	0.91
KS		1.0	0.81	0.79	0.8		1.0	0.84	0.84	0.82		1.0	0.93	0.8	0.92		1.0	0.93	0.88	0.92
AD			1.0	0.74	0.99			1.0	0.74	0.99			1.0	0.83	0.99			1.0	0.97	0.99
ES				1.0	0.7				1.0	0.73				1.0	0.76				1.0	0.97
CVM					1.0					1.0					1.0					1.0

Table 5.3 Test stat correlation across latency %tiles w. variable sentence size

5.5 Conclusion

In conclusion, this chapter has explored the application of non-parametric goodness-of-fit tests to quantify the robustness of streaming systems in the face of workload perturbations. By employing a diverse set of test statistics, including KS, WKS, CVM, AD, and ES, we have demonstrated that each statistic provides unique insights into the system's behaviour and robustness. The analysis has revealed several key findings that have significant implications for practitioners seeking to design, optimise, and operate reliable and resilient streaming systems.

1. Different test statistics capture different aspects of robustness:
    - Each non-parametric goodness-of-fit test statistic provides a unique perspective on the robustness of a streaming system when faced with disturbances in incoming workload characteristics.
    - The test statistics exhibit varying levels of sensitivity to differences in the distributions of latency percentiles, particularly in the extreme right tails.
    - Practitioners should be aware of these differences and choose the most appropriate test statistics based on their specific requirements and the characteristics of their streaming system.
2. Inter-statistic correlation decreases for higher percentile latencies:
    - As the latency percentile under scrutiny increases (e.g. from 50th to 99th percentile), the level of correlation between the test statistics decreases.
    - This suggests that the test statistics capture the effects of extreme latency values differently, providing complementary insights into the system's robustness.
    - Practitioners should consider using multiple test statistics, especially when analysing higher percentile latencies, to obtain a more comprehensive understanding of the system's behaviour.

3. Extreme tail behaviour significantly impacts robustness measures:
    - The occurrence of observed latency values within the extreme right tails of the distributions greatly influences the divergence between test statistics' results and their interpretation.
    - Test statistics that are more sensitive to extreme tail behaviour, such as the ES statistic, may provide additional insights into the system's robustness that other statistics might not capture.
    - Practitioners should pay close attention to the extreme tail behaviour of latency distributions and use test statistics that are sensitive to these regions when evaluating the robustness of their systems.
4. Robustness to initial disturbances varies across latency percentiles:
    - The system's robustness to initial, smaller disturbances may differ across latency percentiles, with higher percentiles often exhibiting greater sensitivity.
    - This suggests that the system's "normal" stable operating region may be more easily perturbed when considering higher latency percentiles, leading to significant performance degradation.
    - Practitioners should carefully monitor the robustness of their systems across different latency percentiles and take proactive measures to ensure stability, particularly for higher percentiles.
5. Combining multiple test statistics is crucial for a comprehensive robustness assessment:
    - Relying on a single test statistic may provide an incomplete or biased view of a streaming system's robustness.
    - Practitioners should employ a diverse set of non-parametric goodness-of-fit tests to capture different aspects of the system's behaviour and obtain a more holistic understanding of its robustness.
    - By combining insights from multiple test statistics, practitioners can make more informed decisions about system design, optimisation, and operation, ultimately leading to more reliable and resilient streaming services.

Chapter 6

Reasoning Over Streaming System Performance Using Response Surface Methodology

6.1 Introduction

Chapter 6 significantly expands the scope and depth of the analysis presented in Chapter 5. It introduces new test statistics, applies the powerful Response Surface Methodology (RSM) methodology, quantifies the effects of workload parameters, compares model performance and stability, and provides actionable insights for practitioners. This progression from the foundational concepts in Chapter 5 to the advanced modelling and analysis techniques in Chapter 6 demonstrates a logical and comprehensive approach to studying streaming system robustness and performance optimisation.

In the context of streaming systems, understanding and optimising system performance is crucial, as these systems are responsible for processing and analysing massive data streams in real time. As the volume, variety, and velocity of data continue to grow, streaming systems must be designed and tuned to handle the increasing workload while maintaining acceptable performance levels. However, the performance of streaming systems is influenced by a multitude of factors, such as workload characteristics. The manner in which streaming systems' performance degrades in the face of changes in incoming workload characteristics depends not only on the magnitude of those changes, but also on non-linear interaction effects among them. This complexity makes it challenging for system designers and engineers to predict and optimise system behaviour through traditional analytical approaches alone.

In this chapter, we demonstrate that these relationships can be measured and modelled using RSM.

RSM is a collection of mathematical and statistical techniques employed for the modelling and analysis of complex processes in which a response of interest is influenced by several variables, and the objective is to optimise this response. This approach has gained significant traction in various fields, including engineering, manufacturing, and science, due to its ability to optimise processes and uncover hidden relationships between multiple factors and response variables [145, 46]. RSM has been applied in several areas within Computer Science; for example, in OS scheduler tuning [7], formulating security cost models [53] and optimisation of server benchmarks [164]. Closest to our work is that of Gencer *et al.* [66] who apply an RSM solver to a simulation of MapReduce workloads. First- and second-order models are used to map the interplay between configuration parameters and system performance.

RSM is particularly useful in situations where the underlying process is not well understood, or the analytical models are too complex to solve directly. By conducting a series of carefully designed experiments, researchers can develop empirical models that describe the behaviour of a system and identify the optimal operating conditions that maximise or minimise a specific response.

This, however, is not the only aim or use for adopting an RSM approach. Modelling the system's behaviour such that it may be expressed as a function of the input variables, in turn, allows exploration of the process space to help better understand the impact of different factors on the response, including their interactions. The effects of individual factors and their interactions can be explicitly quantified, enabling the model to generate predictions of responses, aiding decision-making and process optimisation. RSM also often involves sequential experimentation, leading to more efficient use of resource and cost savings. By leveraging RSM's ability to model complex relationships between input factors and response variables, we aim to gain insights that will inform the development of more efficient and robust streaming systems.

6.2 Background and Motivation

To address the challenge of modelling the impact of differing workload characteristics, we will employ RSM as a framework. The motivation for conducting this RSM experiment is two-fold.

First, we aim to systematically study the performance degradation of streaming systems under various workloads and quantifying the effects of different parameters on system

latency. By fitting a response surface model to the experimental data, we can capture the complex relationship between the input factors and system latency, providing valuable insights into the behaviour of streaming systems under diverse operating conditions. The level of performance degradation will be measured using a selection of non-parametric goodness-of-fit tests, based on various distance measures and test-statistics, applying them to the ECDFs of recorded system latency values, across a range of selected percentile thresholds. We focus on both model performance and model stability; performance measured using a range of relevant score metrics, and stability, measured using bootstrapped confidence intervals of a range of model values (e.g. factor coefficients, model score metrics, etc.).

Second, RSM offers an efficient and effective means to explore the input factor space. The experimental designs employed in RSM, such as 2^kF, CCD, and BBD, allow researchers to estimate the main effects and interactions between factors with relative efficiency regarding the required number of experimental runs. This efficiency is particularly relevant in the context of streaming systems, where conducting large-scale experiments can be time-consuming and resource-intensive.

6.2.1 RSM

Objectives of RSM in Computer Science

- **Modelling System Performance:** Establish quantitative relationships between system input variables and outputs or performance metrics.
- **Optimisation:** Identify the settings of system parameters that optimise a certain performance measure.
- **Sensitivity Analysis:** Understand the influence of each input variable on the output, facilitating better control and tuning of the system.

Key Components of RSM

1. **Design of Experiments (DoE):** Systematic method to plan experiments so that the data obtained can be analysed to yield valid and objective conclusions. Common designs include 2^kFs, CCDs, and BBDs.
2. **Response Surface Models:** Mathematical models that approximate the relationships between input variables and the targeted responses. Typically, polynomial models are used (linear, quadratic, or cubic depending on system complexity).

3. **Optimisation Techniques:** Techniques such as gradient ascent/descent, Newton-Raphson methods, or even genetic algorithms used to find optimal settings of input variables.

4. **Validation:** Statistical methods to validate the model predictions against new data, ensuring the model's reliability and robustness.

Steps in Applying RSM

1. **Problem Definition:** Identify the key inputs and output metrics of the system. Formulate the problem in terms of these variables.

2. **Experimental Design:** Choose an appropriate design strategy that balances the comprehensiveness of the study with resource constraints.

3. **Data Collection:** Conduct the experiments as per the design, collecting data on input variables and system performance.

4. **Model Estimation:** Use regression analysis to estimate the coefficients of the polynomial model.

5. **Model Analysis:** Analyse the model to assess the significance of variables, interaction effects, and the overall fit of the model.

6. **Optimisation and Validation:** Use the model to predict optimal settings of the input variables and validate these findings through additional experiments or cross-validation techniques.

6.2.2 Applications of RSM in Computer Science

- **Performance Tuning:** Optimise the performance of algorithms or systems by tuning hyperparameters or system configurations.

- **Cloud Computing:** Optimise resource allocation strategies in cloud environments to balance performance with cost.

- **Software Engineering:** Improve software quality and performance by systematically varying development and runtime parameters.

- **Networking:** Optimise network configurations and protocols to improve throughput and latency.

6.3 Methodology

The methodology for our experiment consists of a number of steps that provide a structured approach, tailored to our specific RSM experiment design and objectives. The main steps to be carried out are as follows:

1. Objective Definition: Our objectives are to understand and model the performance degradation of our distributed stream processing system under varying RSM experiment designs, workload characteristics and generative functions, latency percentiles measurements, and goodness-of-fit tests. Subsequently, analyse and measure both the performance and stability of the generated models.

   (a) Investigate the relationships between input factors and the response variables.

   (b) Estimate the main effects and interactions between factors.

   (c) Measure the model performance metrics and relevant model coefficients, metrics, statistics and bootstrapped confidence intervals.

   (d) Compare and contrast the results and outputs.

2. Response Variable: Determine the response variable used to measure the performance of the streaming system. We choose as our response variables a number of non-parametric goodness-of-fit test statistics, introduced in Section 5.2, along with two additions, the Kullback-Leibler Divergence (KLD) and Wasserstein Distance (WD) (presented in Section 6.3.1).

3. Workloads and Factors: Define the workload model and select the characteristics and input variables that may affect the system's performance and the chosen response variable. For our workloads we have chosen the (1) sine wave and (2) envelope-guided process workloads (presented in Section 3.2.4). For our factors we have chosen (1) The number of source operators generating the incoming workload, (2) the amplitude of the sine wave workload generation function vs its vertical distance, and (3) the sentence size on the Nexmark Word Count workload logic.

4. Experimental design: Select an appropriate experimental design for the experiment, given our research objectives, factors, levels, response variable, and any experimental constraints; one that can efficiently and effectively explore the relationships between the factors and our response variable. Our experiment will involve the use of the 2^kF, CCD and BBD (presented in Section 6.3.3).

5. Experimental Runs: Carry out the experiments, adjusting the factor level values according to each chosen design and design matrix. Measure and record the system latency for each experimental run, grouped by percentile thresholds (50th, 95th, 99th and max) (presented in Section 6.3.4).

6. Response Surface Model: Our experiment will look to capture the relationships between the factors and the response variable using a linear regression model testing first-order models with interaction terms (presented in Section 6.3.5). We opt specifically for this model due to its simplicity and interpretability, which is vital when the goal is to derive actionable insights.

7. Model Performance and Stability: Measure the model performance using predictions made by the fitted model. We will apply a bootstrapping method using *sampling with replacement* to run 2000 iterations for each goodness-of-fit test, latency percentile, workload model and experiment design combination. Each iteration will involve generating a sample of input data, fitting a linear regression model to that data, and recording all values and metrics of interest (e.g. model coefficients, performance metrics)

8. Model Analysis and Ranking: Rank the models based on accuracy and predictive power and stability of the fitted model.

6.3.1 Response Variable

Within this section, we formalise notation for each test statistic under evaluation as our response variable. The KS (Equation 5.2), WKS (Equation 5.3), CVM (Equation 5.5), AD (Equation 5.6), and ES (Equation 5.7) statistics are previously introduced in Section 5.2.1.

In this chapter we use two additional tests which were made known to the author after Chapter 5 was concluded. The two additional test statistics to be analysed that have not been previously introduced are as follows:

1. **KLD:** The KLD [119] is a measure of how different two probability distributions are from each other. It is also known as *relative entropy*, or *information divergence*. The KLD is a non-negative value that measures the amount of information lost when approximating one distribution with another.

   The KLD is calculated as the sum of the element-wise product of the logarithm of the ratio of the probability distributions and the first distribution. In the mathematical

form, it is defined as in Equation 6.1.

$$D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)} \tag{6.1}$$

where: P = first probability distribution
Q = second probability distribution

$P(i)$ represents the probability of an event i occurring in distribution P, and $Q(i)$ represents the probability of the same event occurring in distribution Q. The KLD is calculated by multiplying the difference in probability of each event by the logarithm of that difference, and then summing over all events. The KLD is not symmetric, which means that $D_{KL}(P||Q)$ is not equal to $D_{KL}(Q||P)$. Therefore, it should be noted that it matters which distribution you're taking the expectation with respect to.

It is mostly used as a loss function in machine learning, and in feature selection and extraction to compare the similarity between two probability distributions [139]. It is also used in information theory, where it is a measure of how much information is lost when approximating a true underlying probability distribution with a simpler one [62].

We introduce the KLD due to its lack of symmetry (i.e. $D_{KL}(P||Q) \neq D_{KL}(Q||P)$ in general) and triangle inequality (i.e. $D_{KL}(R||P) \leq D_{KL}(Q||P) + D_{KL}(R||Q)$ does not hold in general). Due to these two characteristics, the KLD is often used in places where directionality is meaningful [70], as it is in this circumstance.

2. **WD:** The WD [110] (also known as the *Kantorovich-Rubinstein metric* or *Earth Mover's Distance*) is a distance function defined between two probability distributions over a given metric space. Given two probability distributions P and Q defined over a metric space M with metric d, the p-th order WD is defined as:

$$W_p(P,Q) = \left(\inf_{\gamma \in \Gamma(P,Q)} \int_{M \times M} d(x,y)^p \, d\gamma(x,y) \right)^{1/p} \tag{6.2}$$

where: $\Gamma(P,Q)$ = Set of all joint distributions $\gamma(x,y)$ on $M \times M$ with marginals P and Q

In simpler terms, γ represents a way to "transport" the mass distributed according to P so that it becomes distributed according to Q. The WD measures the "cost" of the

optimal way to do this, where the cost is defined in terms of the underlying metric d. The infimum (inf) ranges over all possible such transport plans.

We introduce the WD as unlike the KLD it considers both the likelihoods and the distances between various outcome events. These properties make the WD well-suited to domains where an underlying similarity in outcome is more important than exactly matching likelihoods [19].

6.3.2 Workloads and Factors

To demonstrate generality across diverse workload types and time-frames we incorporate two of the three arrival rate processes mentioned in Section 3.2.3 within the workload generation models; a stylised sine wave function and an envelope guided process. We use a three-step Word Count topology, coupled with realistic arrival processes. Source Operators emit sentences of a specified length, the Splitter Operators split them into individual words, and the Count Operators count the number of times each word occurs. We use two classes of arrival process for our evaluation:

1. Sine wave workload as presented in Section 3.2.4.
2. Envelope-guided process as presented in Section 3.2.4.

The input variables to be varied are: (1) The number of source operators generating the incoming workload, (2) the amplitude of the sine wave workload generation function vs its vertical distance, and (3) the sentence size on the Nexmark Word Count workload logic.

6.3.3 Experimental Design

Choosing an appropriate design for the experiment is crucial to ensure its reliability and efficiency [89]. We determine the levels at which each factor will be tested; the choice of factors and levels will affect the design's complexity and the number of experimental runs required [27, 26].

The 2^k Factorial Design (2^kF)

One major goal of the experiment was to determine which *factors* (input parameters) have the greatest effect on the *response* (system performance), and the nature of that effect. We sought to accomplish this with the least amount of simulating necessary by carrying out *factor screening*. Avoiding unnecessary simulations results in reduced experimental

time, reduced computational cost, reduced cost to run the experiments, and allows greater coverage of the design space in areas of particular interest. Factor screening allows the examination of several factors simultaneously and can make analysis easier by reducing dimensionality. It is usually performed by following a *factorial design* approach [197]. These design approaches are able to overcome some key deficiencies as compared to the *One-Factor-At-a-Time (OFAT)* [124].

With the OFAT, if we wish to measure the effect a particular factor has on the response, all other factors are fixed at some set of values while only the value of the factor being measured is varied. Simulation runs would be made at both chosen values for the factor in question, recording how the response reacts to the change. This process would need to be repeated, one factor at a time, for all the factors needing to be examined. As the number of factors rises, this approach quickly becomes intractable. In addition to being inefficient, OFAT methods are unable to measure any interaction effects between the factors.

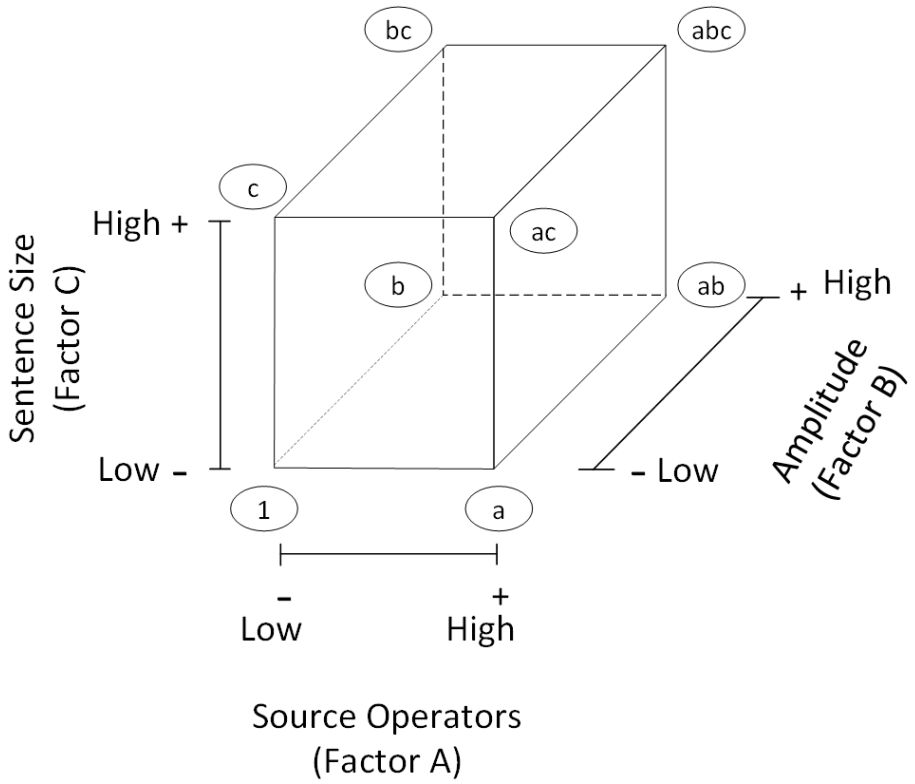


Fig. 6.1 Geometric view of design points.

The 2^k *factorial design* is a special case of the general factorial design, where there are k factors and each factor has only two levels. Each replicate has $2 \times \cdots \times 2 = 2^k$ observations. In the context of factorial designs, a "replicate" refers to the repetition of the entire set of

experimental runs. This repetition is done to obtain a measure of the variability in the response, which helps in assessing the experimental error. Experiments based upon 2^k factorial design make very efficient use of experimental simulations, require relatively few simulations per experimental condition to produce statistical power and also allow the measurement of interaction effects between factors [39]. Furthermore, they provide more information at a similar or lower cost, can find optimal conditions more quickly and allow the effects of a factor to be estimated at several levels, generating conclusions that are valid across a larger range of conditions [137, 141].

2^k factorial design requires that we choose just two levels for each input variable; *high* and *low*. These are called *design points* (e.g. Table 6.1, also represented geometrically in Figure 6.1). It then calls for simulation runs at each of the 2^k possible factor-level combinations, which can be compactly represented in tabular form, also known as the *design matrix* (e.g. Table 6.2). In our example, we have a 2^3 factorial design, with the three factors being the number of source operators, the sine wave amplitude and workload sentence size.

For our factor values, we chose levels that reflected realistic values and also values that resulted in a relative variability in outcome that could be seen as "useful" from an experimental research perspective. If the factor levels were inappropriately disparate in magnitude, the generalisability and value of insight gained, would be compromised. If the factor levels were too similar in magnitudes, any underlying relationship under scrutiny may have been swamped by random noise, or sampling error.

Design Point	Source Operators	Amplitude	Sentence Size
Low	6	5000	100
Base	11	7500	175
High	16	10000	250

Table 6.1 Design points for 2^k Factorial Design.

The Central Composite Design (CCD)

CCD [25] is a commonly used experimental design in response surface methodology and is the most popular class of second-order designs [137]. It is useful for estimating first-order, second-order, and interaction effects in fitting a quadratic surface.

A CCD consists of three main components:

1. Factorial points: These are the points from a 2^kF design, where k is the number of factors. The factorial points allow for the estimation of linear and interaction effects.

Run	Source Operators	Amplitude	Sentence Size
1	−1	−1	−1
2	−1	−1	1
3	−1	1	−1
4	−1	1	1
5	1	−1	−1
6	1	−1	1
7	1	1	−1
8	1	1	1

Table 6.2 Design matrix for 2^k Factorial Design.

2. Axial points (also called star points): These points are added to the factorial design to allow for the estimation of quadratic effects. The axial points are located at a distance α from the center of the design space.

3. Center points: These points are located at the center of the design space and are used to estimate the pure error and to provide a measure of the curvature in the response surface.

The factorial points in a CCD are considered an embedded factorial design because they are a subset of the larger CCD. The embedded factorial design is usually a 2^kF or a fractional 2^kF design, depending on the number of factors and the desired resolution of the experiment. A fractional factorial design is a reduced version of the full factorial design, meaning only a fraction of the runs are used, increasing efficiency but with a trade-off in information.

This design is particularly effective for sequential experimentation. A 2^k factorial or fractional factorial design is initially used, followed by the addition of centre and axial points to estimate curvature and create a quadratic model, if so required. To use the CCD, we start with a 2^k factorial design, where k represents the number of factors. For CCD, the locations of the axial points are typically placed at an *alpha* distance away from the centre point. The value of alpha for coded variables is determined by the number of factors to make the design rotatable or near-rotatable and calculated as $\sqrt{k}$. If a design is rotatable, the variance of the predicted response variable is a function of the distance from the design centre only, and is not dependent at all on the direction. The importance of rotatability arises from the desire to achieve equal precision in prediction at all points equidistant from the design centre. If a design is rotatable, you can predict responses with equal accuracy at all locations that are

the same distance away from the centre of the design. The design points for the CCD are shown in Table 6.3, while the design matrix is shown in Table 6.4.

Design Point	Source Operators	Amplitude	Sentence Size
Low	6	5000	100
Base	11	7500	175
High	16	10000	250
Low Axial	2	7500	175
High Axial	20	7500	175
Low Axial	11	3175	175
High Axial	11	11825	175
Low Axial	11	7500	45
High Axial	11	7500	305
Center	11	7500	175

Table 6.3 Design points for Central Composite Design.

The CCD holds a number of advantages: (1) efficiently building on a 2^k factorial design base allowing for a sequential experimentation approach, (2) being robust to missing data and (3) being rotatable.

CCDs work best when the region of interest is spherical, i.e., when the range of each factor is approximately the same. If the design space is not spherical, it may be disadvantageous to implement a CCD; a Box-Behnken Design (BBD) may be a better choice. A CCD also requires more experimental runs compared to a factorial or fractional factorial design, especially for higher dimensions with many factors.

The Box-Behnken Design (BBD)

BBD [24] allows the coefficients of a quadratic model to be estimated with a reduced number of experimental trials compared to other methods. As a design, it does not contain an embedded factorial nor fractional factorial design and does not require axial points. Rather, it consists of midpoints of edges of the factorial space and centre points.

The design requires that each factor has three levels, generally coded as -1 (for the low level), 0 (for the middle level), and $+1$ (for the high level). The BBD selects points at the midpoints of each edge of the multidimensional factor space and the centre of the space. For each pair of factors, there are three experimental trials: both at their low levels, both at their high levels, or one at its low level and the other at its high level. The remaining factor is

Run	Source Operators	Amplitude	Sentence Size
1	−1	−1	−1
2	−1	−1	1
3	−1	1	−1
4	−1	1	1
5	1	−1	−1
6	1	−1	1
7	1	1	−1
8	1	1	1
9	−1.732	0	0
10	1.732	0	0
11	0	−1.732	0
12	0	1.732	0
13	0	0	−1.732
14	0	0	1.732
15	0	0	0
16	0	0	0

Table 6.4 Design matrix for Central Composite Design.

kept at its middle level. This results in three-factor combinations in blocks of size $k(k-1)$, where k is the number of factors.

Center points are then added to the design matrix, whereby all factors are set to their middle level. These centre points are used in the estimation of the pure error and help identify any curvature in the response surface. The design points for the BBD are shown in Table 6.5, while the design matrix is shown in Table 6.6.

Design Point	Source Operators	Amplitude	Sentence Size
Midpoint	6	7500	175
Midpoint	16	7500	175
Midpoint	11	5000	175
Midpoint	11	10000	175
Midpoint	11	7500	100
Midpoint	11	7500	250
Center	11	7500	175
Center	11	7500	175
Center	11	7500	175

Table 6.5 Design points for Box-Behnken Design.

The advantages of implementing a BBD approach are: (1) fewer experimental runs are required when compared to a three-level, full-factorial design, while allowing for the complete and independent estimation of the first- and second-order coefficients. (2) BBD are nearly rotatable allowing equal accuracy of response prediction at all locations, and (3) their design points are nearly orthogonal, minimising multicollinearity and any negative impact on the accuracy of model parameter estimates.

6.3.4 Experimental Runs

For each experiment design, each workload in the design matrix was simulated and processed through the Flink system (with accompanying replication runs carried out as necessary), and the values of specific system performance metrics were recorded.

Latency values (in milliseconds) were collected which represent the distribution of latency values recorded over each 10-second period, from the source operators to an operator sub-task. Latency tracking marks were emitted from the sensors at 100 milliseconds intervals. The level of granularity used was at the operator level, tracking latency while differentiating between sources but not between sub-tasks. The 50th, 95th, and 99th percentiles, and max latencies were calculated and recorded. These values are captured to represent a range of

Run	Source Operators	Amplitude	Sentence Size
1	0	-1	0
2	0	1	0
3	-1	0	0
4	1	0	0
5	0	0	-1
6	0	0	1
7	-1	-1	0
8	1	-1	0
9	-1	1	0
10	1	1	0
11	0	-1	-1
12	0	1	-1
13	0	-1	1
14	0	1	1
15	0	0	0
16	0	0	0
17	0	0	0

Table 6.6 Design matrix for Box-Behnken Design.

conceptual latency values, from the median latency to the worst-case latency. They represent the values recorded over the 10-second period. The 50th percentile latency is the latency value at which 50% of the latency values recorded over the 10-second period are less than or equal to, and 50% are greater than or equal to. The 95th percentile latency is the latency value at which 95% of the latency values recorded over the 10-second period are less than or equal to, and 5% are greater than or equal to. The 99th percentile latency is the latency value at which 99% of the latency values recorded over the 10-second period are less than or equal to, and 1% are greater than or equal to. The max latency is the maximum latency recorded over the 10-second period. The ECDFs of recorded latency values were then used to calculate the relevant statistics (as presented in Section 6.3.1), as when compared against the baseline workload latency ECDF.

6.3.5 Response Surface Model

Our model specification involves the building of a first-order model with the inclusion of two-way and three-way interaction terms. The model took the form:

$$\begin{aligned} & Y\beta_0 + \beta_1 Src + \beta_2 Amp + \beta_3 Sent \\ & \quad + \beta_{12} SrcAmp + \beta_{13} SrcSent + \beta_{23} AmpSent + \beta_{123} SrcAmpSent \end{aligned} \tag{6.3}$$

where: Src = Number of Source Operators
Amp = Sine wave amplitude
$Sent$ = Sentence Size (words)

For each combination of experiment design, workload generation function, latency percentile and response variable test statistic, the above model was fit to the relevant results data, using a number of approaches. At each iteration through the above combinations, the relevant test statistic results values where run though a 2000 iteration process, carried out using Bootstrapping Random Sampling with Replacement. Each iteration's randomly sampled data was of equal size to the original experiment's design matrix output data. The model was then fit to the sampled data and the relevant values of interest recorded. Various approaches were applied in the model fitting process. Each bootstrapped sample was first used to fit and score the model. Each model was then also subjected to an ANOVA test to investigate the statistical significance of the model as a whole, and that of the individual coefficients. During each of these two fitting and scoring procedures, the following values were measured and recorded:

Model Metrics

Standardised Model Coefficient Values The fitted model coefficients were first standardised by dividing by their relevant standard errors (i.e. their *t-stat* value) then recorded along with their coefficient $p-$values. This was carried out for all three independent input variables (i.e. experiment design factors), including all two-way and three-way interaction terms.

ANOVA Statistics The overall model $F-$stats were collected along with the individual coefficient $F-$stats.

Performance Metrics

Root Mean Squared Error of the Model (RMSE) The RMSE of the model is the explained sum of squares divided by the model's degrees of freedom. It is a measure of the discrepancy between the data and an estimation model. A small RMSE indicates a good fit between the model and the data. The RMSE is calculated as follows:

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n-p}} \tag{6.4}$$

where: y_i = is the actual value for the ith observation.
$\hat{y}_i$ = is the predicted value for the ith observation.
n = the number of observations.
p = the number of parameter estimates, including the constant.

R-Squared The R-squared value is a statistical measure of how close the data are to the fitted regression line; it is the percentage of the response variable variation that is explained by a linear model.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{6.5}$$

where: y_i = is the actual value for the ith observation.
$\hat{y}_i$ = is the predicted value for the ith observation.
$\bar{y}$ = is the mean of the actual values.
n = the number of observations.

Adjusted R-Squared The adjusted R-squared is a modified version of R-squared that has been adjusted for the number of predictors in the model. The adjusted R-squared increases only if the new term improves the model more than would be expected by chance. It decreases when a predictor improves the model by less than expected by chance.

$$R^2_{\text{adjusted}} = 1 - \frac{(1 - R^2)(n-1)}{n-p-1} \tag{6.6}$$

where: R^2 = is the coefficient of determination.
n = is the number of observations.
p = is the number of predictors.

Explained Variance Score (EVS) The explained variance score is a measure of how well the model accounts for the variation within the data. It is calculated as follows:

$$\text{EVS} = 1 - \frac{\text{Var}(y - \hat{y})}{\text{Var}(y)} \tag{6.7}$$

where: y = is the actual value.
$\hat{y}$ = is the predicted value.
Var = represents the variance.

Mean Error (ME) The mean error is the average of the residuals, i.e. the difference between the observed and predicted values. It is calculated as follows:

$$\text{ME} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i) \tag{6.8}$$

where: y_i = is the actual value for the ith observation.
$\hat{y}_i$ = is the predicted value for the ith observation.
n = is the number of observations.

Mean Absolute Error (MAE) The mean absolute error is the average of the absolute residuals, i.e. the absolute difference between the observed and predicted values. It is calculated as follows:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{6.9}$$

where: y_i = is the actual value for the ith observation.
$\hat{y}_i$ = is the predicted value for the ith observation.
n = is the number of observations.

Mean Squared Error (MSE) The mean squared error is the average of the squared residuals, i.e. the squared difference between the observed and predicted values. It is calculated as follows:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{6.10}$$

where: y_i = is the actual value for the ith observation.
$\hat{y}_i$ = is the predicted value for the ith observation.
n = is the number of observations.

Median Abslute Error (MedAE) The median absolute error is the median of the absolute residuals, i.e. the absolute difference between the observed and predicted values. It is calculated as follows:

$$\text{MedAE} = \text{median}(|y_1 - \hat{y}_1|, |y_2 - \hat{y}_2|, \ldots, |y_n - \hat{y}_n|) \tag{6.11}$$

where: y_i = is the actual value for the ith observation.
$\hat{y}_i$ = is the predicted value for the ith observation.
n = is the number of observations.

Mean Absolute Percentage Error (MAPE) The mean absolute percentage error is the average of the absolute percentage residuals, i.e. the absolute difference between the observed and predicted values, divided by the observed value. It is calculated as follows:

$$\text{MAPE} = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{6.12}$$

where: y_i = is the actual value for the ith observation.
$\hat{y}_i$ = is the predicted value for the ith observation.
n = is the number of observations.

6.4 Results and Discussion

We present analysis and discussion of the experiment results, broken down by observations regarding the recorded model performance metrics (Section 6.4.1), and by the recorded model statistics, coefficients and metrics (Section 6.4.2) for each response variable test statistic.

6.4.1 Performance Metrics

Table 6.7 displays the mean values and standard deviations for each performance metric and test statistic. Tables 6.8 to 6.10 display the mean values and standard deviations for each of the same performance metrics and test statistics, grouped by experiment design (2^kF, CCD and BBD respectively). Tables 6.11 to 6.14 display the values, grouped by latency percentile (50th, 95th, 99th percentile, and max latency respectively).

The Best-Performing: ES Statistic

We can see from Table 6.7 that the ES statistic has the best average value for seven out of the nine recorded performance metrics (i.e. EVS, ME, MAE, MSE, MedAE, RSquared and Adj. RSquared) overall, across the full range of experimental runs. The ES statistic also displays the lowest overall standard deviation for the same collection of metrics, versus the other test statistics under consideration.

When grouped by experiment design, we see the ES statistic maintaining its strong average performance metric values across all three experiment designs. The results for the CCD approach (Table 6.9), identify the ES statistic's mean values outperform versus the other test statistics across five of the nine metrics. The 2^kF and BBD approaches report outperformance by the ES statistic across seven of the nine metrics. The standard deviations recorded for the ES statistic's performance metrics show clear outperformance when using the BBD approach, displaying the lowest values across eight of the nine metrics. However, the ES statistic is outperformed in this regard by the KS and KLD statistics when using the 2^kF and CCD approaches respectively. These early findings suggest the ES statistic as an appropriate choice to underpin RSM methodologies. However, to fully understand its

suitability it is necessary to explore how its performance differs across the range of percentile latencies.

The ES statistic's distribution of Adjusted RSquared values, grouped by latency percentile, appears to display the most favourable characteristics of all the test statistics. For the lower percentile latencies (i.e. 50th and 95th), all test statistics analysed appear to display closely similar distributions. However, for the the higher percentiles (i.e. 99th and max), the ES statistic's ECDFs (shown in Figure 6.3) appear significantly more convex to the origin, representing a tighter distribution of values. This can be interpreted as less uncertainty regarding the expected value of the Adjusted RSquared metric value for a model based on the ES statistic, versus the other statistics tested.

The distribution of Adjusted RSquared values for the ES statistic also center around the highest mean value of all the test statistics for the the higher percentiles (shown in Figures 6.3a and 6.3b). The fact the ES statistic displays these divergent characteristics most prominently for the higher percentile latencies is of note as these more extreme latency values are generally of much higher interest and significance to system operators. This would imply that, if modelling performance degradation of a streaming system, when one is most concerned with the more extreme latency percentile values found in the right-tail of a distribution, employing the ES statistic as one's response variable, offers the most performant model, and carries with it the highest confidence and least uncertainty regarding the model's recorded metrics.

Figure 6.2 shows that when using the ES statistic as the response variable, the BBD approach appears to offer the best results and similar characteristics to those mentioned above, namely a tighter distribution and higher mean value of the recorded Adjusted RSquared metric (versus the 2^kF and CCD approach). This further implies that a ES statistic response variable and BBD experiment design offers the optimal combination to performantly model higher percentile latency values and therefore potentially mission-critical levels of streaming system performance degradation.

Based on the results presented, the ES statistic appears to be the best-performing test statistic for several reasons:

- **Performance metrics:** The ES statistic has the best average value for seven out of the nine recorded performance metrics (EVS, ME, MAE, MSE, MedAE, R^2 and Adjusted R^2) overall, across the full range of experimental runs. This indicates that models using the ES statistic as the response variable generally fit the data better and make more accurate predictions compared to models using other test statistics.
- **Consistency across experiment designs:** The ES statistic maintains its strong performance across all three experiment designs (2^kF, CCD, and BBD). This suggests that

the ES statistic's effectiveness is not dependent on a particular experimental setup, making it a robust choice for modelling streaming system performance degradation.

- **Performance across latency percentiles:** The ES statistic's distribution of Adjusted R^2 values, grouped by latency percentile, displays favorable characteristics, particularly for higher percentile latencies (99th and max). For these critical latency values, the ES statistic's ECDFs are more convex to the origin, indicating a tighter distribution of values and less uncertainty regarding the expected value of the Adjusted R^2 metric.

The ES statistic can be considered more "informative" in this context because:

- It captures more relevant information about the differences between the distributions being compared. The ES statistic compares the empirical characteristic functions (ECFs) of two samples, which completely characterise any distribution and can be used to derive its moments. This means the ES statistic incorporates more information about the distributions than just a single measure like the mean or variance.
- It is sensitive to differences in the tails of the distributions, which is crucial when dealing with high percentile latency values that represent extreme and potentially critical performance degradation scenarios.

Models using the ES statistic as the response variable consistently outperform models using other test statistics in terms of performance metrics and stability, suggesting that the ES statistic is more effective at capturing the complex relationships between workload characteristics and system performance.

	Mean							Standard Deviation						
Metric	AD	CVM	ES	KLD	KS	WD	WKS	AD	CVM	ES	KLD	KS	WD	WKS
EVS	0.66	0.66	0.79	0.75	0.74	0.61	0.71	0.24	0.24	0.19	0.2	0.19	0.28	0.21
ME	1.28	1.29	0.91	1.08	1.11	1.38	1.14	0.72	0.73	0.6	0.66	0.64	0.84	0.63
MAE	0.31	0.31	0.24	0.27	0.27	0.32	0.29	0.18	0.18	0.16	0.16	0.16	0.18	0.17
MSE	0.32	0.32	0.19	0.24	0.24	0.36	0.27	0.22	0.22	0.18	0.19	0.18	0.26	0.19
MedAE	0.16	0.15	0.13	0.13	0.14	0.16	0.15	0.19	0.18	0.16	0.16	0.17	0.18	0.18
MAPE	0.49	0.6	0.28	0.16	0.11	0.16	0.16	0.47	0.61	0.28	0.12	0.08	0.13	0.11
RSquared	0.66	0.66	0.79	0.75	0.74	0.61	0.71	0.24	0.24	0.19	0.2	0.19	0.28	0.21
Adj RSquared	0.38	0.38	0.63	0.54	0.53	0.3	0.48	0.43	0.44	0.35	0.36	0.35	0.5	0.38
RMSE Model	1.19	1.19	1.32	1.28	1.27	1.13	1.25	0.23	0.23	0.2	0.2	0.18	0.3	0.2

Table 6.7 Performance Metric Means and Standard Deviations

	Mean							Standard Deviation						
Metric	AD	CVM	ES	KLD	KS	WD	WKS	AD	CVM	ES	KLD	KS	WD	WKS
EVS	0.72	0.72	0.81	0.77	0.79	0.7	0.77	0.25	0.25	0.22	0.23	0.2	0.27	0.22
ME	0.94	0.94	0.72	0.82	0.8	0.98	0.85	0.7	0.71	0.64	0.68	0.63	0.71	0.65
MAE	0.22	0.22	0.17	0.2	0.19	0.23	0.2	0.18	0.18	0.16	0.17	0.16	0.18	0.17
MSE	0.25	0.25	0.17	0.2	0.19	0.27	0.21	0.22	0.23	0.2	0.21	0.18	0.24	0.2
MedAE	0.05	0.04	0.04	0.05	0.04	0.04	0.04	0.15	0.14	0.13	0.14	0.13	0.13	0.14
MAPE	0.22	0.27	0.12	0.1	0.07	0.11	0.1	0.22	0.3	0.15	0.1	0.06	0.12	0.09
RSquared	0.72	0.72	0.81	0.77	0.79	0.7	0.77	0.25	0.25	0.22	0.23	0.2	0.27	0.22
Adj RSquared	0.45	0.44	0.64	0.56	0.58	0.41	0.54	0.49	0.5	0.41	0.44	0.38	0.53	0.43
RMSE Model	1.19	1.18	1.27	1.23	1.26	1.16	1.23	0.22	0.23	0.21	0.22	0.19	0.25	0.21

Table 6.8 Performance Metric Means and Standard Deviations (2^kF)

	Mean							Standard Deviation						
Metric	AD	CVM	ES	KLD	KS	WD	WKS	AD	CVM	ES	KLD	KS	WD	WKS
EVS	0.59	0.6	0.76	0.75	0.69	0.57	0.69	0.22	0.22	0.17	0.16	0.18	0.26	0.19
ME	1.52	1.51	1.07	1.17	1.3	1.62	1.29	0.64	0.64	0.51	0.52	0.54	0.78	0.5
MAE	0.37	0.37	0.29	0.28	0.32	0.37	0.33	0.16	0.16	0.15	0.13	0.15	0.15	0.15
MSE	0.38	0.38	0.23	0.23	0.29	0.41	0.29	0.21	0.21	0.16	0.15	0.17	0.24	0.17
MedAE	0.21	0.21	0.18	0.15	0.19	0.19	0.18	0.19	0.19	0.17	0.14	0.18	0.17	0.18
MAPE	0.76	0.92	0.42	0.19	0.14	0.19	0.19	0.62	0.8	0.34	0.11	0.08	0.13	0.11
RSquared	0.59	0.6	0.76	0.75	0.69	0.57	0.69	0.22	0.22	0.17	0.16	0.18	0.26	0.19
Adj RSquared	0.28	0.29	0.57	0.56	0.46	0.23	0.45	0.39	0.39	0.31	0.28	0.32	0.46	0.33
RMSE Model	1.15	1.15	1.32	1.31	1.25	1.11	1.25	0.23	0.23	0.18	0.16	0.18	0.29	0.18

Table 6.9 Performance Metric Means and Standard Deviations (CCD)

	Mean							Standard Deviation						
Metric	AD	CVM	ES	KLD	KS	WD	WKS	AD	CVM	ES	KLD	KS	WD	WKS
EVS	0.66	0.65	0.81	0.71	0.73	0.58	0.68	0.22	0.23	0.18	0.2	0.19	0.29	0.21
ME	1.37	1.4	0.93	1.25	1.22	1.56	1.29	0.7	0.72	0.59	0.67	0.63	0.85	0.62
MAE	0.34	0.33	0.26	0.32	0.3	0.37	0.34	0.16	0.16	0.13	0.14	0.15	0.16	0.15
MSE	0.32	0.32	0.18	0.27	0.25	0.39	0.3	0.21	0.22	0.17	0.19	0.18	0.27	0.2
MedAE	0.21	0.2	0.18	0.21	0.19	0.24	0.23	0.17	0.16	0.13	0.15	0.15	0.17	0.16
MAPE	0.48	0.59	0.32	0.19	0.12	0.19	0.18	0.3	0.4	0.24	0.11	0.07	0.13	0.1
RSquared	0.66	0.65	0.81	0.71	0.73	0.58	0.68	0.22	0.23	0.18	0.2	0.19	0.29	0.21
Adj RSquared	0.42	0.4	0.68	0.5	0.54	0.27	0.46	0.39	0.4	0.31	0.35	0.32	0.5	0.36
RMSE Model	1.24	1.23	1.38	1.29	1.31	1.13	1.26	0.23	0.24	0.18	0.2	0.18	0.34	0.22

Table 6.10 Performance Metric Means and Standard Deviations (BBD)

	Mean							Standard Deviation						
Metric	AD	CVM	ES	KLD	KS	WD	WKS	AD	CVM	ES	KLD	KS	WD	WKS
EVS	0.7	0.72	0.75	0.83	0.76	0.7	0.78	0.22	0.22	0.19	0.14	0.2	0.22	0.17
ME	1.09	1.07	1.01	0.81	0.98	1.1	0.91	0.57	0.56	0.6	0.5	0.54	0.55	0.49
MAE	0.3	0.29	0.27	0.23	0.27	0.3	0.26	0.18	0.18	0.17	0.15	0.17	0.17	0.17
MSE	0.27	0.26	0.23	0.16	0.23	0.28	0.2	0.21	0.2	0.18	0.13	0.18	0.2	0.16
MedAE	0.16	0.16	0.15	0.14	0.15	0.15	0.16	0.2	0.2	0.18	0.16	0.19	0.19	0.19
MAPE	0.25	0.25	0.24	0.16	0.08	0.11	0.09	0.17	0.18	0.17	0.12	0.06	0.06	0.06
RSquared	0.7	0.72	0.75	0.83	0.76	0.7	0.78	0.22	0.22	0.19	0.14	0.2	0.22	0.17
Adj RSquared	0.46	0.48	0.54	0.7	0.56	0.45	0.61	0.42	0.4	0.35	0.25	0.35	0.42	0.3
RMSE Model	1.24	1.25	1.28	1.36	1.29	1.23	1.32	0.23	0.22	0.19	0.14	0.19	0.23	0.17

Table 6.11 Performance Metric Means and Standard Deviations (50th Percentile Latency)

	Mean							Standard Deviation						
Metric	AD	CVM	ES	KLD	KS	WD	WKS	AD	CVM	ES	KLD	KS	WD	WKS
EVS	0.7	0.7	0.67	0.73	0.76	0.79	0.76	0.22	0.23	0.22	0.2	0.19	0.16	0.17
ME	1.21	1.21	1.28	1.13	1.07	0.95	1.07	0.77	0.77	0.68	0.64	0.7	0.52	0.6
MAE	0.28	0.28	0.31	0.28	0.25	0.26	0.26	0.17	0.17	0.16	0.16	0.16	0.16	0.14
MSE	0.28	0.28	0.3	0.25	0.22	0.2	0.22	0.21	0.21	0.20	0.19	0.17	0.15	0.16
MedAE	0.15	0.15	0.15	0.14	0.13	0.15	0.13	0.17	0.17	0.17	0.16	0.16	0.17	0.15
MAPE	0.39	0.46	0.38	0.24	0.10	0.1	0.14	0.34	0.43	0.28	0.14	0.07	0.07	0.09
RSquared	0.7	0.7	0.67	0.73	0.76	0.79	0.76	0.22	0.23	0.22	0.2	0.19	0.16	0.17
Adj RSquared	0.45	0.46	0.41	0.51	0.57	0.62	0.58	0.4	0.41	0.4	0.35	0.34	0.28	0.3
RMSE Model	1.23	1.23	1.21	1.26	1.3	1.32	1.3	0.21	0.22	0.22	0.19	0.18	0.15	0.16

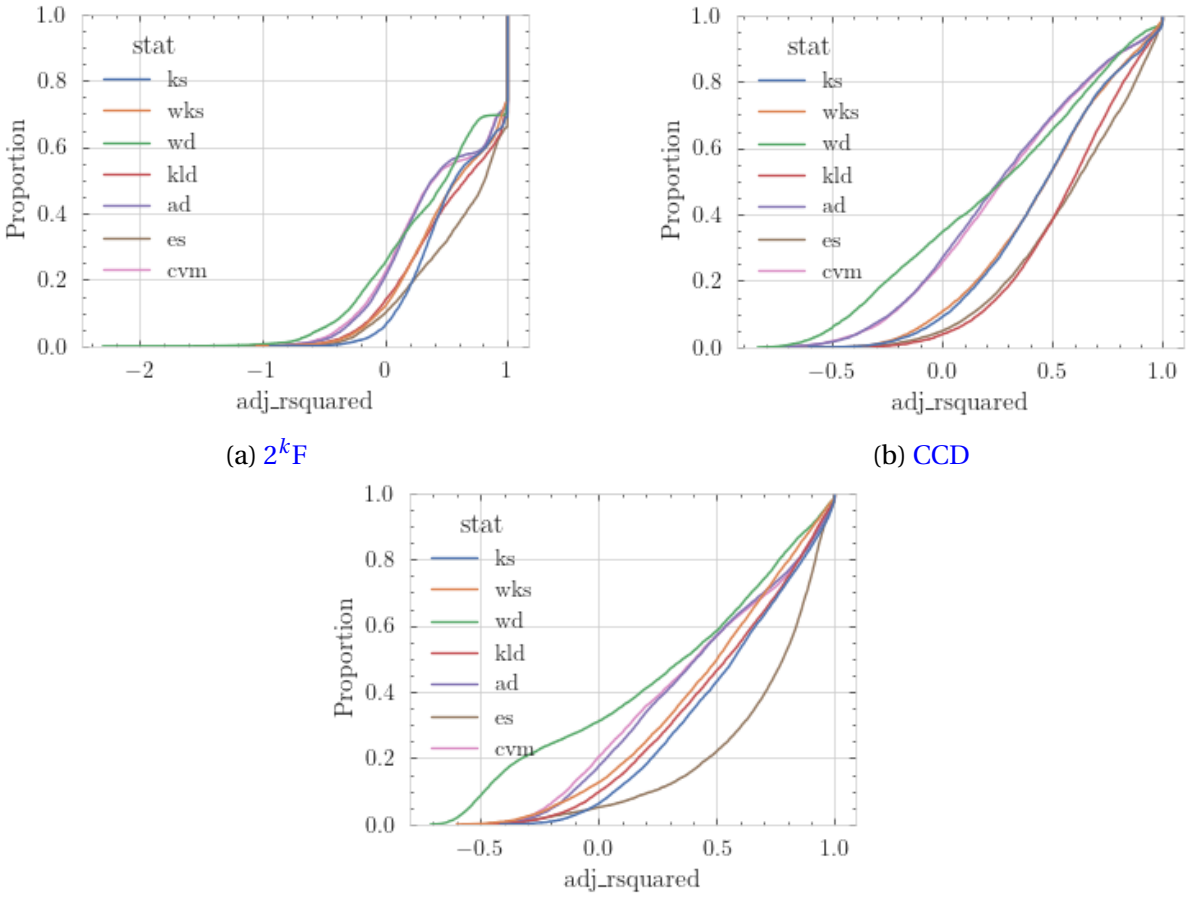
Table 6.12 Performance Metric Means and Standard Deviations (95th Percentile Latency)

	Mean							Standard Deviation						
Metric	AD	CVM	ES	KLD	KS	WD	WKS	AD	CVM	ES	KLD	KS	WD	WKS
EVS	0.62	0.61	0.85	0.71	0.72	0.49	0.67	0.24	0.24	0.13	0.22	0.19	0.29	0.22
ME	1.41	1.42	0.77	1.2	1.19	1.72	1.26	0.74	0.76	0.44	0.72	0.64	0.91	0.64
MAE	0.33	0.33	0.21	0.28	0.28	0.37	0.31	0.17	0.17	0.13	0.16	0.15	0.18	0.17
MSE	0.36	0.36	0.14	0.27	0.26	0.47	0.3	0.22	0.23	0.12	0.20	0.18	0.27	0.20
MedAE	0.15	0.15	0.12	0.14	0.13	0.16	0.15	0.18	0.18	0.15	0.16	0.16	0.18	0.18
MAPE	0.63	0.8	0.27	0.13	0.12	0.22	0.19	0.52	0.69	0.32	0.08	0.08	0.15	0.12
RSquared	0.62	0.61	0.85	0.71	0.72	0.49	0.67	0.24	0.24	0.13	0.22	0.19	0.29	0.22
Adj RSquared	0.31	0.30	0.73	0.47	0.49	0.08	0.41	0.43	0.45	0.24	0.41	0.34	0.52	0.39
RMSE Model	1.15	1.14	1.38	1.24	1.26	1.0	1.21	0.23	0.24	0.15	0.22	0.18	0.3	0.20

Table 6.13 Performance Metric Means and Standard Deviations (99th Percentile Latency)

	Mean							Standard Deviation						
Metric	AD	CVM	ES	KLD	KS	WD	WKS	AD	CVM	ES	KLD	KS	WD	WKS
EVS	0.61	0.6	0.9	0.72	0.71	0.48	0.63	0.25	0.25	0.11	0.21	0.20	0.29	0.24
ME	1.4	1.44	0.56	1.19	1.20	1.76	1.32	0.75	0.75	0.38	0.68	0.63	0.94	0.67
MAE	0.33	0.34	0.17	0.28	0.29	0.37	0.33	0.18	0.18	0.13	0.15	0.16	0.18	0.18
MSE	0.36	0.37	0.09	0.26	0.27	0.48	0.34	0.23	0.23	0.1	0.19	0.18	0.27	0.22
MedAE	0.16	0.16	0.11	0.13	0.14	0.16	0.16	0.19	0.18	0.13	0.15	0.17	0.18	0.19
MAPE	0.68	0.87	0.24	0.11	0.13	0.23	0.20	0.59	0.72	0.32	0.07	0.09	0.15	0.13
RSquared	0.61	0.6	0.9	0.72	0.71	0.48	0.63	0.25	0.25	0.11	0.21	0.20	0.29	0.24
Adj RSquared	0.30	0.27	0.83	0.48	0.48	0.06	0.33	0.45	0.45	0.19	0.38	0.35	0.52	0.43
RMSE Model	1.14	1.13	1.42	1.25	1.25	0.98	1.16	0.24	0.24	0.14	0.20	0.18	0.31	0.23

Table 6.14 Performance Metric Means and Standard Deviations (Max Latency)

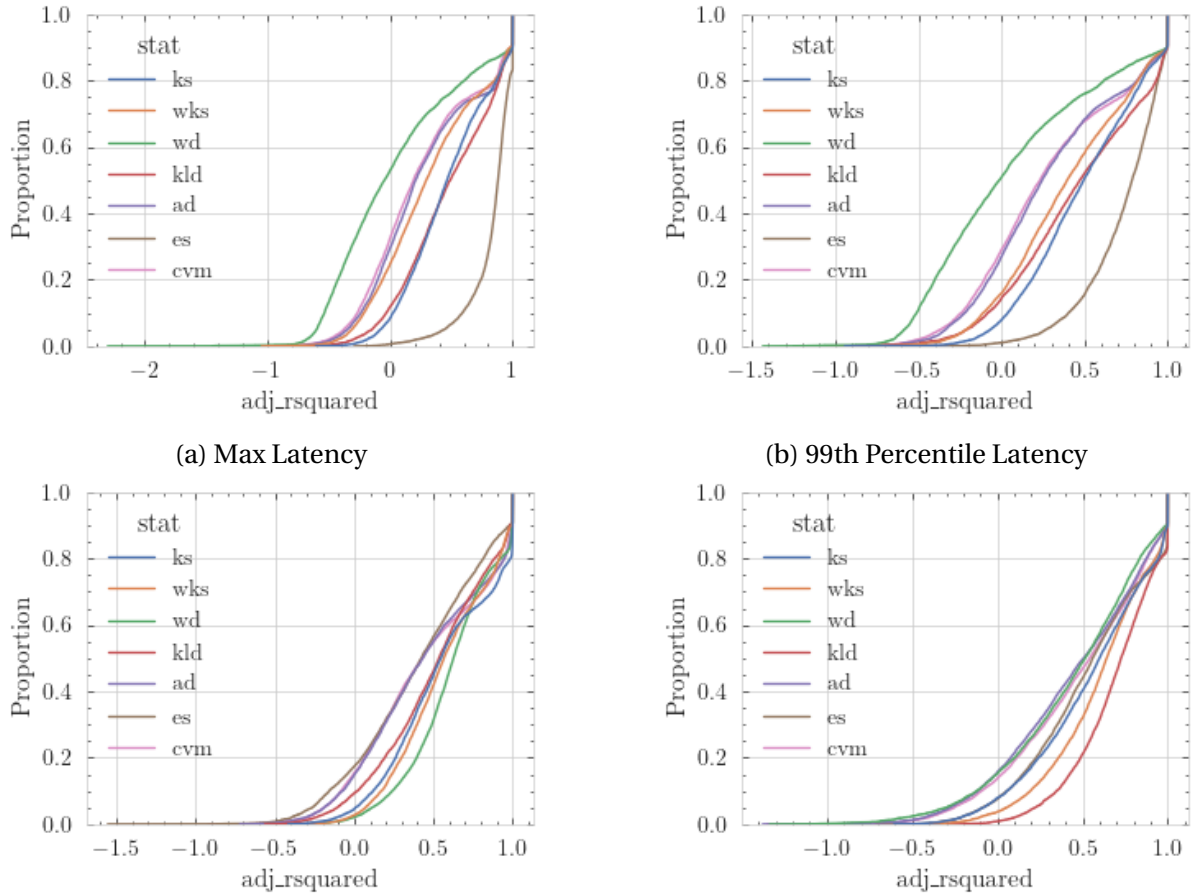


(a) 2^kF

(b) CCD

(c) BBD

Fig. 6.2 Distribution of Adj. RSquared Values per Statistic (by Experiment Design)



(a) Max Latency

(b) 99th Percentile Latency

(c) 95th Percentile Latency

(d) 50th Percentile Latency

Fig. 6.3 Distribution of Adj. RSquared Values per Statistic (by Percentile Latency)

The Worst-Performing: WD Statistic

We can see that the WD statistic has the worst average, overall value for seven out of the nine recorded performance metrics (i.e. EVS, ME, MAE, MSE, MedAE, RSquared and Adj. RSquared). The WD stat also displays the highest overall standard deviation for seven out of the nine recorded performance metrics, versus the other test statistics under consideration (i.e. EVS, ME, MAE, MSE, RSquared and Adj. RSquared, RMSE Model).

When grouped by experiment design, we see the WD statistic maintaining its inferior position with regards to average performance metric values across all three experiment designs. The results for the 2^kF (Table 6.9), and CCD approaches identify the WD statistic's mean values as underperforming the other test statistics across six of the nine metrics. The BBD approach reports underperformance by the WD statistic across seven of the nine metrics. The standard deviations recorded for the WD statistic's performance metrics show clear underperformance when using all three of the experiment design approaches, recording the highest standard deviations in values across six, seven and eight of the nine metrics, for the CCD, 2^kF and BBD approaches respectively.

The WD statistic's distribution of Adjusted RSquared values, grouped by latency percentile, appears to display the least favourable characteristics of all the test statistics (Figure 6.3). For the lower percentile latencies (i.e. 50th and 95th), all test statistics analysed appear to display closely similar distributions. albeit the WD statistic does record the best average performance metric values for the 95th percentile latency models, with the corresponding lowest levels of standard deviation. However, for the the higher percentiles (i.e. 99th and max), the WD statistic's ECDFs (shown in Figures 6.3a and 6.3b) appear significantly more concave to the origin, representing a more heavily dispersed distribution of values. This can be interpreted as an increased uncertainty regarding the expected value of the Adjusted RSquared metric for a model, measuring the more extreme percentile latency values, and based on the WD statistic, versus the other the statistics tested.

The distribution of Adjusted RSquared values for the WD statistic also center around the lowest mean value of all the test statistics for the the higher percentiles (shown in Figure 6.3a and Figure 6.3b). The fact the WD statistic displays these divergent characteristics most prominently for the higher percentile latencies is, again, of note. This implies that the confidence a system operator can attach to the recorded values of the WD statistic based model, deteriorates, just at the time it is needed most (i.e. when modelling extreme latency percentiles values and potentially significant streaming system degradation)

Figure 6.2 shows that when using the WD statistic as the response variable, the BBD approach appears to offer the worst results and similar characteristics to those mentioned above, namely a more heavily dispersed distribution and lower mean value of the recorded

Adjusted RSquared metric (versus the 2^kF and CCD approach). The 2^kF approach appears to offer the best results, should one use the WD statistic as the response variable. This would suggest system operators and practitioners might prefer to avoid building a model using a combination of the BBD approach and the WD statistic as the response variable. Especially so when attempting to model latency values or percentiles located further into the right-tail of that distribution.

Based on the results, the WD statistic appears to be the worst-performing test statistic for several reasons:

- **Performance metrics:** The WD statistic has the worst average value for seven out of the nine recorded performance metrics (EVS, ME, MAE, MSE, MedAE, R^2, and Adjusted R^2) overall, across the full range of experimental runs. It also displays the highest standard deviation for seven out of the nine performance metrics. This indicates that models using the WD statistic as the response variable generally have a poorer fit to the data and make less accurate predictions compared to models using other test statistics.

- **Consistency across experiment designs:** The WD statistic maintains its inferior performance across all three experiment designs (2^kF, CCD, and BBD). This suggests that the WD statistic's ineffectiveness is not limited to a particular experimental setup, making it a consistently poor choice for modelling streaming system performance degradation.

- **Performance across latency percentiles:** The WD statistic's distribution of Adjusted R^2 values, grouped by latency percentile, displays the least favourable characteristics, particularly for higher percentile latencies (99th and max). For these critical latency values, the WD statistic's ECDFs are more concave to the origin, indicating a more heavily dispersed distribution of values and increased uncertainty regarding the expected value of the Adjusted R^2 metric.

The WD statistic can be considered the least "informative" in this context because:

- It fails to capture relevant information about the differences between the distributions being compared as effectively as other test statistics. Although the WD statistic considers both the likelihoods and the distances between various outcome events, it may not be as sensitive to the specific characteristics of the distributions that are most relevant to modelling streaming system performance degradation.

- It is less sensitive to differences in the tails of the distributions, which is crucial when dealing with high percentile latency values that represent extreme and potentially critical performance degradation scenarios.

Models using the WD statistic as the response variable consistently underperform models using other test statistics in terms of performance metrics and stability, suggesting that the WD statistic is less effective at capturing the complex relationships between workload characteristics and system performance.

Noteworthy Findings

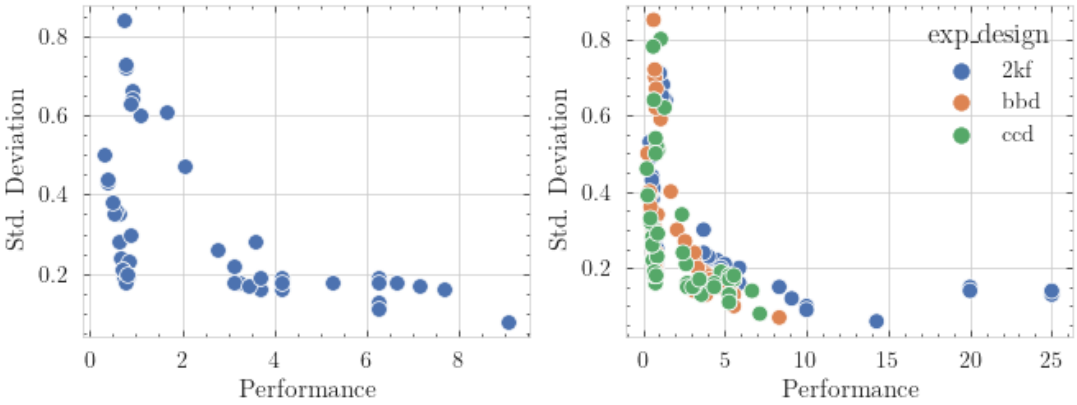
Overall the KS statistic displays performances metric standard deviation values that outperform other test statistics across seven of the nine metrics, and all nine metrics when calculated across all experimental runs overall, and when calculated using the 2^kF approach respectively (shown in Tables 6.7 and 6.8). The KS statistics's corresponding average performance metric values across the same groupings, however, do not represent similar outperformance versus other test statistics.

The KLD displays the lowest performance metric value standard deviations for the CCD approach, but no corresponding outperformance is seen regarding the mean performance metric values. It shows outperformance in both standard deviation and mean performance metric values when used to model the 50th percentile latency values. In general terms, this percentile latency is of least interest or significance to streaming system operators (versus the higher percentile latencies), as it does not represent an extreme value in relative terms, or a corresponding threat to maintaining system performance levels and SLAs.

It is of interest that Figure 6.4 indicates there to be an inverse relationship between a model's performance metrics and their standard deviations. We can see that as the average value of a model's performance metric rises, so does the standard deviation of those values fall. This is important as it implies system operators may be successful in identifying a combination of experiment design and response factor test statistic that optimises for both levels of model performance, as well as stability and robustness of those performance levels.

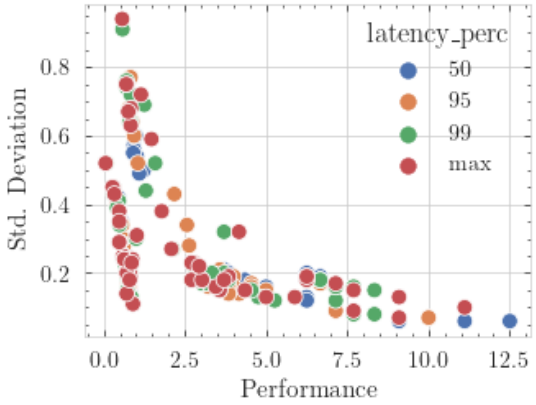
6.4.2 Model Stability Metrics

Figure 6.5 displays the mean values of the fitted linear regression models for each of the test statistics, recorded across all experiment runs and bootstrapped iterations. Table 6.15 displays both the mean values and the standard deviations for the same outputs as Table 6.15. Tables 6.16 and 6.17 display the standard deviations of the linear regression models'



(a) Across Experiments Overall

(b) By Experiment Design



(c) By Latency Percentile

Fig. 6.4 A comparison of model performance and stability in terms of performance metrics, illustrating the relationship between the average value of a model's performance metric and the standard deviation of those values across different experimental settings and test statistics.

coefficient values, for each of the test statistics, grouped by experiment design and latency percentile respectively. Figure 6.6 displays the distributions of model coefficient values, grouped by latency percentile.

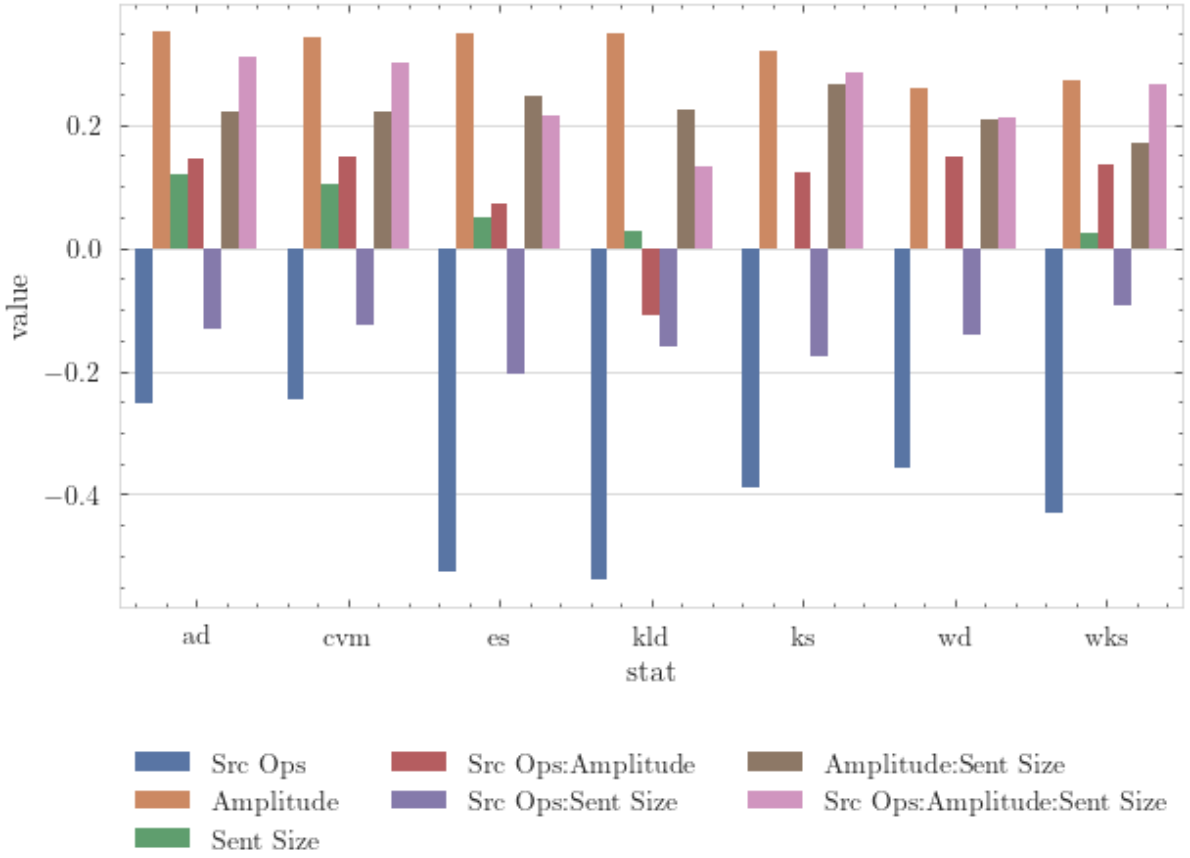


Fig. 6.5 Mean Model Coefficient Values per Statistic

The Most Stable: KLD Statistic

We can see from Figure 6.5 and Table 6.15 that, overall, across all experimental runs, all included test statistics appear to broadly agree regarding the direction and magnitude of the fitted regression models' coefficient values, for the majority of factors and interaction terms. The main exception to this, is the Source Operators and Amplitude factors' interaction term, when measured using the KLD statistic, which records of value of -0.11, while all other test statistics record values of similar magnitude to the KLD statistic, but with a positive, rather than negative sign.

When considering results across all experimental runs, we can see from Table 6.15 that the KLD statistic displays lower levels of standard deviation across all model factor and

interaction term coefficient values. This represents outperformance with respect to the stability of those values, and therefore an increased level of confidence one is able to assign to the characteristics of such a fitted model.

When the results of the experimental runs are grouped by experiment design Table 6.16 shows that the KLD statistic displays the lowest standard deviation of model factor coefficient values across five of the seven factors when using a 2^kF approach, and all seven factors when using a CCD approach. This can be interpreted as the KLD statistic outperforming other statistics when selected as the response variable, generating less uncertainty as to whether the fitted model coefficient values accurately reflect the true underlying system interactions and relationships. When using a BBD approach the KLD statistic is outperformed in the above regard by the ES statistic, which displays the lowest standard deviation for all relevant factors (the three-way interaction terms is not captured by the BBD approach, leaving six relevant factor coefficients).

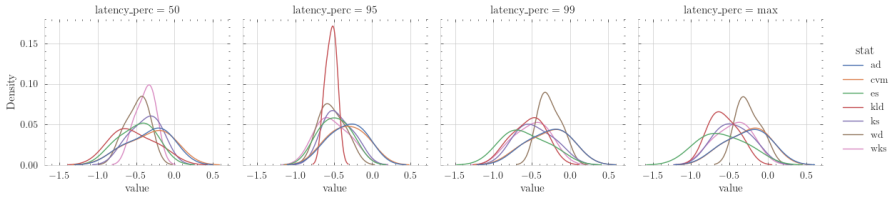
	Mean							Standard Deviation						
Factor	AD	CVM	ES	KLD	KS	WD	WKS	AD	CVM	ES	KLD	KS	WD	WKS
SrcOps	-0.25	-0.25	-0.52	-0.54	-0.39	-0.36	-0.43	0.64	0.64	0.5	0.49	0.58	0.64	0.55
Amplitude	0.35	0.34	0.35	0.35	0.32	0.26	0.27	0.63	0.63	0.55	0.53	0.58	0.63	0.57
SentSize	0.12	0.1	0.05	0.03	0	0	0.02	0.68	0.68	0.59	0.54	0.66	0.71	0.64
SrcOps:Amplitude	0.15	0.15	0.07	-0.11	0.12	0.15	0.13	0.88	0.87	0.8	0.70	0.81	0.79	0.75
SrcOps:Sent Size	-0.13	-0.12	-0.21	-0.16	-0.18	-0.14	-0.09	0.84	0.82	0.75	0.74	0.83	0.84	0.77
Amplitude:SentSize	0.22	0.22	0.25	0.22	0.27	0.21	0.17	0.83	0.83	0.76	0.68	0.8	0.83	0.78
SrcOps:Amplitude:SentSize	0.31	0.3	0.21	0.13	0.29	0.21	0.27	0.69	0.69	0.65	0.52	0.67	0.62	0.61

Table 6.15 Model Coefficient Mean Values and Standard Deviations

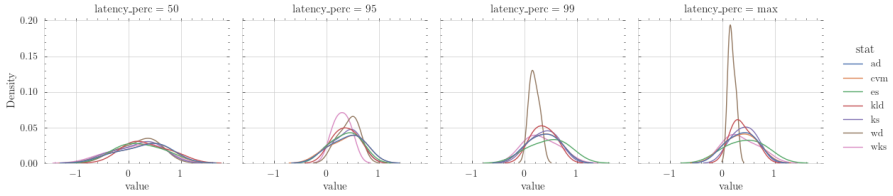
When the results of the experimental runs are grouped by latency percentile, Table 6.17 shows the KLD statistic, continuing to dominate and generally outperform other test statistics with regard to the standard deviations of the recorded model factor coefficient values. Across the various percentile latency thresholds, the KLD statistic records the lowest standard deviations across five of the seven factors for the 50th percentile across six for the 95th percentile, across four for the 99th percentile, and across six of the seven for the max latency. This implies high levels of model robustness across the full range of system latency environments, when the KLD statistic is used as the experiment response variable.

Based on the results, the KLD statistic appears to be the most stable test statistic for several reasons:

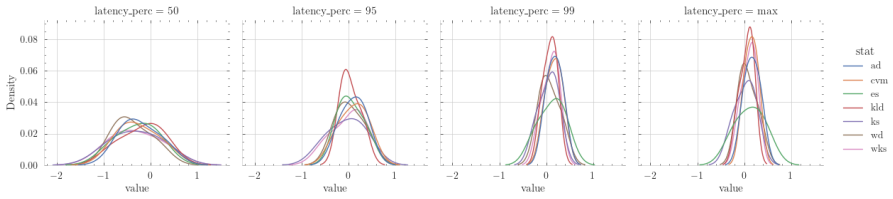
- Coefficient value standard deviations: Across all experimental runs, the KLD statistic displays lower levels of standard deviation for all model factor and interaction term coefficient values compared to other test statistics. This suggests that models using the



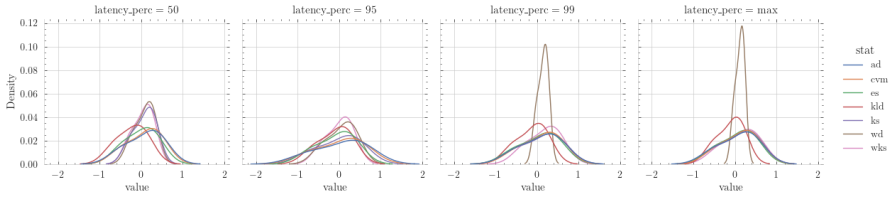
(a) Source Operators



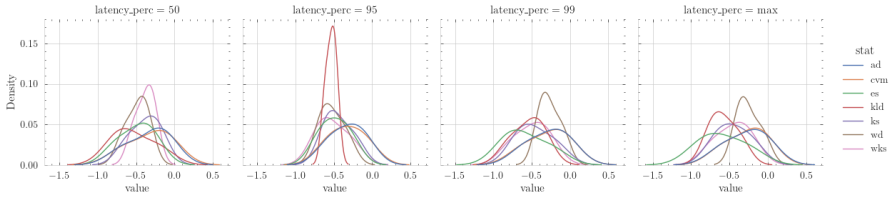
(b) Amplitude



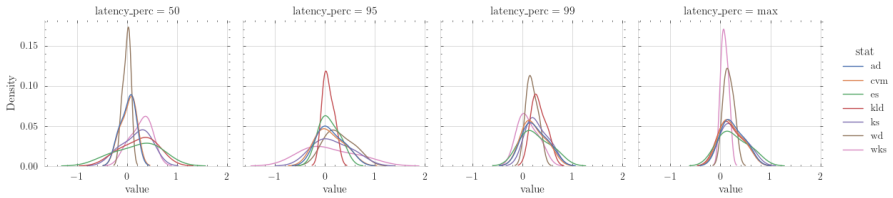
(c) Sentence Size



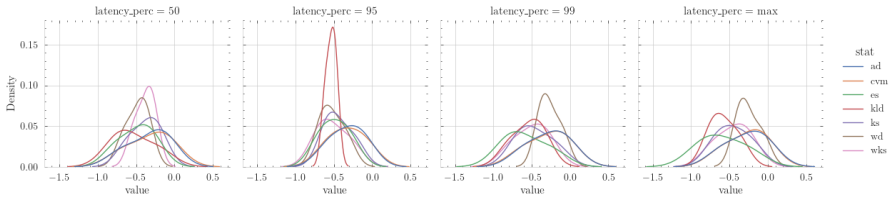
(d) Source Operators:Amplitude



(e) Source Operators:Sentence Size



(f) Amplitude:Sentence Size



(g) Source Operators:Amplitude:Sentence Size

Fig. 6.6 Distribution of Coefficient Values per Statistic (by Percentile Latency)

Exp. Design	Stat	SrcOps	Amplitude	SentSize	SrcOps:Amplitude	SrcOps:SentSize	Amplitude:SentSize	SrcOps:Amplitude:SentSize
2kF	AD	0.62	0.63	0.64	0.64	0.62	0.6	0.64
	CVM	0.63	0.63	0.64	0.64	0.64	0.61	0.63
	ES	0.46	0.48	0.51	0.51	0.46	0.47	0.54
	KLD	0.49	0.47	0.44	0.45	0.47	0.45	0.48
	KS	0.57	0.58	0.57	0.56	0.57	0.55	0.59
	WD	0.65	0.63	0.64	0.62	0.62	0.62	0.62
	WKS	0.50	0.54	0.50	0.52	0.51	0.52	0.53
CCD	AD	0.52	0.53	0.56	0.91	0.96	1.01	0.94
	CVM	0.51	0.53	0.55	0.89	0.95	1.00	0.94
	ES	0.47	0.52	0.5	0.92	0.91	1.00	0.95
	KLD	0.41	0.43	0.43	0.71	0.74	0.78	0.75
	KS	0.51	0.47	0.52	0.88	0.93	0.98	0.92
	WD	0.47	0.45	0.56	0.81	0.87	0.89	0.83
	WKS	0.45	0.47	0.52	0.82	0.83	0.93	0.85
BBD	AD	0.61	0.64	0.71	0.73	0.79	0.82	0.00
	CVM	0.62	0.64	0.72	0.73	0.75	0.81	0.00
	ES	0.46	0.49	0.61	0.58	0.64	0.71	0.00
	KLD	0.55	0.57	0.67	0.69	0.81	0.75	0.00
	KS	0.57	0.61	0.72	0.68	0.71	0.8	0.00
	WD	0.73	0.76	0.8	0.84	0.89	0.94	0.00
	WKS	0.61	0.63	0.77	0.68	0.76	0.82	0.00

Table 6.16 Model Coefficient Values' Standard Deviations (by Experiment Design)

KLD statistic as the response variable produce more consistent and reliable estimates of the relationships between the input factors and the system performance.

- Consistency across experiment designs: When the results are grouped by experiment design, the KLD statistic displays the lowest standard deviation of model factor coefficient values for most factors in the 2^kF and CCD. Although it is outperformed by the ES statistic in the BBD, the KLD statistic still maintains a high level of stability across all designs.

- Consistency across latency percentiles: When the results are grouped by latency percentile, the KLD statistic continues to outperform other test statistics in terms of the standard deviations of the model factor coefficient values. This indicates that the KLD statistic provides a stable and reliable measure of system performance degradation across the full range of latency percentiles.

The stability of the KLD statistic implies several important things:

- Increased confidence in model interpretation: The lower variability in the coefficient values suggests that the relationships between the input factors and the system performance, as captured by the KLD statistic, are more consistent and reliable. This increases the confidence in the interpretation of the model results and the conclusions drawn from them.

- Robustness to uncertainties: The stability of the KLD statistic indicates that models using it as the response variable are less sensitive to uncertainties or variability in the experimental data. This robustness is crucial when dealing with complex systems like streaming architectures, where many factors can introduce variability in the performance measurements.
- Reproducibility and generalizability: The consistency of the KLD statistic across different experimental designs and latency percentiles suggests that the insights gained from models using this statistic are more likely to be reproducible and generalisable to other streaming system scenarios. This is important for developing best practices and guidelines for system design and optimisation.

In summary, the KLD statistic's stability, as evidenced by the lower variability in model coefficient values across different experimental conditions, implies increased confidence in model interpretation, robustness to uncertainties, and better reproducibility and generalisability of the insights gained from the RSM analysis.

The Least Stable: WD Statistic

When considering results across all experimental runs table Table 6.15 identifies three test statistics that display relatively higher standard deviations of model coefficient values, across four or more of the seven factors in question. The CVM statistic records the highest standard deviations across four, the WD statistic across five, and the AD statistic across six of the seven coefficient values. When grouped by experiment design, the same three statistics remain the poorest performers. When combined with the 2^kF approach each of the three statistics in question record the highest standard deviations across four of the seven factor coefficient values. The AD statistic is the clear underperformed when the CCD approach is implemented, while the BBD approach results in the WD statistic underperforming. Regarding the experiment output, when grouped by latency percentile, the most noticeable observation concerns the degradation in the stability of the coefficient values when using the WD statistic as the response variable, as we move up through the latency percentiles. This is concerning as it implies that the confidence a system operator can attach to the recorded values of the WD statistic based model, deteriorates, just at the time it is needed most (i.e. when modelling extreme latency percentiles values and potentially significant streaming system degradation)

Based on the results presented in Section 6.4.2, the WD statistic appears to be the least stable test statistic for several reasons:

Latency Perc.	Stat	SrcOps	Amplitude	SentSize	SrcOps:Amplitude	SrcOps:SentSize	Amplitude:SentSize	SrcOps:Amplitude:SentSize
50	AD	0.56	0.63	0.62	0.84	0.73	0.81	0.69
	CVM	0.57	0.64	0.60	0.80	0.69	0.81	0.68
	ES	0.47	0.56	0.54	0.74	0.62	0.77	0.58
	KLD	0.46	0.59	0.53	0.74	0.62	0.69	0.55
	KS	0.55	0.58	0.61	0.74	0.71	0.77	0.67
	WD	0.48	0.57	0.54	0.71	0.65	0.71	0.62
	WKS	0.48	0.58	0.60	0.73	0.68	0.76	0.65
95	AD	0.57	0.57	0.61	0.9	0.85	0.81	0.69
	CVM	0.57	0.57	0.63	0.9	0.83	0.82	0.72
	ES	0.52	0.55	0.60	0.84	0.78	0.77	0.60
	KLD	0.42	0.47	0.48	0.64	0.69	0.64	0.46
	KS	0.52	0.51	0.62	0.82	0.89	0.83	0.67
	WD	0.54	0.55	0.64	0.75	0.83	0.79	0.58
	WKS	0.49	0.45	0.54	0.66	0.77	0.77	0.57
99	AD	0.69	0.64	0.69	0.9	0.86	0.84	0.7
	CVM	0.71	0.65	0.69	0.89	0.87	0.83	0.68
	ES	0.5	0.54	0.55	0.81	0.78	0.73	0.68
	KLD	0.56	0.53	0.57	0.72	0.79	0.68	0.54
	KS	0.62	0.59	0.65	0.84	0.83	0.78	0.66
	WD	0.71	0.67	0.75	0.83	0.91	0.86	0.63
	WKS	0.57	0.60	0.65	0.77	0.77	0.78	0.60
Max	AD	0.71	0.66	0.70	0.9	0.88	0.83	0.69
	CVM	0.71	0.64	0.69	0.89	0.85	0.83	0.67
	ES	0.5	0.51	0.55	0.78	0.77	0.74	0.69
	KLD	0.5	0.5	0.53	0.67	0.8	0.67	0.51
	KS	0.64	0.61	0.68	0.84	0.84	0.82	0.64
	WD	0.74	0.7	0.76	0.84	0.93	0.91	0.65
	WKS	0.63	0.62	0.68	0.81	0.81	0.78	0.59

Table 6.17 Model Coefficient Values' Standard Deviations (by Latency Percentile)

- Coefficient value standard deviations: Across all experimental runs, the WD statistic, along with the AD and CVM statistics, displays relatively higher standard deviations of model coefficient values for most factors compared to other test statistics. This suggests that models using the WD statistic as the response variable produce less consistent and reliable estimates of the relationships between the input factors and the system performance.

- Inconsistency across experiment designs: When the results are grouped by experiment design, the WD statistic is among the poorest performers in terms of the standard deviations of model coefficient values. It underperforms in the 2^kF and CCD and is the clear underperformer in the BBD.

- Degradation of stability for higher latency percentiles: When the results are grouped by latency percentile, the stability of the coefficient values for the WD statistic-based models deteriorates as the latency percentile increases. This is particularly concerning because the higher percentile latencies (e.g., 99th and max) are often the most critical for assessing system performance and identifying potential issues.

The lack of stability in the WD statistic implies several important things:

- Reduced confidence in model interpretation: The higher variability in the coefficient values suggests that the relationships between the input factors and the system performance, as captured by the WD statistic, are less consistent and reliable. This reduces the confidence in the interpretation of the model results and the conclusions drawn from them.

- Sensitivity to uncertainties: The lack of stability in the WD statistic indicates that models using it as the response variable are more sensitive to uncertainties or variability in the experimental data. This sensitivity can lead to less robust and less reliable insights, especially when dealing with complex streaming systems where many factors can introduce variability in the performance measurements.

- Limited reproducibility and generalizability: The inconsistency of the WD statistic across different experimental designs and latency percentiles suggests that the insights gained from models using this statistic may be less reproducible and generalisable to other streaming system scenarios. This can limit the usefulness of the Response Surface Methodology analysis for developing best practices and guidelines for system design and optimisation.

- Potential misinterpretation of system performance: The degradation of stability for higher latency percentiles is particularly concerning because it implies that the WD statistic may not provide a reliable measure of system performance when it matters most, i.e., when the system is experiencing critical performance issues.

In summary, the WD statistic's lack of stability, as evidenced by the higher variability in model coefficient values across different experimental conditions, implies reduced confidence in model interpretation, sensitivity to uncertainties, limited reproducibility and generalisability, and potential misinterpretation of system performance.

6.5 Conclusion

In this chapter, we applied RSM to systematically study the performance degradation of streaming systems under various workloads and quantify the effects of different workload model parameters on system latency. We focused on both model performance and model stability, comparing and contrasting the results for each generated model under varying RSM experiment designs, workload characteristics, generative functions, latency percentile measurements, and goodness-of-fit tests.

The key findings and best practices derived from this chapter are as follows:

1. Choice of goodness-of-fit test and experiment design matters:
    - The selection of the goodness-of-fit test used as the response variable and the RSM experiment design approach significantly impacts the performance and stability of the generated models.
    - The ES statistic generally outperformed other test statistics across most experimental combinations, particularly when modelling higher percentile latency thresholds.
    - The BBD approach often provided the best results when used in conjunction with the ES statistic.
2. Consider the latency percentile of interest:
    - The impact of the chosen goodness-of-fit test and experiment design is magnified when considering the specific latency percentile threshold of interest.
    - Higher percentile latencies (e.g. 99th percentile and max latency) are more sensitive to the choice of test statistic and experiment design compared to lower percentiles.

- Practitioners should carefully select the most appropriate combination of test statistic and experiment design based on the latency percentile they are most concerned with optimising.

3. Avoid suboptimal combinations:
   - Certain combinations of test statistics and experiment designs consistently underperformed, such as the WD statistic across most experimental combinations.
   - Practitioners should be aware of these suboptimal combinations and avoid them when modelling streaming system performance degradation.
4. Validate and iterate:
   - While the findings provide valuable insights, practitioners should validate the best practices and recommendations within their specific streaming system environments.
   - Iterative experimentation and fine-tuning may be necessary to identify the most suitable combination of test statistic, experiment design, and workload characteristics for a given system.

In terms of tuning an autoscaler like DS2 in practice, the following recommendations can be made:

1. Start with the ES statistic and BBD approach:
   - When modelling streaming system performance degradation to inform autoscaler tuning, consider using the ES statistic as the response variable and the BBD experiment design as a starting point.
2. Focus on the relevant latency percentile:
   - Identify the latency percentile that is most critical for your specific use case and SLAs, and prioritise optimising the autoscaler for that percentile.
3. Iteratively refine the model:
   - Use the insights gained from the initial RSM experiments to refine the autoscaler configuration and workload characteristics.
   - Conduct additional experiments to validate the improvements and further optimise the autoscaler settings.

4. Monitor and adapt:

    - Continuously monitor the streaming system's performance and robustness under the tuned autoscaler settings.
    - Be prepared to adapt the autoscaler configuration as workload characteristics, system requirements, or business objectives change over time.

By applying the best practices and recommendations derived from this chapter, practitioners can more effectively model and optimise the performance and robustness of their streaming systems, ultimately leading to better-tuned autoscalers and more resilient systems in the face of varying workload conditions.

Chapter 7

Conclusions

This chapter presents a summary of the thesis and the research work contained within, investigating the performance, robustness and behaviours of DSPS autoscalers. We identify and discuss the contributions and limitations of these works, provide a recap of works undertaken, and outline suggested, open research problems in the field, laying the foundations for a number of ongoing research efforts.

7.1 Thesis Summary

In this thesis we have explored the performance, robustness and behavioural phenomena experienced by state-of-the-art DSPSs and autoscalers. We adopt a number of approaches to empirically study, quantify and analyse these characteristics and behaviours. In Chapter 2 we present a comprehensive literature review, covering stream processing, streaming system autoscalers, workload generation and modelling, and streaming system benchmarking and evaluation. Chapter 3 outlines the overall research methodology applied throughout the course of the PhD and provides details of our assumptions and information regarding any pre-existing systems used as our default choice for experiments.

In Chapter 4, we turn our attention towards the impact of parameter tuning for a state-of-the-art autoscaler, empirically studying its impacts, both in terms of Stability, Accuracy, Short settling time, and no Overshoot (SASO) properties as well as behavioural phenomena outside the scope of SASO, also contributing a categorisation of autoscaler mechanisms. We demonstrate the potential for the application of moving average models to produce more robust autoscaler behaviour and to significantly ameliorate a number of undesirable behavioural phenomena (e.g. extreme parallelism shifts and lack of overall autoscaler behavioural robustness). We also establish applicable methods to allow the systematic

evaluation of these models. We show applying Moving Average (MA) models to autoscaler output can:

1. Successfully mitigate instances of undesirable, extreme parallelism shifts, with different models displaying differing characteristics and levels of effectiveness. We have demonstrated the potential for MA models to mitigate over 90% of extreme parallelism shifts.

2. Reduce the number of rescaling decisions and their latency and throughput impacts, through:

   (a) reduced oscillations (our Bias measure shows our approaches realise these benefits without causing under-/over-provisioning),

   (b) mitigating costly-to-enact extreme parallelism shifts.

3. Significantly reduce autoscaler uncertainty and volatility when faced with dynamic and variable incoming workloads and, in particular, we confirm our expectations that smoother models are most successful in mitigating volatility.

The work presented in Chapter 4 was published at DEBS 2023 [98].

Chapter 5 examined the question of how to measure and quantify the robustness of DSPSs in the face of perturbations in the operating environment. We present a range of non-parametric goodness-of-fit tests which can act as quantifiers of such system robustness. Through their application to the measurement and quantification of streaming system robustness, analysis and comparison, we show that different tests produce differing relative measures of system robustness and display differing relationships, interactions and levels of correlation between each other. These differing measures are affected not only by the test statistics' inherent characteristics, but also by the particular performance metric (i.e. latency percentile) under scrutiny. In general, the higher the occurrence of observed latency values within the extreme tails of the distributions, or the higher the latency percentile under scrutiny, the larger the divergence between test statistics' results interpretation and the lower the levels of inter-statistic correlation. This chapter suggests that when seeking to quantify the robustness of distributed systems, practitioners should look to build multiple non-parametric goodness-of-fit measures into their analysis, rather than rely on a single metric. The work presented in Chapter 5 was published at EPEW2022 [97].

Chapter 6 builds upon and develops out the foundations laid down in Chapter 5. We apply a selection of non-parametric goodness-of-fit tests (first presented in Chapter 5, with two new additions) and draw on RSM to help systematically study the performance degradation of streaming systems under various workloads. We quantify the effects of different

workload model parameters on system latency. We focus on both model performance and model stability, and compare and contrast the results for each of our generated models, under varying RSM experiment designs, workload characteristics and generative functions, latency percentiles measurements, and goodness-of-fit tests.

We have demonstrated that the choice of which test statistic to use as the response variable, along with which RSM experiment design approach, has a significant impact on outputs, including the performance and stability of the generated model. This impact is further magnified when the particularly percentile latency threshold of interest is taken into account.

There are combinations of approaches which clearly perform well, compared to other possible combinations (i.e. of experiment design and response variable test statistic). For example, the ES statistic tends to dominate other statistic choices across most experimental combinations, especially when concerned with modelling the higher percentile latency thresholds. However, the WD appears to be a suboptimal choice across most combinations and across the experimental runs overall.

7.2 Future Research Directions

Here we motivate a number of areas of future research, arising from lessons learnt throughout the PhD.

7.2.1 Generalisability

An area of key interest is the generalisability of our developed approaches to other operating environments. We identify three major threats to the level of generalisability, as follows:

1. Single streaming platform: The experimental results in this thesis consider a single streaming platform, Apache Flink.

2. Single autoscaler: The experimental results in this thesis consider a single state-of-the-art autoscaler, DS2.

3. Single workload: The experimental results used a single Wordcount workload, leveraging four arrival processes.

Future research should involve the expansion of the experimental environment to include multiple, varied additions to the above three listed selections.

7.2.2 Windowing and Weighting

Our work in Chapter 4 highlights several interesting avenues of future research. Firstly, we see the potential for an ensemble approach to provide the distinct benefits of several MA models. Secondly, we see potential to incorporate these measures to inform other operating parameters for autoscalers. For example, to increase contingency multipliers for scaling decisions in periods of uncertainty, or introduce MA models dynamically when dynamic and variable incoming workloads are detected. Finally, we may incorporate estimates on rescaling duration (e.g. accounting for state size) as a mechanism to optimise the choice of window size.

7.2.3 Robustness: Measurement and Quantification

Our work in Chapters 5 and 6 suggests that when seeking to quantify the robustness of distributed systems, or model system performance degradation, practitioners should look to build multiple non-parametric goodness-of-fit and distance measures into their analysis, rather than rely on a single metric. We have shown that different goodness-of-fit tests produce differing relative measures of implied system robustness in the face of various disturbances to the incoming workload characteristics and workload function input variables. The selected test statistics display differing relationships, interactions and levels of correlation between each other. Our results imply these are affected by not only the test statistics' inherent characteristics, sensitivities and calculation methods, but also by the particular percentile of observed latency values under scrutiny.

[70] show that even after a metric is chosen, it can still be useful to familiarise oneself with other metrics, especially if one also considers the relationships among them and that analysis of a problem using several different metrics can provide complementary insights. For these reasons we propose future research avenues include the research and analysis of further, as yet untested goodness-of-fit tests and distance metrics, approaches and concepts.

As a starting point, we propose the following:

- Discrepancy metric [70]
- Hellinger distance [70]
- Levy metric [70]
- Prokhorov metric [70]
- Separation distance [70]

- Total variation distance [70]
- χ^2 distance [70]

7.2.4 Tooling Support for Benchmarking Practitioners

This thesis has highlighted several implications for performance engineering and benchmarking practitioners. We have made several recommendations which have the potential to improve the rigour with which we benchmark systems. Throughout this work we have demonstrated a strong commitment to open practices, for example through the Artifact Evaluation process with our DEBS submission, and by making available data alongside this thesis. We see further opportunities to embed our methods within standard tooling and benchmarking (e.g. the Nexmark benchmarks). We recognise previous efforts in the literature which explore existing statistical practices in systems research [91, 90, 44] and see potential to extend these to capture the current state of practice in robustness measures. We also recognise the need for training of practitioners to have the confidence in applying these methods, noting prior efforts including [117], and ensuring that these principles flow through into artifact evaluation guidance for the systems community (e.g. [14]).

Glossary

Train-Test Split A technique in machine learning where the dataset is divided into a training set for model learning and a test set for model evaluation. This method helps prevent over-fitting and ensures the model's ability to generalize to unseen data.

K-Fold Cross Validation A resampling procedure in machine learning used to evaluate a model's performance on a limited data sample. The method divides the dataset into k subsets, or folds, then iteratively trains the model on $k-1$ folds while using the remaining fold as the test set.

Stream Processing Engine The specific core component within a stream processing system that deals with the real-time processing of data streams. It consists of the computational algorithms and methods that operate on the incoming data streams. This includes tasks like filtering, aggregating, transforming, and analyzing the streaming data.

Event Processing Also known as Complex Event Processing (CEP), refers to the identification and analysis of meaningful events, patterns, or relationships within data streams. CEP systems can detect and respond to specific conditions, triggers, or patterns in real-time, facilitating rapid decision-making and action.

Streaming Analytics A subfield of stream processing that focuses on the real-time analysis and processing of data streams in order to extract actionable insights. This term is often used interchangeably with real-time analytics.

Data Stream A continuous, unbounded sequence of data elements generated at a high rate from sources such as sensors, social media platforms, web applications, financial transactions, or IoT devices. Data streams are characterized by their dynamic nature and potential for rapid change.

Source The component responsible for ingesting data streams from external data producers. Sources can be connected to various data providers, such as message brokers, databases, or APIs, and can handle data serialisation and deserialisation, as well as schema management.

Bootstrapping Random Sampling with Replacement A resampling technique used to estimate statistics on a population by sampling a dataset with replacement.

Sink The component in a SPSs that outputs the results of data processing to external systems, such as databases, message queues, or storage systems. Sinks are responsible for handling data serialisation and deserialisation, as well as managing connections and data consistency with the target system.

Operator the processing elements in a SPS that perform transformations, computations, or aggregations on the data streams. Operators can be stateless, meaning they do not maintain any internal state between processing events, or stateful, meaning they maintain internal state to perform computations across multiple events.

Parallelism Refers to the concurrent execution of multiple instances of processing operators, enabling the system to process data streams more efficiently and achieve higher throughput. Parallelism can be achieved at the level of individual operators or across entire processing pipelines.

Resource Allocation the process of assigning computational resources, such as CPU, memory, and network capacity, to the various components of a stream processing system. Effective resource allocation is crucial for maintaining system performance, scalability, and cost-efficiency and can be guided by workload models and autoscaling algorithms.

Autoscaling the dynamic adjustment of resource allocations and processing parallelism in response to changes in workload patterns or system performance. Autoscalers use workload models, system monitoring data, and scaling policies to make resource allocation decisions, helping to ensure that SPSs maintain performance and resource utilization targets.

DSMS a specialized system designed to handle the unique challenges associated with processing data streams. It typically includes features such as continuous queries, sliding window operators, and mechanisms for handling out-of-order data.

Continuous Query an ongoing query that processes data streams in real-time, as opposed to traditional queries which operate on static data sets. Continuous queries enable the continuous extraction of insights from streaming data, allowing for real-time or near-real-time decision-making.

Windowing a technique used in stream processing to divide data streams into finite, manageable subsets called windows. Windows can be defined based on time intervals, the number of events, or custom criteria, and are used to perform aggregate computations or pattern detection on the Data Stream.

Publish/Subscribe Systems (Pub/Sub) messaging frameworks that enable asynchronous, real-time communication between distributed systems. In this paradigm, data producers (publishers) disseminate data to data consumers (subscribers) who have expressed interest in specific data types or topics. Pub/Sub systems streamline data dissemination in real-time processing environments.

Acronyms

2^k**F** 2^k Factorial Design.
API Application Programming Interface.
BBD Box-Behnken Design.
CCD Central Composite Design.
CES Complex Event System.
DSPE Distributed Stream Processing Engine.
DSPS Distributed Stream Processing System.
RDD Resilient Distributed Dataset.
SLA Service Level Agreement.
SPE Stream Processing Engine.
SPS Stream Processing System.
VM Virtual Machine.
AD Anderson-Darling.
ANOVA Analysis of Variance.
AT Activation Time.
BaaS Benchmarking as a Service.
CDF Cumulative Distribution Function.
CEP Complex Event Processing.
CLT Central Limit Theorem.
CPU Central Processing Unit.
CVM Cramér-von Mises.
DAG Directed Acyclic Graph.
DAS Data Accelerator for Streaming.
DSMS Data Stream Management System.
DSRM Design Science Research Methodology.
ECDF Empirical Cumulative Distribution Function.
ECF Empirical Characteristic Function.
EDF Empirical Distribution Function.

ES Epps-Singleton.
ETL Extract, Transform, Load.
EVT Extreme Value Theory.
I/O Input/Output.
IoT Internet of Things.
KLD Kullback-Leibler Divergence.
KS Kolmogorov-Smirnov.
LAN Local Area Network.
MA Moving Average.
MAPE Monitor, Analyse, Plan, Execute.
OFAT One-Factor-At-a-Time.
PIR Policy Interval Rate.
QoS Quality of Service.
RL Reinforcement Learning.
RO Research Objective.
RQ Research Question.
RSM Response Surface Methodology.
SASO Stability, Accuracy, Short settling time, and no Overshoot.
WD Wasserstein Distance.
WKS Weighted Kolmogorov-Smirnov.
WUT Warm-Up Time.
WWW World Wide Web.
RMSE Root Mean Squared Error of the Model.

References

[1] URL: https://databrickslabs.github.io/dbldatagen/public_docs/index.html.

[2] Tarek Abdelzaher, Yixin Diao, Joseph L Hellerstein, Chenyang Lu, and Xiaoyun Zhu. "Introduction to control theory and its application to computing systems". In: *Performance Modeling and Engineering*. Springer, 2008, pp. 185–215.

[3] Kiryong Ha Agape and Daniel Boeve Anca. *Throughput autoscaling: Dynamic sizing for Facebook.com*. Engineering at Meta. Sept. 14, 2020. URL: https://engineering.fb.com/2020/09/14/networking-traffic/throughput-autoscaling/ (visited on 02/26/2023).

[4] Tyler Akidau, Robert Bradshaw, Craig Chambers, Slava Chernyak, Rafael J. Fernández-Moctezuma, Reuven Lax, Sam McVeety, Daniel Mills, Frances Perry, Eric Schmidt, and Sam Whittle. "The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing". en. In: *Proceedings of the VLDB Endowment* 8.12 (Aug. 2015), pp. 1792–1803. ISSN: 2150-8097. DOI: 10.14778/2824032.2824076. URL: https://dl.acm.org/doi/10.14778/2824032.2824076 (visited on 10/25/2021).

[5] Ahmed Ali-Eldin, Johan Tordsson, and Erik Elmroth. "An adaptive hybrid elasticity controller for cloud infrastructures". In: Apr. 2012, pp. 204–212. DOI: 10.1109/NOMS.2012.6211900.

[6] Lamees Alqassem, Thanos Stouraitis, Ernesto Damiani, and Ibrahim (Abe) Elfadel. "Proactive Random-Forest Autoscaler for Microservice Resource Allocation". In: *IEEE Access* PP (Jan. 2023), pp. 1–1. DOI: 10.1109/ACCESS.2023.3234021.

[7] George Anderson, Tshilidzi Marwala, and Fulufhelo Vincent Nelwamondo. "A response surface methodology approach to operating system scheduler tuning". In: *2010 IEEE International Conference on Systems, Man and Cybernetics*. IEEE. 2010, pp. 2684–2689.

[8] T. W. Anderson and D. A. Darling. "Asymptotic Theory of Certain "Goodness of Fit" Criteria Based on Stochastic Processes". en. In: *The Annals of Mathematical Statistics* 23.2 (June 1952), pp. 193–212. ISSN: 0003-4851. DOI: 10.1214/aoms/1177729437. (Visited on 03/26/2022).

[9] Leonardo Aniello, Silvia Bonomi, Federico Lombardi, and Alessandro Zelli. "An Architecture for Automatic Scaling of Replicated Services". In: Aug. 2014, pp. 122–137. ISBN: 978-3-319-09580-6. DOI: 10.1007/978-3-319-09581-3_9.

[10] *App Scaling - AWS Application Auto Scaling - AWS*. 2023. URL: https://aws.amazon.com/autoscaling/ (visited on 07/01/2023).

[11] Hamid Arabnejad, Claus Pahl, Pooyan Jamshidi, and Giovani Estrada. "A Comparison of Reinforcement Learning Techniques for Fuzzy Cloud Auto-Scaling". en. In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. Madrid, Spain: IEEE, May 2017, pp. 64–73. ISBN: 978-1-5090-6611-7. DOI: 10.1109/CCGRID.2017.15. URL: http://ieeexplore.ieee.org/document/7973689/ (visited on 10/24/2021).

[12] Arvind Arasu, Mitch Cherniack, Eduardo Galvez, David Maier, Anurag S. Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. "Linear Road: A Stream Data Management Benchmark". In: *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*. VLDB '04. Toronto, Canada: VLDB Endowment, 2004, pp. 480–491. ISBN: 0120884690.

[13] Arnaud Legoux and Dimitrios Kouzis Loukas. *ALMA-Arnaud-Legoux-Moving-Average*. 2009.

[14] *Artifact Review and Badging - Current*. URL: https://www.acm.org/publications/policies/artifact-review-and-badging-current.

[15] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. "Workload analysis of a large-scale key-value store". en. In: (), p. 12.

[16] Luciano Baresi, Sam Guinea, Alberto Leva, and Giovanni Quattrocchi. "A discrete-time feedback controller for containerized cloud applications". en. In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. Seattle WA USA: ACM, Nov. 2016, pp. 217–228. ISBN: 978-1-4503-4218-6. DOI: 10.1145/2950290.2950328. URL: https://dl.acm.org/doi/10.1145/2950290.2950328 (visited on 10/24/2021).

[17] Ermanno Battista, S. Martino, S. Meglio, F. Scippacercola, and L. L. L. Starace. "E2E-Loader: A Framework to Support Performance Testing of Web Applications". In: *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)* (2023), pp. 351–361. DOI: 10.1109/ICST57152.2023.00040.

[18] Scott Bekker and Editor in Chief. *Veeam Puts Hourly Downtime Cost at $85K*. Redmond Channel Partner. URL: https://rcpmag.com/blogs/scott-bekker/2021/03/veeam-hourly-downtime-cost.aspx (visited on 02/27/2023).

[19] Marc G. Bellemare, Will Dabney, and Rémi Munos. "A Distributional Perspective on Reinforcement Learning". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 449–458.

[20] Netflix Technology Blog. *Scryer: Netflix's Predictive Auto Scaling Engine — Part 2*. Medium. Apr. 18, 2017. URL: https://netflixtechblog.com/scryer-netflixs-predictive-auto-scaling-engine-part-2-bb9c4f9b9385 (visited on 02/26/2023).

[21] Peter Bodík, Rean Griffith, Charles Sutton, Armando Fox, Michael I. Jordan, and David A. Patterson. "Statistical Machine Learning Makes Automatic Control Practical for Internet Datacenters". In: *USENIX Workshop on Hot Topics in Cloud Computing*. 2009.

[22] Maycon Viana Bordin, Dalvan Griebler, G. Mencagli, C. Geyer, and L. G. Fernandes. "DSPBench: A Suite of Benchmark Applications for Distributed Data Stream Processing Systems". In: *IEEE Access* 8 (2020), pp. 222900–222917. DOI: 10.1109/ACCESS.2020.3043948.

[23] Maycon Viana Bordin, Dalvan Griebler, Gabriele Mencagli, Claudio Geyer, and Luiz Gustavo Fernandes. "DSPBench: A Suite of Benchmark Applications for Distributed Data Stream Processing Systems". In: *IEEE Access* 8 (2020), pp. 222900–222917.

[24] G. Box and Donald Behnken. "Some New Three Level Designs for the Study of Quantitative Variables". In: *Technometrics* 2 (Nov. 1960), pp. 455–475. DOI: 10.1080/00401706.1960.10489912.

[25] G. Box and J.S. Hunter. "MultiFactor Experimental Designs for Exploring Response Surfaces". In: *The Annals of Mathematical Statistics* 28 (Mar. 1957). DOI: 10.1214/aoms/1177707047.

[26] G. Box, Stuart Hunter, and William Hunter. *Statistics for experimenters. Design, innovation, and discovery. 2nd ed.* Vol. 2. Jan. 2005.

[27] George E P Box and Norman R Draper. *Empirical model-building and response surface*. USA: John Wiley & Sons, Inc., 1986. ISBN: 0471810339.

[28] Matthew Brookes, Vasiliki Kalavri, and John Liagouris. "FASTER State Management for Timely Dataflow". In: *Proceedings of Real-Time Business Intelligence and Analytics*. BIRTE 2019. Los Angeles, CA, USA: Association for Computing Machinery, 2019. ISBN: 9781450376600. DOI: 10.1145/3350489.3350493. URL: https://doi.org/10.1145/3350489.3350493.

[29] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. "Apache Flink™: Stream and Batch Processing in a Single Engine". In: (2015), p. 12.

[30] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. "Apache Flink™: Stream and Batch Processing in a Single Engine". In: *IEEE Data Engineering Bulletin* 38 (Jan. 2015).

[31] Valeria Cardellini, Francesco Lo Presti, Matteo Nardelli, and Gabriele Russo Russo. "Runtime adaptation of data stream processing systems: The state of the art". In: *ACM Computing Surveys* 54.11s (2022), pp. 1–36.

[32] J. M. Carlson and John Doyle. "Highly Optimized Tolerance: Robustness and Design in Complex Systems". en. In: *Physical Review Letters* 84.11 (Mar. 2000), pp. 2529–2532. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.84.2529. (Visited on 03/26/2022).

[33] Raul Castro Fernandez, Matteo Migliavacca, Evangelia Kalyvianaki, and Peter Pietzuch. "Integrating scale out and fault tolerance in stream processing using operator state management". In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. SIGMOD/PODS'13: International Conference on Management of Data. New York New York USA: ACM, June 22, 2013, pp. 725–736. ISBN: 978-1-4503-2037-5. DOI: 10.1145/2463676.2465282. URL: https://dl.acm.org/doi/10.1145/2463676.2465282 (visited on 02/26/2023).

[34] Uğur Çetintemel, Daniel Abadi, Yanif Ahmad, Hari Balakrishnan, Magdalena Balazinska, Mitch Cherniack, Jeong-Hyon Hwang, Samuel Madden, Anurag Maskey, Alexander Rasin, Esther Ryvkina, Mike Stonebraker, Nesime Tatbul, Ying Xing, and Stan Zdonik. "The Aurora and Borealis Stream Processing Engines". en. In: *Data Stream Management*. Ed. by Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Series Title: Data-Centric Systems and Applications. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 337–359. ISBN: 978-3-540-28607-3 978-3-540-28608-0. DOI: 10.1007/978-3-540-28608-0_17. URL: http://link.springer.com/10.1007/978-3-540-28608-0_17 (visited on 10/24/2021).

[35] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael Franklin, Joseph Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel Madden, Vijayshankar Raman, Frederick Reiss, and Mehul Shah. "Telegraphcq: Continuous dataflow processing for an uncertain world". In: Jan. 2003.

[36] Yingchao Cheng, Zhifeng Hao, and Ruichu Cai. "Auto-scaling for Real-time Stream Analytics on HPC Cloud". In: *Service Oriented Computing and Applications* 13 (June 2019). DOI: 10.1007/s11761-019-00262-0.

[37] Sanket Chintapalli, Derek Dagit, Bobby Evans, Reza Farivar, Thomas Graves, Mark Holderbaugh, Zhuo Liu, Kyle Nusbaum, Kishorkumar Patil, Boyang Jerry Peng, and Paul Poulosky. "Benchmarking Streaming Computation Engines: Storm, Flink and Spark Streaming". In: *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2016, pp. 1789–1792. DOI: 10.1109/IPDPSW.2016.138.

[38] Joaquin Chung, Mainak Adhikari, Satish Narayana Srirama, Eun-Sung Jung, and Rajkumar Kettimuthu. "Resource Management for Processing Wide Area Data Streams on Supercomputers". In: *2020 29th International Conference on Computer Communications and Networks (ICCCN)*. 2020, pp. 1–6. DOI: 10.1109/ICCCN49398.2020.9209669.

[39] Linda M. Collins, John J. Dziak, Kari C. Kugler, and Jessica B. Trail. "Factorial Experiments". en. In: *American Journal of Preventive Medicine* 47.4 (Oct. 2014), pp. 498–504. ISSN: 07493797. DOI: 10.1016/j.amepre.2014.06.021. URL: https://linkinghub.elsevier.com/retrieve/pii/S0749379714003250 (visited on 10/24/2021).

[40] M.E. Crovella and A. Bestavros. "Self-similarity in World Wide Web traffic: evidence and possible causes". In: *IEEE/ACM Transactions on Networking* 5.6 (Dec. 1997), pp. 835–846. ISSN: 10636692. DOI: 10.1109/90.650143. URL: http://ieeexplore.ieee.org/document/650143/ (visited on 04/13/2022).

[41] Miyuru Dayarathna and Srinath Perera. "Recent advancements in event processing". In: *ACM Computing Surveys (CSUR)* 51.2 (2018), pp. 1–36.

[42] Bonaventura Del Monte, Steffen Zeuch, Tilmann Rabl, and Volker Markl. "Rhino: Efficient management of very large distributed state for stream processing engines". In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 2471–2486.

[43] Dhananjay Dhananjay, Chinya Ravishankar, and Mitch Cherniack. "Real-time, load-adaptive processing of continuous queries over data streams". In: July 2008, pp. 277–288. DOI: 10.1145/1385989.1386024.

[44] Dmitry Duplyakin, Nikhil Ramesh, Carina Imburgia, Hamza Fathallah Al Sheikh, Semil Jain, Prikshit Tekta, Aleksander Maricq, Gary Wong, and Robert Ricci. "Avoiding the Ordering Trap in Systems Performance Measurement". In: *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. 2023, pp. 373–386.

[45] John F. Ehlers. *Rocket Science for Traders: Digital Signal Processing Applications*. USA: John Wiley & Sons, Inc., 2001. ISBN: 0471405671.

[46] A. S. C. Ehrenberg. In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28.1 (1979), pp. 79–83. ISSN: 00359254, 14679876. URL: http://www.jstor.org/stable/2346818 (visited on 08/12/2023).

[47] Simon Eismann, Diego Elias Costa, Lizhi Liao, Cor-Paul Bezemer, Weiyi Shang, André van Hoorn, and Samuel Kounev. "A case study on the stability of performance tests for serverless applications". In: *Journal of Systems and Software* 189 (2022), p. 111294.

[48] D. England, J. Weissman, and J. Sadagopan. "A new metric for robustness with application to job scheduling". en. In: *HPDC-14. Proceedings. 14th IEEE International Symposium on High Performance Distributed Computing, 2005*. IEEE, 2005, pp. 135–143. ISBN: 978-0-7803-9037-9. DOI: 10.1109/HPDC.2005.1520948. (Visited on 03/26/2022).

[49] Sonja Engmann and Denis Cousineau. "Comparing distributions: the two-sample Anderson–Darling test as an alternative to the Kolmogorov–Smirnov test". In: *Journal of Applied Quantitative Methods* 6 (Sept. 2011), pp. 1–17.

[50] TW Epps and Kenneth J Singleton. "An omnibus test for the two-sample problem using the empirical characteristic function". In: *Journal of Statistical Computation and Simulation* 26.3-4 (1986), pp. 177–203.

[51] A. Erramilli, O. Narayan, and W. Willinger. "Experimental queueing analysis with long-range dependent packet traffic". In: *IEEE/ACM Transactions on Networking* 4.2 (Apr. 1996), pp. 209–223. ISSN: 10636692. DOI: 10.1109/90.491008. URL: http://ieeexplore.ieee.org/document/491008/ (visited on 04/14/2022).

[52] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. "The Many Faces of Publish/Subscribe". In: *ACM Comput. Surv.* 35.2 (June 2003), pp. 114–131. ISSN: 0360-0300. DOI: 10.1145/857076.857078. URL: https://doi.org/10.1145/857076.857078.

[53] Vivian Fang, Lloyd Brown, William Lin, Wenting Zheng, Aurojit Panda, and Raluca Ada Popa. "CostCO: An automatic cost modeling framework for secure multi-party computation". In: *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2022, pp. 140–153.

[54] Avrilia Floratou, Ashvin Agrawal, Bill Graham, Sriram Rao, and Karthik Ramasamy. "Dhalion: self-regulating stream processing in heron". en. In: *Proceedings of the VLDB Endowment* 10.12 (Aug. 2017), pp. 1825–1836. ISSN: 2150-8097. DOI: 10.14778/3137765.3137786. URL: https://dl.acm.org/doi/10.14778/3137765.3137786 (visited on 10/24/2021).

[55] Daniel D. Frey, Fredrik Engelhardt, and Edward M. Greitzer. "A role for "one-factor-at-a-time" experimentation in parameter design". en. In: *Research in Engineering Design* 14.2 (May 2003), pp. 65–74. ISSN: 0934-9839. DOI: 10.1007/s00163-002-0026-9. (Visited on 04/02/2022).

[56] Tom Z. J. Fu, Jianbing Ding, Richard T. B. Ma, Marianne Winslett, Yin Yang, and Zhenjie Zhang. "DRS: Auto-Scaling for Real-Time Stream Analytics". en. In: *IEEE/ACM Transactions on Networking* 25.6 (Dec. 2017), pp. 3338–3352. ISSN: 1063-6692, 1558-2566. DOI: 10.1109/TNET.2017.2741969. URL: http://ieeexplore.ieee.org/document/8024162/ (visited on 10/24/2021).

[57] Anshul Gandhi, Parijat Dube, Alexei Karve, Andrzej Kochut, and Li Zhang. "Modeling the Impact of Workload on Cloud Resource Scaling". In: *Proceedings - Symposium on Computer Architecture and High Performance Computing* (Dec. 2014), pp. 310–317. DOI: 10.1109/SBAC-PAD.2014.16.

[58] Anshul Gandhi, Parijat Dube, Alexei A. Karve, Andrzej Kochut, and Li Zhang. "Adaptive, Model-driven Autoscaling for Cloud Applications". In: *International Conference on Automation and Computing*. 2014.

[59] Anshul Gandhi, Parijat Dube, Alexei A. Karve, Andrzej Kochut, and Li Zhang. "Adaptive, Model-driven Autoscaling for Cloud Applications". In: *11th International Conference on Autonomic Computing, ICAC '14, Philadelphia, PA, USA, June 18-20, 2014*. Ed. by Xiaoyun Zhu, Giuliano Casale, and Xiaohui Gu. USENIX Association, 2014, pp. 57–64. URL: https://www.usenix.org/conference/icac14/technical-sessions/presentation/gandhi.

[60] Anshul Gandhi, Mor Harchol-Balter, Ram Raghunathan, and Michael A. Kozuch. "AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers". In: *ACM Trans. Comput. Syst.* 30.4 (Nov. 2012). ISSN: 0734-2071. DOI: 10.1145/2382553.2382556. URL: https://doi.org/10.1145/2382553.2382556.

[61] Sandra Garcia-Rodriguez, Mohammad Alshaer, and C. Gouy-Pailler. "STREAMER: A Powerful Framework for Continuous Learning in Data Streams". In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management* (2020). DOI: 10.1145/3340531.3417427.

[62] Robert A. Gatenby and B. Roy Frieden. "Information Theory in Living Systems, Methods, Applications, and Challenges". In: *Bulletin of Mathematical Biology* 69.2 (Feb. 1, 2007), pp. 635–657. ISSN: 1522-9602. DOI: 10.1007/s11538-006-9141-5. URL: https://doi.org/10.1007/s11538-006-9141-5.

[63] Bugra Gedik, Scott Schneider, Martin Hirzel, and Kun-Lung Wu. "Elastic Scaling for Data Stream Processing". In: *Parallel and Distributed Systems, IEEE Transactions on* 25 (June 2014), pp. 1447–1463. DOI: 10.1109/TPDS.2013.295.

[64] Sandra Geisler. "Data Stream Management Systems". In: *Data Exchange, Information, and Streams*. 2013.

[65] Jeff Gemet. *App Store Downtime Cost Apple $25M in Sales*. The Mac Observer. 2015. URL: https://www.macobserver.com/tmo/article/app-store-downtime-cost-apple-25m-in-sales.html.

[66] Adem Efe Gencer, David Bindel, Emin Gün Sirer, and Robbert van Renesse. "Configuring distributed computations using response surfaces". In: *Proceedings of the 16th Annual Middleware Conference*. 2015, pp. 235–246.

[67] Ian Gergin, Bradley Simmons, and Marin Litoiu. "A Decentralized Autonomic Architecture for Performance Control in the Cloud". In: Mar. 2014, pp. 574–579. DOI: 10.1109/IC2E.2014.75.

[68] Hamoun Ghanbari, Bradley Simmons, Marin Litoiu, Cornel Barna, and Gabriel Iszlai. "Optimal Autoscaling in a IaaS Cloud". In: *Proceedings of the 9th International Conference on Autonomic Computing*. ICAC '12. San Jose, California, USA: Association for Computing Machinery, 2012, pp. 173–178. ISBN: 9781450315203. DOI: 10.1145/2371536.2371567. URL: https://doi.org/10.1145/2371536.2371567.

[69] Mostafa Ghobaei-Arani and Ali Shahidinejad. "An efficient resource provisioning approach for analyzing cloud workloads: a metaheuristic-based clustering approach". In: *The Journal of Supercomputing* 77 (Jan. 2021). DOI: 10.1007/s11227-020-03296-w.

[70] Alison L. Gibbs and Francis Edward Su. *On choosing and bounding probability metrics*. 2002. arXiv: math/0209021 [math.PR].

[71] Sebastian J. Goerg and Johannes Kaiser. "Nonparametric Testing of Distributions—the Epps–Singleton Two-Sample Test using the Empirical Characteristic Function". en. In: *The Stata Journal: Promoting communications on statistics and Stata* 9.3 (Sept. 2009), pp. 454–465. ISSN: 1536-867X, 1536-8734. DOI: 10.1177/1536867X0900900307. (Visited on 03/18/2022).

[72] Heitor Murilo Gomes, Jesse Read, Albert Bifet, Jean Paul Barddal, and Joao Gama. "Machine learning for streaming data: state of the art, challenges, and opportunities". In: *ACM SIGKDD Explorations Newsletter* 21 (Nov. 2019), pp. 6–22. DOI: 10.1145/3373464.3373470.

[73] M.E. Gomez and V. Santonja. "Self-similarity in I/O workload: analysis and modelling". In: *Workload Characterization: Methodology and Case Studies. Based on the First Workshop on Workload Characterization*. Workload Characterization: Methodology and Case Studies. Based on the First Workshop on Workload Characterization. Dallas, TX, USA: IEEE Comput. Soc, 1999, pp. 97–104. ISBN: 978-0-7695-0450-6. DOI: 10.1109/WWC.1998.809365. URL: http://ieeexplore.ieee.org/document/809365/ (visited on 04/13/2022).

[74] Martin Grambow, Fabian Lehmann, and David Bermbach. "Benchmarking the performance of microservice applications". In: *Proceedings of the 19th International Conference on Web Engineering*. Springer. 2019, pp. 283–298.

[75] S.D. Gribble. "Robustness in complex systems". en. In: *Proceedings Eighth Workshop on Hot Topics in Operating Systems*. IEEE Comput. Soc, 2001, pp. 21–26. ISBN: 978-0-7695-1040-8. DOI: 10.1109/HOTOS.2001.990056. (Visited on 03/26/2022).

[76] Steven D. Gribble, Gurmeet Singh Manku, Drew Roselli, Eric A. Brewer, Timothy J. Gibson, and Ethan L. Miller. "Self-Similarity in File Systems". In: *SIGMETRICS Perform. Eval. Rev.* 26.1 (June 1998), pp. 141–150. ISSN: 0163-5999. DOI: 10.1145/277858.277894. URL: https://doi.org/10.1145/277858.277894.

[77] Michele Guerriero, Damian Andrew Tamburri, and Elisabetta Di Nitto. "StreamGen: Model-Driven Development of Distributed Streaming Applications". In: *ACM Trans. Softw. Eng. Methodol.* 30.1 (Jan. 2021). ISSN: 1049-331X. DOI: 10.1145/3408895. URL: https://doi.org/10.1145/3408895.

[78] Vincenzo Gulisano, Ricardo Jimenez-Peris, Marta Patino-Martinez, Claudio Soriente, and Patrick Valduriez. "StreamCloud: An Elastic and Scalable Data Streaming System". en. In: *IEEE Transactions on Parallel and Distributed Systems* 23.12 (Dec. 2012), pp. 2351–2365. ISSN: 1045-9219. DOI: 10.1109/TPDS.2012.24. URL: http://ieeexplore.ieee.org/document/6127868/ (visited on 10/24/2021).

[79] Vincenzo Gulisano, Daniel Jorde, Ruben Mayer, Hannaneh Najdataei, and Dimitris Palyvos-Giannas. "The DEBS 2020 Grand Challenge". In: *Proceedings of the 14th ACM International Conference on Distributed and Event-Based Systems*. DEBS '20. Montreal, Quebec, Canada: Association for Computing Machinery, 2020, pp. 183–186. ISBN: 9781450380287. DOI: 10.1145/3401025.3402684. URL: https://doi.org/10.1145/3401025.3402684.

[80] Vincenzo Gulisano, Daniel Jorde, Ruben Mayer, Hannaneh Najdataei, and Dimitris Palyvos-Giannas. "The DEBS 2020 grand challenge". In: *Proceedings of the 14th ACM International Conference on Distributed and Event-Based Systems*. 2020, pp. 183–186.

[81] Ted Hahn and Mark Hahn. "Control Theory for SRE". In: Dublin: USENIX Association, Oct. 2019.

[82] Rui Han, Moustafa Ghanem, Li Guo, Yike Guo, and Michelle Osmond. "Enabling cost-aware and adaptive elasticity of multi-tier cloud applications". In: *Future Generation Computer Systems* 32 (Mar. 2014), pp. 82–98. DOI: 10.1016/j.future.2012.05.018.

[83] Rui Han, Li Guo, Moustafa M. Ghanem, and Yike Guo. "Lightweight Resource Scaling for Cloud Applications". In: *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). Ottawa, Canada: IEEE, May 2012, pp. 644–651. ISBN: 978-1-4673-1395-7 978-0-7695-4691-9. DOI: 10.1109/CCGrid.2012.52. URL: http://ieeexplore.ieee.org/document/6217477/ (visited on 07/03/2023).

[84] Mor Harchol-Baiter and Allen B Downeyt. "Exploiting Process Lifetime Distributions for Dynamic Load Balancing". en. In: (), p. 12.

[85] Mor Harchol-Balter and Allen Downey. "Exploiting Process Lifetime Distributions for Dynamic Load Balancing." In: *ACM Trans. Comput. Syst.* 15 (Aug. 1997), pp. 253–285. DOI: 10.1145/224056.225838.

[86] M. Heemskerk, M. Mandjes, and B. Mathijsen. "Staffing for many-server systems facing non-standard arrival processes". In: *arXiv:2006.00515 [cs, math]* (May 31, 2020). arXiv: 2006.00515. URL: http://arxiv.org/abs/2006.00515 (visited on 04/13/2022).

[87] Thomas Heinze, Yuanzhen Ji, Lars Roediger, Valerio Pappalardo, Andreas Meister, Zbigniew Jerzak, and Christof Fetzer. "FUGU: Elastic Data Stream Processing with Latency Constraints". In: *IEEE Data Eng. Bull.* 38.4 (2015), pp. 73–81. URL: http://sites.computer.org/debull/A15dec/p73.pdf.

[88] Herodotos Herodotou, Lambros Odysseos, Yuxing Chen, and Jiaheng Lu. "Automatic Performance Tuning for Distributed Data Stream Processing Systems". In: *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 2022, pp. 3194–3197. DOI: 10.1109/ICDE53745.2022.00296.

[89] Klaus Hinkelmann and Oscar Kempthorne. *Design and Analysis of Experiments, Volume 1: Introduction to Experimental Design*. 2nd ed. New York: John Wiley & Sons, 2008.

[90] Torsten Hoefler. "Benchmarking data science: 12 ways to lie with statistics and performance on parallel computers". In: *Computer* 55.8 (2022), pp. 49–56.

[91] Torsten Hoefler and Roberto Belli. “Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results”. In: *Proceedings of the international conference for high performance computing, networking, storage and analysis*. 2015, pp. 1–12.

[92] Moritz Hoffmann, Andrea Lattuada, Frank McSherry, Vasiliki Kalavri, John Liagouris, and Timothy Roscoe. “Megaphone: Latency-conscious state migration for distributed streaming dataflows”. In: *Proceedings of the VLDB Endowment* 12.9 (2019), pp. 1002–1015.

[93] Haruna Isah, Tariq Abughofa, Sazia Mahfuz, Dharmitha Ajerla, Farhana Zulkernine, and Shahzad Khan. “A Survey of Distributed Data Stream Processing Frameworks”. In: *IEEE Access* 7 (Oct. 2019), pp. 1–1. DOI: 10.1109/ACCESS.2019.2946884.

[94] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. “Empirical prediction models for adaptive resource provisioning in the cloud”. en. In: *Future Generation Computer Systems* 28.1 (Jan. 2012), pp. 155–162. ISSN: 0167739X. DOI: 10.1016/j.future.2011.05.027. URL: https://linkinghub.elsevier.com/retrieve/pii/S0167739X11001129 (visited on 10/24/2021).

[95] J P Morgan. “RiskMetrics Technical Document - Fourth Edition 1996, December”. In: (1996), p. 296. URL: https://www.msci.com/documents/10199/5915b101-4206-4ba0-aee2-3449d5c7e95a.

[96] Stuart Jamieson. “Dynamic Scaling of Distributed Data-Flows under Uncertainty”. In: *Proceedings of the 14th ACM International Conference on Distributed and Event-Based Systems*. DEBS ’20. Montreal, Quebec, Canada, 2020, pp. 230–233. ISBN: 9781450380287. DOI: 10.1145/3401025.3406444. URL: https://doi.org/10.1145/3401025.3406444.

[97] Stuart Jamieson and Matthew Forshaw. “Measuring Streaming System Robustness Using Non-parametric Goodness-of-Fit Tests”. In: *Computer Performance Engineering: 18th European Workshop, EPEW 2022, Santa Pola, Spain, September 21–23, 2022, Proceedings*. Springer. 2023, pp. 3–18.

[98] Stuart Jamieson and Matthew Forshaw. “On Improving Streaming System Autoscaler Behaviour using Windowing and Weighting Methods”. In: *Proceedings of the 17th ACM International Conference on Distributed and Event-Based Systems (DEBS)*. New York, NY, USA: Association for Computing Machinery, 2023. DOI: TBC. URL: TBC.

[99] Pooyan Jamshidi, Amir M. Sharifloo, Claus Pahl, Andreas Metzger, and Giovani Estrada. “Self-Learning Cloud Controllers: Fuzzy Q-Learning for Knowledge Evolution”. en. In: *2015 International Conference on Cloud and Autonomic Computing*. Boston, MA, USA: IEEE, Sept. 2015, pp. 208–211. ISBN: 978-1-4673-9566-3. DOI: 10.1109/ICCAC.2015.35. URL: http://ieeexplore.ieee.org/document/7312157/ (visited on 10/24/2021).

[100] Hung-Chin Jang and Shih-Yu Luo. “Enhancing Node Fault Tolerance through High-Availability Clusters in Kubernetes”. In: *2023 IEEE 3rd International Conference on Electronic Communications, Internet of Things and Big Data (ICEIB)*. 2023, pp. 30–35. DOI: 10.1109/ICEIB57887.2023.10170110.

[101] Erica Jen. “Stable or robust? What’s the difference?” In: *Complex*. 8 (2003), pp. 12–18.

[102] Mikkel T. Jensen. “Improving robustness and flexibility of tardiness and total flow-time job shops using robustness measures”. en. In: *Applied Soft Computing* 1.1 (June 2001), pp. 35–52. ISSN: 15684946. DOI: 10.1016/S1568-4946(01)00005-9. (Visited on 03/26/2022).

[103] Jing Jiang, Jie Lu, Guangquan Zhang, and Guodong Long. “Optimal Cloud Resource Auto-Scaling for Web Applications”. In: *Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. CCGRID ’13. Delft, Netherlands: IEEE Press, 2013, pp. 58–65. ISBN: 9780768549965. DOI: 10.1109/CCGrid.2013.73. URL: https://doi.org/10.1109/CCGrid.2013.73.

[104] Kevin Johnson. *Pandas TA - A Technical Analysis Library in Python 3*. original-date: 2019-02-19T16:41:09Z. 2022. URL: https://github.com/twopirllc/pandas-ta/blob/084dbe1c4b76082f383fa3029270ea9ac35e4dc7/pandas_ta/overlap/ (visited on 10/28/2022).

[105] Artjom Joosen, Ahmed Hassan, Martin Asenov, Rajkarn Singh, Luke Darlow, Jianfeng Wang, and Adam Barker. “How Does It Function? Characterizing Long-term Trends in Production Serverless Workloads”. In: *Proceedings of the 2023 ACM Symposium on Cloud Computing*. SoCC ’23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 443–458. ISBN: 9798400703874. DOI: 10.1145/3620678.3624783. URL: https://doi.org/10.1145/3620678.3624783.

[106] V. Jorge Leon, S. David Wu, and Robert H. Storer. “ROBUSTNESS MEASURES AND ROBUST SCHEDULING FOR JOB SHOPS”. en. In: *IIE Transactions* 26.5 (Sept. 1994), pp. 32–43. ISSN: 0740-817X, 1545-8830. DOI: 10.1080/07408179408966626. (Visited on 03/26/2022).

[107] Vasiliki Kalavri and John Liagouris. “In support of workload-aware streaming state management”. In: *Proceedings of the 12th USENIX Conference on Hot Topics in Storage and File Systems*. 2020, pp. 19–19.

[108] Vasiliki Kalavri, John Liagouris, Moritz Hoffmann, Desislava Dimitrova, Matthew Forshaw, and Timothy Roscoe. “Three steps is all you need: fast, accurate, automatic scaling decisions for distributed streaming dataflows”. In: *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*. OSDI’18. Carlsbad, CA, USA: USENIX Association, 2018, pp. 783–798. ISBN: 9781931971478.

[109] Supun Kamburugamuve, Geoffrey Fox, Judy Qiu, and David Leake. “Survey of Distributed Stream Processing for Large Stream Sources”. In: (Dec. 2014). DOI: 10.13140/RG.2.1.2938.7927.

[110] Leonid Kantorovich and Gennady S. Rubinstein. “On a space of totally additive functions”. In: *Vestnik Leningrad. Univ* 13 (1958), pp. 52–59.

[111] Jeyhun Karimov, Tilmann Rabl, Asterios Katsifodimos, Roman Samarev, Henri Heiskanen, and Volker Markl. “Benchmarking Distributed Stream Data Processing Systems”. en. In: *2018 IEEE 34th International Conference on Data Engineering (ICDE)* (Apr. 2018). arXiv: 1802.08496, pp. 1507–1518. DOI: 10.1109/ICDE.2018.00169. URL: http://arxiv.org/abs/1802.08496 (visited on 10/24/2021).

[112] P.J. Kaufman. *Trading Systems and Methods*. Wiley finance series. John Wiley & Sons, Incorporated, 2013. ISBN: 978-1-119-20256-1. URL: https://books.google.com.gi/books?id=j%5C_YnswEACAAJ.

[113] Pankaj Kaur and Inderveer Chana. “A resource elasticity framework for QoS-aware execution of cloud applications”. In: *Future Generation Computer Systems* 37 (July 2014). DOI: 10.1016/j.future.2014.02.018.

[114] J.O. Kephart and D.M. Chess. “The vision of autonomic computing”. en. In: *Computer* 36.1 (Jan. 2003), pp. 41–50. ISSN: 0018-9162. DOI: 10.1109/MC.2003.1160055. URL: http://ieeexplore.ieee.org/document/1160055/ (visited on 10/25/2021).

[115] Sabrine Khriji, Yahia Benbelgacem, Rym Chéour, Dhouha El Houssaini, and Olfa Kanoun. “Design and implementation of a cloud-based event-driven architecture for real-time data processing in wireless sensor networks”. In: *The Journal of Supercomputing* 78.3 (Feb. 2022), pp. 3374–3401. ISSN: 0920-8542, 1573-0484. DOI: 10.1007/s11227-021-03955-6. URL: https://link.springer.com/10.1007/s11227-021-03955-6 (visited on 06/20/2023).

[116] Mariam Kiran, Peter Murphy, Inder Monga, Jon Dugan, and Sartaj Singh Baveja. “Lambda architecture for cost-effective batch and speed big data processing”. In: *2015 IEEE International Conference on Big Data (Big Data)*. 2015 IEEE International Conference on Big Data (Big Data). Santa Clara, CA, USA: IEEE, Oct. 2015, pp. 2785–2792. ISBN: 978-1-4799-9926-2. DOI: 10.1109/BigData.2015.7364082. URL: http://ieeexplore.ieee.org/document/7364082/ (visited on 04/15/2023).

[117] SAMUEL. KOUNEV. *SYSTEMS BENCHMARKING: For Scientists and Engineers*. SPRINGER, 2021.

[118] Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M. Patel, Karthik Ramasamy, and Siddarth Taneja. “Twitter Heron: Stream Processing at Scale”. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’15. Melbourne, Victoria, Australia: Association for Computing Machinery, 2015, pp. 239–250. ISBN: 9781450327589. DOI: 10.1145/2723372.2742788. URL: https://doi.org/10.1145/2723372.2742788.

[119] S. Kullback and R. A. Leibler. “On Information and Sufficiency”. en. In: *The Annals of Mathematical Statistics* 22.1 (Mar. 1951), pp. 79–86. ISSN: 0003-4851. DOI: 10.1214/aoms/1177729694. (Visited on 05/08/2022).

[120] Mayuresh Kunjir and Shivnath Babu. “Black or White? How to Develop an AutoTuner for Memory-based Analytics”. In: *Proceedings of the 2020 International Conference on Management of Data*. 2020, pp. 1667–1683.

[121] Mayuresh Kunjir and Shivnath Babu. “Black or White? How to Develop an AutoTuner for Memory-based Analytics”. In: *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. Ed. by David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo. ACM, 2020, pp. 1667–1683. DOI: 10.1145/3318464.3380591. URL: https://doi.org/10.1145/3318464.3380591.

[122] Reese Kuper, Ipoom Jeong, Yifan Yuan, Jiayu Hu, Ren Wang, Narayan Ranganathan, and Nam Sung Kim. *A Quantitative Analysis and Guideline of Data Streaming Accelerator in Intel 4th Gen Xeon Scalable Processors*. 2023. arXiv: 2305.02480 [cs.AR].

[123] Palden Lama and Xiaobo Zhou. “Efficient server provisioning with end-to-end delay guarantee on multi-tier clusters”. In: Aug. 2009, pp. 1–9. DOI: 10.1109/IWQoS.2009.5201420.

[124] Averill M. Law. "A tutorial on design of experiments for simulation modeling". en. In: *Proceedings of the Winter Simulation Conference 2014*. Savanah, GA, USA: IEEE, Dec. 2014, pp. 66–80. ISBN: 978-1-4799-7486-3 978-1-4799-7484-9. DOI: 10.1109/WSC.2014.7019878. URL: http://ieeexplore.ieee.org/document/7019878/ (visited on 10/24/2021).

[125] W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson. "On the self-similar nature of Ethernet traffic (extended version)". In: *IEEE/ACM Transactions on Networking* 2.1 (Feb. 1994), pp. 1–15. ISSN: 10636692. DOI: 10.1109/90.282603. URL: http://ieeexplore.ieee.org/document/282603/ (visited on 04/13/2022).

[126] Stacy Liberatore. *Zuckerberg's million dollar outage: Finance expert reveals Meta lost roughly $100 MILLION during two-hour glitch that took down Facebook, Instagram and Messenger*. Mail Online. 2024. URL: https://www.dailymail.co.uk/sciencetech/article-13161047/meta-lost-millions-facebook-instagram-outage.html.

[127] Harold C. Lim, Shivnath Babu, and Jeffrey S. Chase. "Automated Control for Elastic Storage". In: *Proceedings of the 7th International Conference on Autonomic Computing*. ICAC '10. Washington, DC, USA: Association for Computing Machinery, 2010, pp. 1–10. ISBN: 9781450300742. DOI: 10.1145/1809049.1809051. URL: https://doi.org/10.1145/1809049.1809051.

[128] Harold C. Lim, Shivnath Babu, Jeffrey S. Chase, and Sujay S. Parekh. "Automated Control in Cloud Computing: Challenges and Opportunities". In: *Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds*. ACDC '09. Barcelona, Spain: Association for Computing Machinery, 2009, pp. 13–18. ISBN: 9781605585857. DOI: 10.1145/1555271.1555275. URL: https://doi.org/10.1145/1555271.1555275.

[129] Virginia Lo, Jens Mache, and Kurt Windisch. "A comparative study of real workload traces and synthetic workload models for parallel job scheduling". en. In: *Job Scheduling Strategies for Parallel Processing*. Ed. by Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Dror G. Feitelson, and Larry Rudolph. Vol. 1459. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 25–46. ISBN: 978-3-540-64825-3 978-3-540-68536-4. DOI: 10.1007/BFb0053979. URL: http://link.springer.com/10.1007/BFb0053979 (visited on 10/24/2021).

[130] Björn Lohrmann, Daniel Warneke, and Odej Kao. "Nephele streaming: stream processing under QoS constraints at scale". en. In: *Cluster Computing* 17.1 (Mar. 2014), pp. 61–78. ISSN: 1386-7857, 1573-7543. DOI: 10.1007/s10586-013-0281-8. URL: http://link.springer.com/10.1007/s10586-013-0281-8 (visited on 10/24/2021).

[131] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A. Lozano. "A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments". en. In: *Journal of Grid Computing* 12.4 (Dec. 2014), pp. 559–592. ISSN: 1570-7873, 1572-9184. DOI: 10.1007/s10723-014-9314-7. URL: http://link.springer.com/10.1007/s10723-014-9314-7 (visited on 10/24/2021).

[132] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A. Lozano. "A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments". In: *Journal of Grid Computing* 12.4 (2014), pp. 559–592. ISSN: 1570-7873, 1572-9184. DOI: 10.1007/s10723-014-9314-7. URL: http://link.springer.com/10.1007/s10723-014-9314-7 (visited on 04/13/2022).

[133] Google Cloud Solutions Manager. *Anomaly detection using streaming analytics & AI*. Apr. 2, 2024. URL: https://cloud.google.com/blog/products/data-analytics/anomaly-detection-using-streaming-analytics-and-ai (visited on 04/29/2024).

[134] Alfeu Martinho, Henrique Hippert, and Leonardo Goliatt. "Short-term streamflow modeling using data-intelligence evolutionary machine learning models". In: *Scientific Reports* 13 (Aug. 2023). DOI: 10.1038/s41598-023-41113-5.

[135] Peter Mell and Timothy Grance. "The NIST Definition of Cloud Computing". en. In: (), p. 7.

[136] Richard von Mises. *Wahrscheinlichkeit Statistik und Wahrheit*. de. Berlin, Heidelberg: Springer Berlin Heidelberg, 1928. ISBN: 978-3-662-35402-5 978-3-662-36230-3. DOI: 10.1007/978-3-662-36230-3. (Visited on 03/26/2022).

[137] Douglas C. Montgomery. *Design and analysis of experiments*. en. Eighth edition. Hoboken, NJ: John Wiley & Sons, Inc, 2013. ISBN: 978-1-118-14692-7.

[138] Janakiram MSV. *All the Apache Streaming Projects: An Exploratory Guide*. 2016. URL: https://thenewstack.io/apache-streaming-projects-exploratory-guide/ (visited on 06/20/2023).

[139] Feiping Nie, Hu Zhanxuan, and Xuelong Li. "An investigation for loss functions widely used in machine learning". In: *Communications in Information and Systems* 18 (Jan. 2018), pp. 37–52. DOI: 10.4310/CIS.2018.v18.n1.a2.

[140] Matthew Nokleby, Haroon Raja, and Waheed U. Bajwa. "Scaling-Up Distributed Processing of Data Streams for Machine Learning". In: *Proceedings of the IEEE* 108.11 (2020), pp. 1984–2012. DOI: 10.1109/JPROC.2020.3021381.

[141] Gary W. Oehlert. *A first course in design and analysis of experiments*. en. New York: W.H. Freeman, 2000. ISBN: 978-0-7167-3510-6.

[142] Jorge L Ortega-Arjona and Departamento de Matem. "The Parallel Pipes and Filters Pattern". In: ().

[143] Soumaya Ounacer, Mohamed Amine, Soufiane Ardchir, Abderrahmane Daif, and Mohamed Azouazi. "A New Architecture for Real Time Data Stream Processing". In: *International Journal of Advanced Computer Science and Applications* 8.11 (2017). ISSN: 21565570, 2158107X. DOI: 10.14569/IJACSA.2017.081106. URL: http://thesai.org/Publications/ViewPaper?Volume=8%5C&Issue=11%5C&Code=ijacsa%5C&SerialNo=6 (visited on 04/15/2023).

[144] Pradeep Padala, Kai-Yuan Hou, Kang G. Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, and Arif Merchant. "Automated Control of Multiple Virtualized Resources". In: *Proceedings of the 4th ACM European Conference on Computer Systems*. EuroSys '09. Nuremberg, Germany: Association for Computing Machinery, 2009, pp. 13–26. ISBN: 9781605584829. DOI: 10.1145/1519065.1519068. URL: https://doi.org/10.1145/1519065.1519068.

[145] Kayaroganam Palanikumar. "Introductory Chapter: Response Surface Methodology in Engineering Science". In: *Response Surface Methodology in Engineering Science*. Ed. by Palanikumar Kayaroganam. Rijeka: IntechOpen, 2021. Chap. 1. DOI: 10.5772/intechopen.100484. URL: https://doi.org/10.5772/intechopen.100484.

[146] Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee. "A Design Science Research Methodology for Information Systems Research". en. In: *Journal of Management Information Systems* 24.3 (Dec. 2007), pp. 45–77. ISSN: 0742-1222, 1557-928X. DOI: 10.2753/MIS0742-1222240302. URL: https://www.tandfonline.com/doi/full/10.2753/MIS0742-1222240302 (visited on 12/10/2021).

[147] Max Petrov, Nikolay Butakov, Denis Nasonov, and Mikhail Melnik. "Adaptive performance model for dynamic scaling Apache Spark Streaming". en. In: *Procedia Computer Science* 136 (2018), pp. 109–117. ISSN: 18770509. DOI: 10.1016/j.procs.2018.08.243. URL: https://linkinghub.elsevier.com/retrieve/pii/S1877050918315485 (visited on 10/24/2021).

[148] Rekha Pitchumani, Shayna Frank, and Ethan L. Miller. "Realistic request arrival generation in storage benchmarks". en. In: *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*. Santa Clara, CA, USA: IEEE, May 2015, pp. 1–10. ISBN: 978-1-4673-7619-8. DOI: 10.1109/MSST.2015.7208286. URL: http://ieeexplore.ieee.org/document/7208286/ (visited on 10/24/2021).

[149] Chris Pleasance. *Facebook outage: Company lost '$100million' during seven-hour blackout*. Mail Online. 2021. URL: https://www.dailymail.co.uk/news/article-10060287/Facebook-outage-Zuckerberg-apologises-site-come-online.html.

[150] Meikel Poess, Bryan Smith, Lubor Kollar, and Per-Ake Larson. "TPC-DS: Taking decision support benchmarking to the next level". In: *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. 2002, pp. 582–587.

[151] Pravega. *When speeding makes sense â€" Fast, consistent, durable and scalable streaming data with Pravega*. English. Oct. 2020. URL: https://cncf.pravega.io/blog/2020/10/01/when-speeding-makes-sense-fast-consistent-durable-and-scalable-streaming-data-with-pravega/ (visited on 12/26/2021).

[152] Chenhao Qu, Rodrigo Calheiros, and Rajkumar Buyya. "A Reliable and Cost-Efficient Auto-Scaling System for Web Applications Using Heterogeneous Spot Instances". In: *Journal of Network and Computer Applications* 65 (Mar. 2016), pp. 167–180. DOI: 10.1016/j.jnca.2016.03.001.

[153] Chenhao Qu, Rodrigo N. Calheiros, and Rajkumar Buyya. "Auto-Scaling Web Applications in Clouds: A Taxonomy and Survey". In: *ACM Computing Surveys* 51.4 (July 31, 2019), pp. 1–33. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3148149. URL: https://dl.acm.org/doi/10.1145/3148149 (visited on 07/01/2023).

[154] E.G. Radhika and G. Sudha Sadasivam. "A review on prediction based autoscaling techniques for heterogeneous applications in cloud environment". In: *Materials Today: Proceedings* 45 (2021). International Conference on Advances in Materials Research - 2019, pp. 2793–2800. ISSN: 2214-7853. DOI: https://doi.org/10.1016/j.matpr.2020.11.789. URL: https://www.sciencedirect.com/science/article/pii/S2214785320394657.

[155] Aistis Raudys, Vaidotas Lenčiauskas, and Edmundas Malčius. "Moving Averages for Financial Data Smoothing". In: *Communications in Computer and Information Science*. Communications in Computer and Information Science, 2013, pp. 34–45. DOI: 10.1007/978-3-642-41947-8_4.

[156] Aistis Raudys, Vaidotas Lenčiauskas, and Edmundas Malčius. “Moving Averages for Financial Data Smoothing”. In: *Information and Software Technologies*. Ed. by Tomas Skersys, Rimantas Butleris, and Rita Butkiene. Vol. 403. Series Title: Communications in Computer and Information Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 34–45. ISBN: 978-3-642-41946-1 978-3-642-41947-8. DOI: 10.1007/978-3-642-41947-8_4. URL: http://link.springer.com/10.1007/978-3-642-41947-8_4 (visited on 08/17/2022).

[157] E. Riedel, A. Riska, E. Riedel, and A. Riska. “Long-Range Dependence at the Disk Drive Level”. In: *Third International Conference on the Quantitative Evaluation of Systems - (QEST'06)*. 2006, pp. 41–50. DOI: 10.1109/QEST.2006.27.

[158] Matthew Roughan, Albert Greenberg, Charles Kalmanek, Michael Rumsewicz, Jennifer Yates, and Yin Zhang. “Experience in Measuring Backbone Traffic Variability: Models, Metrics, Measurements and Meaning”. In: *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment*. IMW '02. Marseille, France: Association for Computing Machinery, 2002, pp. 91–92. ISBN: 158113603X. DOI: 10.1145/637201.637213. URL: https://doi.org/10.1145/637201.637213.

[159] Gabriele Russo Russo, Valeria Cardellini, and Francesco Lo Presti. “Reinforcement Learning Based Policies for Elastic Stream Processing on Heterogeneous Resources”. In: June 2019, pp. 31–42. DOI: 10.1145/3328905.3329506.

[160] Matthias J. Sax. “Apache Kafka”. In: *Encyclopedia of Big Data Technologies*. Ed. by Sherif Sakr and Albert Zomaya. Cham: Springer International Publishing, 2018, pp. 1–8. ISBN: 978-3-319-63962-8. DOI: 10.1007/978-3-319-63962-8_196-1. URL: https://doi.org/10.1007/978-3-319-63962-8_196-1.

[161] Deepika Saxena and Ashutosh Kumar Singh. “A proactive autoscaling and energy-efficient VM allocation framework using online multi-resource neural network for cloud data center”. In: *Neurocomputing* 426 (2021), pp. 248–264. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2020.08.076. URL: https://www.sciencedirect.com/science/article/pii/S0925231220315873.

[162] Eman Shaikh, Iman Mohiuddin, Yasmeen Alufaisan, and Irum Nahvi. “Apache Spark: A Big Data Processing Engine”. In: *2019 2nd IEEE Middle East and North Africa COMMunications Conference (MENACOMM)*. 2019 2nd IEEE Middle East and North Africa COMMunications Conference (MENACOMM). Manama, Bahrain: IEEE, 2019, pp. 1–6. ISBN: 978-1-72813-687-5. DOI: 10.1109/MENACOMM46666.2019.8988541. URL: https://ieeexplore.ieee.org/document/8988541/ (visited on 12/01/2022).

[163] Ziad A. Al-Sharif, Yaser Jararweh, Ahmad Al-Dahoud, and Luay M. Alawneh. “ACCRS: autonomic based cloud computing resource scaling”. en. In: *Cluster Computing* 20.3 (Sept. 2017), pp. 2479–2488. ISSN: 1386-7857, 1573-7543. DOI: 10.1007/s10586-016-0682-6. URL: http://link.springer.com/10.1007/s10586-016-0682-6 (visited on 10/24/2021).

[164] Piyush Shivam, Varun Marupadi, Jeff Chase, Thileepan Subramaniam, and Shivnath Babu. “Cutting corners: Workbench automation for server benchmarking”. In: *2008 USENIX Annual Technical Conference (USENIX ATC 08)*. 2008.

[165] Rahul Singh, Upendra Sharma, Emmanuel Cecchet, and Prashant Shenoy. "Autonomic Mix-Aware Provisioning for Non-Stationary Data Center Workloads". In: *Proceedings of the 7th International Conference on Autonomic Computing*. ICAC '10. Washington, DC, USA: Association for Computing Machinery, 2010, pp. 21–30. ISBN: 9781450300742. DOI: 10.1145/1809049.1809053. URL: https://doi.org/10.1145/1809049.1809053.

[166] Tomas Skersys, Rimantas Butleris, and Rita Butkiene, eds. *Information and Software Technologies: 19th International Conference, ICIST 2013, Kaunas, Lithuania, October 2013. Proceedings*. Vol. 403. Communications in Computer and Information Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. ISBN: 978-3-642-41946-1 978-3-642-41947-8. DOI: 10.1007/978-3-642-41947-8. URL: http://link.springer.com/10.1007/978-3-642-41947-8 (visited on 08/17/2022).

[167] Vladimir Sladojević, Sebastian Frischbier, Alexander Echler, Mario Paic, and Alessandro Margara. "Deriving a Realistic Workload Model to Simulate High-Volume Financial Data Feeds for Performance Benchmarking". In: *Proceedings of the 16th ACM International Conference on Distributed and Event-Based Systems*. DEBS '22. Copenhagen, Denmark, 2022, pp. 126–131. ISBN: 9781450393089. DOI: 10.1145/3524860.3539653. URL: https://doi.org/10.1145/3524860.3539653.

[168] Nikolai V Smirnov. "On the Estimation of the Discrepancy Between Empirical Curves of Distribution for Two Independent Samples". In: *Bull. Math. Uni. Moscou* 2.2 (1939), pp. 3–14.

[169] Simon Spinner, Samuel Kounev, Xiaoyun Zhu, Lei Lu, Mustafa Uysal, Anne Holler, and Rean Griffith. "Runtime Vertical Scaling of Virtualized Applications via Online Model Estimation". In: vol. 2014. Sept. 2014. DOI: 10.1109/SASO.2014.29.

[170] Kunwadee Sripanidkulchai, Bruce Maggs, and Hui Zhang. "An analysis of live streaming workloads on the internet". en. In: *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement - IMC '04*. Taormina, Sicily, Italy: ACM Press, 2004, p. 41. ISBN: 978-1-58113-821-4. DOI: 10.1145/1028788.1028795. URL: http://portal.acm.org/citation.cfm?doid=1028788.1028795 (visited on 10/24/2021).

[171] Laura Stevens. *Amazon Finds the Cause of Its AWS Outage: A Typo*. 2017. URL: https://www.wsj.com/articles/amazon-finds-the-cause-of-its-aws-outage-a-typo-1488490506.

[172] Martin Straesser, Simon Eismann, Jóakim von Kistowski, André Bauer, and Samuel Kounev. "Autoscaler Evaluation and Configuration: A Practitioner's Guideline". In: *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*. ICPE '23. Coimbra, Portugal: Association for Computing Machinery, 2023, pp. 31–41. ISBN: 9798400700682. DOI: 10.1145/3578244.3583721. URL: https://doi.org/10.1145/3578244.3583721.

[173] Martin Straesser, Simon Eismann, Jóakim von Kistowski, André Bauer, and Samuel Kounev. "Autoscaler Evaluation and Configuration: A Practitioner's Guideline (Author Preprint)". In: (2023).

[174] *STREAM: The Stanford Stream Data Manager*. Stanford University. URL: http://infolab.stanford.edu/stream/ (visited on 11/08/2023).

[175] Guoxin Su, Li Liu, Minjie Zhang, and David S. Rosenblum. “Quantitative Verification for Monitoring Event-Streaming Systems”. In: *IEEE Transactions on Software Engineering* 48 (2022), pp. 538–550. DOI: 10.1109/tse.2020.2996033.

[176] Tejas Subramanya and Roberto Riggio. “Machine Learning-Driven Scaling and Placement of Virtual Network Functions at the Network Edges”. In: *2019 IEEE Conference on Network Softwarization (NetSoft)*. 2019, pp. 414–422. DOI: 10.1109/NETSOFT.2019.8806631.

[177] Salman Taherizadeh and Vlado Stankovski. “Dynamic Multi-level Auto-scaling Rules for Containerized Applications”. en. In: *The Computer Journal* 62.2 (Feb. 2019). Ed. by Jin-Hee Cho, pp. 174–197. ISSN: 0010-4620, 1460-2067. DOI: 10.1093/comjnl/bxy043. URL: https://academic.oup.com/comjnl/article/62/2/174/4993728 (visited on 10/24/2021).

[178] *The Cost of Downtime*. Andrew Lerner. July 16, 2014. URL: https://blogs.gartner.com/andrew-lerner/2014/07/16/the-cost-of-downtime/ (visited on 02/27/2023).

[179] Richard Tibbetts. “Linear Road : benchmarking stream-based data management systems”. In: (June 2005).

[180] Tim Tillson. “Smoothing Techniques For More Accurate Signals”. In: *Technical Analysis of STOCKS & COMMODITIES* 16.1 (1998), pp. 33–37.

[181] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M. Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, Nikunj Bhagat, Sailesh Mittal, and Dmitriy Ryaboy. “Storm@twitter”. In: *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. SIGMOD-/PODS’14: International Conference on Management of Data. Snowbird Utah USA: ACM, 2014, pp. 147–156. ISBN: 978-1-4503-2376-5. DOI: 10.1145/2588555.2595641. URL: https://dl.acm.org/doi/10.1145/2588555.2595641 (visited on 12/01/2022).

[182] D. Tsoumakos, I. Konstantinou, C. Boumpouka, S. Sioutas, and N. Koziris. “Automated, Elastic Resource Provisioning for NoSQL Clusters Using TIRAMOLA”. en. In: *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. Delft: IEEE, May 2013, pp. 34–41. ISBN: 978-0-7695-4996-5 978-1-4673-6465-2. DOI: 10.1109/CCGrid.2013.45. URL: http://ieeexplore.ieee.org/document/6546056/ (visited on 10/24/2021).

[183] Peter A. Tucker, Kristin Tufte, Vassilis Papadimos, and David Maier. “NEXMark – A Benchmark for Queries over Data Streams DRAFT”. In: 2002.

[184] J.W. Tukey. “Exploratory Data Analysis”. In: *Reading* 26 (Jan. 1977).

[185] Vinay Kumar Vennu and Sai Ram Yepuru. “A performance study for autoscaling big data analytics containerized applications”. In: (2022), p. 50.

[186] Adriano Vogel, Dalvan Griebler, Marco Danelutto, and Luiz Gustavo Fernandes. “Self-adaptation on parallel stream processing: A systematic review”. In: *Concurrency and Computation: Practice and Experience* 34.6 (2022), e6759.

[187] Kai Waehner. *Kai Waehner*. URL: https://www.kai-waehner.de/blog/author/kai-waehner/ (visited on 04/26/2024).

[188] Guozhang Wang, Lei Chen, Ayusman Dikshit, Jason Gustafson, Boyang Chen, Matthias J. Sax, John Roesler, Sophie Blee-Goldman, Bruno Cadonna, Apurva Mehta, Varun Madan, and Jun Rao. "Consistency and Completeness: Rethinking Distributed Stream Processing in Apache Kafka". In: *Proceedings of the 2021 International Conference on Management of Data*. SIGMOD '21. Virtual Event, China: Association for Computing Machinery, 2021, pp. 2602–2613. ISBN: 9781450383431. DOI: 10.1145/3448016.3457556. URL: https://doi.org/10.1145/3448016.3457556.

[189] Xiaotong Wang. "A comprehensive study on fault tolerance in stream processing systems". In: *Frontiers of Computer Science* 16 (Sept. 2020). DOI: 10.1007/s11704-020-0248-x.

[190] Yuanli Wang, Baiqing Lyu, and Vasiliki Kalavri. "The non-expert tax: quantifying the cost of auto-scaling in cloud-based data stream analytics". In: *Proceedings of The International Workshop on Big Data in Emergent Distributed Environments*. 2022, pp. 1–7.

[191] Aws Whitepaper. "Amazon Web Services: Overview of Security Processes". In: 2008.

[192] J.W. Wilder. *New Concepts in Technical Trading Systems*. Trend Research, 1978. ISBN: 978-0-89459-027-6. URL: https://books.google.co.uk/books?id=WesJAQAAMAAJ.

[193] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[194] Jielong Xu, Zhenhua Chen, Jian Tang, and Sen Su. "T-storm: Traffic-aware online scheduling in storm". English (US). In: *Proceedings - International Conference on Distributed Computing Systems*. Proceedings - International Conference on Distributed Computing Systems. Publisher Copyright: © 2014 IEEE.; 2014 IEEE 34th International Conference on Distributed Computing Systems, ICDCS 2014 ; Conference date: 30-06-2014 Through 03-07-2014. Institute of Electrical and Electronics Engineers Inc., Aug. 2014, pp. 535–544. DOI: 10.1109/ICDCS.2014.61.

[195] Le Xu, Boyang Peng, and Indranil Gupta. "Stela: Enabling Stream Processing Systems to Scale-in and Scale-out On-demand". en. In: *2016 IEEE International Conference on Cloud Engineering (IC2E)*. Berlin, Germany: IEEE, Apr. 2016, pp. 22–31. ISBN: 978-1-5090-1961-8. DOI: 10.1109/IC2E.2016.38. URL: http://ieeexplore.ieee.org/document/7484160/ (visited on 10/24/2021).

[196] Piyush Yadav and Edward Curry. *Poster: Challenges in Complex Event Processing for Multimedia Data Streams*. Sept. 2018. DOI: 10.13140/RG.2.2.28894.15685.

[197] F. Yates. "Sir Ronald Fisher and the Design of Experiments". In: *Biometrics* 20.2 (1964), pp. 307–321. ISSN: 0006341X, 15410420. URL: http://www.jstor.org/stable/2528399 (visited on 07/23/2023).

[198] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. "Apache Spark: A Unified Engine for Big Data Processing". In: *Commun. ACM* 59.11 (Oct. 2016), pp. 56–65. ISSN: 0001-0782. DOI: 10.1145/2934664. URL: https://doi.org/10.1145/2934664.

[199] Liang Zhang, Wenli Zheng, Chao Li, Yao Shen, and Minyi Guo. “AuTraScale: An Automated and Transfer Learning Solution for Streaming System Auto-Scaling”. In: *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS). Portland, OR, USA: IEEE, 2021, pp. 912–921. ISBN: 978-1-66544-066-0. DOI: 10.1109/IPDPS49936.2021.00100. URL: https://ieeexplore.ieee.org/document/9460552/ (visited on 05/22/2022).