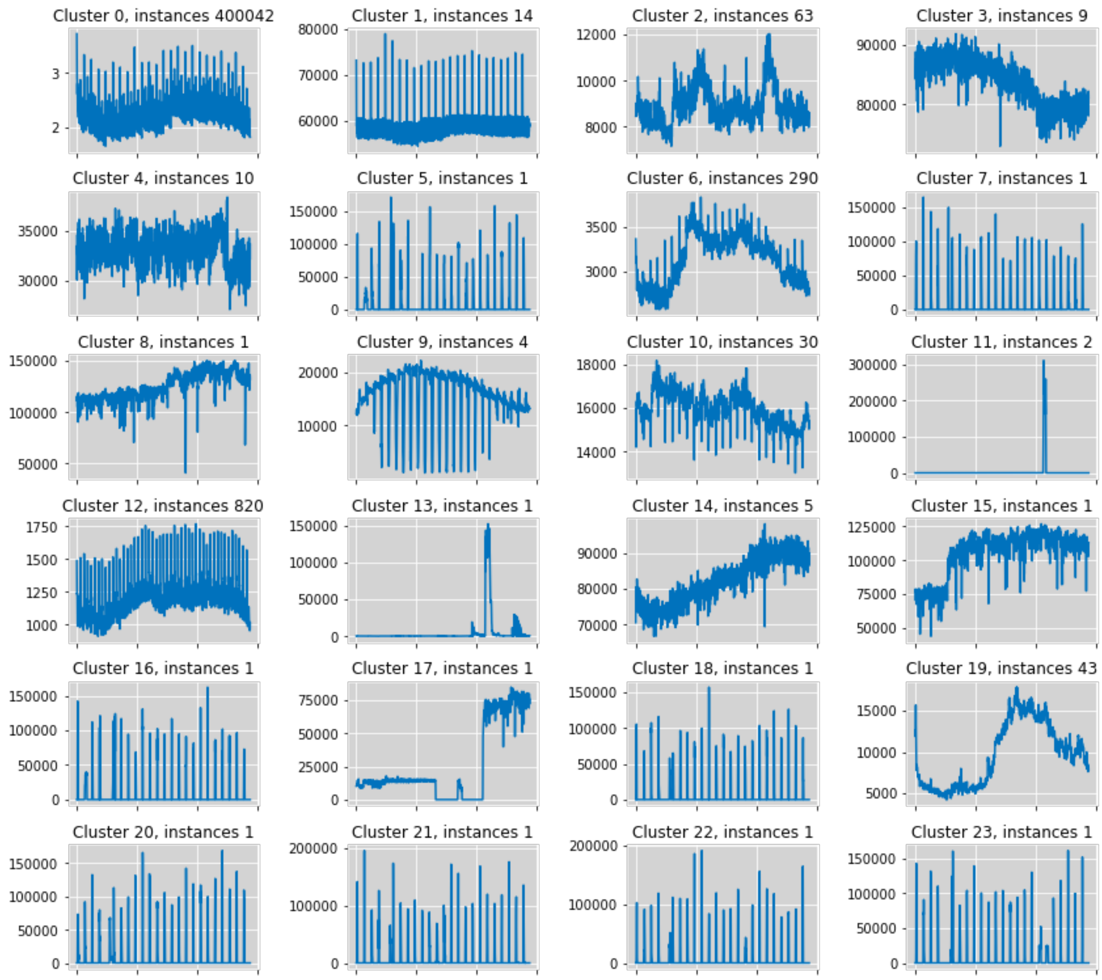
CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG



Serverless Workload Characteristics

A Comprehensive Analysis of Function Invocation Traces Using Clustering Techniques

Master's thesis in Software engineering and technology

Ilja Pavlov & Viktor Vestlund

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Master's thesis 2022

Serverless Workload Characteristics

A Comprehensive Analysis of Function Invocation Traces Using Clustering Techniques

Ilja Pavlov
Viktor Vestlund

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Serverless Workload Characteristics
A Comprehensive Analysis of Function Invocation Traces Using Clustering Techniques
Ilja Pavlov
Viktor Vestlund

© Ilja Pavlov, 2022.
© Viktor Vestlund, 2022.

Supervisor: Joel Scheuner, Department of Computer Science and Engineering
Examiner: Gregory Gay, Department of Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualization of the cluster centroids derived from the time series of the Serverless invocation function traces.

Typeset in LaTeX
Gothenburg, Sweden 2022

Serverless Workload Characteristics
A Comprehensive Analysis of Function Invocation Traces Using Clustering Techniques
Ilja Pavlov & Viktor Vestlund
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Serverless computing and Function-as-a-Service (FaaS) have seen a steady rise, and with more usage, the higher the demand for quick and reliable services becomes. FaaS delivers stress-free scalability and is only billed per execution time for each deployed application for the customer, while the providers retain control over the infrastructure. The objective of this paper focuses on analysing workloads in serverless computing environments with the help of machine learning, to analyse if existing machine learning techniques can be implemented for time-series, or if problems can be overcome. Using machine learning and validation methods, the workloads were analysed, clustered, and characterized in order to identify patterns and offer a way for future improvements to be applied to already existing techniques used by service providers.
The approach taken in this paper is to implement a baseline clustering technique, followed by multiple others to ensure valid results. The clusters are sampled and visualized for characterization, and then grouped based on similar patterns. Multiple common workload patterns are identified across the clusters produced by the defined techniques, which included but are not limited to: Constant, Mean adjacent, Pulse, Sinusoidal, and Spike patterns. Concluding these findings is a discussion about the validity of these common patterns, and suggestions for improvements on these techniques and future studies.

Keywords: Serverless computing, function triggers, clustering, time series, patterns, unsupervised learning, workload characteristics.

Acknowledgements

We would like to express a special thank you to our supervisor Joel Scheuner for all the time, vital expertise, and continuous support that he provided during the thesis. Our deepest gratitude goes to our dearest parents and friends for their invaluable feedback, boundless patience and moral support throughout this journey.

Ilja Pavlov, Viktor Vestlund, Gothenburg, October 2022

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

DBA	DTW Barycenter Averaging
DTW	Dynamic Time Warping
ED	Euclidean Distance
FaaS	Function as a Service
PAM	Partition Around Medoids
PCA	Principal Component Analysis
SOM	Self Organizing Map
OS	Operating System
VM	Virtual Machine

Contents

List of Figures

List of Tables

1 Introduction

Serverless computing is a relatively new subset of cloud computing which has seen a steady rise in prominence over the last decade [1]. While the definition of 'Serverless' has undergone several permutations in meanings over the last couple of years, the recent terminology is often associated with the Function-as-a-Service (FaaS) business model [2]. FaaS platforms enable users to develop, test, and host applications in the Cloud without the need to manage the hardware and software components of the platform itself [3].

FaaS focuses on providing on-demand environments, with deployed products loaded and executed only when their usage is requested, without a need to actively run applications longer than is needed. As a result, FaaS offers an unprecedented level of convenience for all involved parties. The customers benefit from the worry-free scalability and being billed only for the time necessary to execute the deployed applications, while service providers retain a high degree of control over their infrastructure, enabling ways to optimize exactly how the hardware and software resources should be allocated to cover the operation demands of the deployed products.

Yet even as the popularity of FaaS continues to grow, multiple open challenges continue to persist within the field [2], [4], [5], turning the continuous operation of serverless platforms into a balancing act of compromises. The service providers must ensure that the environment's responsiveness to external demands remains quick and constant, maximizing the utilized resources to their fullest potential to bring quality to existing customers, while keeping the operational costs to a minimum. One challenge that illustrates this is the problem of cold starts [6], [7]. How long and how many application instances should be kept 'warm' with pre-allocated resources in order to improve the responsiveness at the cost of wasting resources on 'idling' processes? Or on the contrary, how long and how many applications should be kept in an inactive state so that the more pertinent applications could benefit from freed up resources, at the risk of making the former sluggish and less responsive if the demand for them suddenly spikes? As such, it becomes important to understand how various applications perform within the same environments, how different demands affect the performance of coexisting functions, how external demands affect the serverless ecosystem, and how exactly all mentioned aspects influence one another. Understanding these aspects sets the foundation for novel and efficient solutions for these existing challenges.

1.1 Purpose

The purpose of the study is to provide an in-depth analysis of workloads that exist in serverless computing environments by investigating recurrent function execution patterns with the help of machine learning. The study assesses the feasibility of identifying such patterns in available real-world invocation traces by trialing a selection of existing models applicable to the problem domain. The consequent benchmarking and evaluation of the algorithms and the collected results was synthesized with the descriptive statistical analysis of the data set, and the workload characterizations provided in the previous study, in order to determine identified patterns, evaluate their uniqueness in relation to existing studies, and describe the key "building-block" characteristics that differentiate the types of workloads.

The goal was accomplished through an investigation about existing machine learning algorithms that were applicable to the problem domain. From this a baseline algorithm, followed by candidate algorithms, was implemented and tested. The algorithms was empirically assessed in an experimental setting and evaluated for validity within the context of real-world data. Finally, the results complement the analysis of the identified patterns and the data as a whole, as well as findings from related work, with the goal of formalizing a workload categorization system applicable to the Function-as-a-Service (FaaS) serverless environments.

This study seek to expand the understanding of existing serverless workflow patterns, as well as evaluate the practicality of deriving such insights directly from serverless function invocation traces. Furthermore, investigating the efficacy of machine learning techniques with an analytical focus in the serverless problem domain, as well as documenting the process, may offer novel ways for developing future improvements to the existing techniques employed by serverless platforms.

1.2 Research questions

The thesis intends to provide an answer to three primary research questions listed below.

RQ1. How can existing clustering techniques be applied to Serverless workload pattern identification?

The goal of this research question is to analyze and understand serverless workload patterns, find and deal with any biases and quirks in the data in order to find a suitable set of clustering techniques suitable for the problem domain. Given the nature of the data set consisting of timed recordings, the study, in particular, aims to draw a connection between the time series data and the known algorithms applicable for deriving patterns from such data. Hence, in order to answer this question, the thesis intends to investigate the baseline problem domain and summarize specifications for how the time series focused clustering techniques can be adapted to consume serverless invocation traces and produce the intended results.

RQ2. How can the applicability of workload pattern identification techniques be empirically assessed?

The confidence of findings suffers when no constructive explanation can be given for why discovered results are valid and faithfully represent the problem domain. As such, this research question intends to touch upon the results produced by the clustering techniques with the aim of evaluating and validating the findings and, in case of identified issues, fine-tune and iterate upon the artefacts in order to ensure the applicability of the results in the context of the available, real-world data.

RQ3. What workload characterizations can be derived from the serverless function invocation traces?

The goal of the study is to investigate the invocation traces and describe what findings and elusive patterns can be derived from the breadth of the available data. From the identification of anomalies and inter-dependencies, to correlations between recurrent invocation patterns and function types, the aim is to synthesize the insights via systematic descriptive analysis of the results and come up with qualitative characterizations of the workloads. Given the broad area of investigation that the question covers, the specific focus that the study intends to take can be further summarized as two sub questions:

- **RQ3.1**: How can workload patterns be characterized based on serverless function invocation traces?
- **RQ3.2**: What features and factors differentiate workload patterns based on the serverless function invocation traces?

The listed sub questions define the scope of the investigation, specifically focusing on the summation of observations

1.3 Study limitations and delimitations

Due to the limited time frame of the thesis and the size complexity of the used data set, it is not possible to extensively investigate all potential machine learning techniques applicable to the problem domain, across the entirety of the available data and Azure Function triggers. The computational and time requirements for performing such an investigation falls outside the boundaries of the available time frame, hence limiting the study to a small selection of clustering algorithms. Additionally, given the number of data points involved, investigating patterns across all of the possible intervals of the data set's timeline are too time consuming within the scope of the thesis. Therefore, the overall workload investigation is limited to three scopes: exploratory data analysis on the original data set, and two smaller scopes (1 day and 1 hour) subsets used in pattern identification.
Lastly, while the clustering framework used in the experiment is automated to replicate the end results, the extensive memory utilization during the tests and hardware limitations of the potential testing environments made parallel execution of the test

cases impractical to implement at this stage. As such, replicating the experiment from start to finish can take up to a week of continuous running, depending on the hardware specifications of the testing environments. As such, the replication package[8] includes Jupyter [9] notebooks which incorporate each individual module of the benchmarking framework, permitting the analysis and execution of individual test cases and routines without a need to invoke the entire framework suite.

1.4 Outline

The thesis is structured as follows. Chapter 2 describes the fundamentals required to contextualize the problem domain, covering the topic of workloads in cloud environment, serverless computing, common metrics and triggers and function traces. Workload characterizations that exist within the domain at the time of the study. Furthermore, the time series and the different machine learning techniques are touched upon from the perspective of discovery of patterns and trends within the data. This is followed by chapter 3 that overviews the theoretical fundamentals for clustering time series. Chapter 4 covers the methodology used in this thesis, followed by chapter 5 which presents the results. This is followed by a discussion about the results, in chapter 6. Lastly, the study is summarized and ideas for future work are provided in chapter 7.

2
Background

This chapter describes the background knowledge necessary for establishing a broad understanding of the problem domain pertaining to the study. First, section 2.1 delves into the industrial context of the problem domain, highlighting the evolution of cloud computing technologies and their relation to the workload characteristics sought out by this study. Then, section 2.2 expands further upon the key principles of serverless computing and how general workloads are defined within the domain. Next, section 2.3 describes the common metrics, labels, and triggers that characterize the workloads in serverless environments. Furthermore, section 2.4 presents studies that used function traces to characterize the workloads both in the serverless and general web domains. Lastly, Section 2.5 goes over the studies related to the problem domain as well as techniques used for pattern and trend identification in data, with a particular focus on the time series data mining domain.

2.1 Workloads in cloud environments

Serverless can be originated back to peer-to-peer (P2P) software solutions, however in the cloud domain serverless computing was introduced in an AWS Event in 2014 [1]. Serverless followed naturally after advancements and inventions in the container and virtual machine (VM) domain, which were used to abstract different system layers in order to provide an easier way for managing them by the providers. The abstraction also led to fewer costs and consumption, development speed and ease of deployment.

Serverless can be split into multiple service models with different abstraction layers, where developers have different levels control over the low-level architecture. The four service models that will be discussed are FaaS, as well as Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS).

IaaS is one of the services where the developers have more control of the life time of the service. The provider still manages the hardware, networks and such that are close to the providers, where the developer have more control of the data, the applications, the runtime, and the operating system. Common workloads that use IaaS are: Testing and developments environments, Web applications, and High-Performance Computing [10].

PaaS is the middle part of services, where developers only have control over the apps or applications that are hosted on the cloud and its data. All management of VM, OS and servers are handled by the providers. This is a simpler way of having applications on the cloud, however, one gets less control of the service and only

the development, management and analytic tools the provider provides. Common workload used with PaaS are: A development framework for developers, such as DevOps platforms, analytics or business intelligence, and other additional services for applications [11].
FaaS is the service where one only controls the application(s) that are on the cloud. The applications are functions, and the provider manages everything else, such as scaling and the underlying infrastructure. FaaS provides simpler scaling and less managing, which leads to a more attractive cost model where only the used time is billed. Compare the aforementioned to PaaS, where idle time is billed for and scaling is self-managed [1]. Common FaaS workloads are: Internet-of-Things (IoT) applications, and microservices.
SaaS is a service where everything is managed by the provider, and where the user uses the software over the internet. The software provider manages everything, and the consumers usually pay-as-you-go [12]. Common SaaS workloads are: Web-based email services, Slack, or similar applications. [13]
Additionally, different types of cloud environments can be categorized through the ownership factor, ranging from public clouds offered by different service providers, to privately managed cloud environments that can be organized for internal use without an external user-base, using open source frameworks such as OpenStack.
This study focuses on serverless workloads that originate from Microsoft Azure (specifically, Azure Functions), with the platform supporting multiple services ranging from IaaS to FaaS. Other known providers that offer FaaS solutions are: Amazon Web Services Lambda (AWS), Google Cloud Functions, IBM Cloud Functions, and Cloudflare Cloud Functions.

2.2 Serverless computing

The term serverless computing can be understood by the term itself. It can be defined as a service where developers do not have to worry about the servers and their setup during development. The management and scaling is done by the provider and the service is payed for per request or event. Castro *et al.* defines serverless computing as "[...] a platform that hides server usage from developers and runs code on-demand automatically scaled and billed only for the time the code is running." [1].
This definition encapsulates the most important parts of serverless computing; cost and elasticity. Cost can be considered in regards to only being charged for what you run, in other words, pay-as-you-go. This model comes naturally due to the nature of serverless computing, where developers not have control over the servers, and thus only pay for what is run and not for when the servers are idle (or other types of overhead time). Elasticity refers to scaling on the servers, and due to the developers having no control over them, all decisions regarding servers are up to the providers. The providers are trusted that they will be able to cover all usage the developers want, with good performance, scalability, and maintainability of the servers [1].
Serverless computing is typically used for smaller units of computations, which are easier to manage and scale in the cloud. One of the more popular methods to use serverless computing is to provide a function to be executed, which introduced the service many providers provide today - FaaS. The difference between the two is

minuscule and they are sometimes used interchangeably, however, depending on how you define it, serverless computing could be further developed with more additions (such as state management) which FaaS does not have [1].
FaaS and serverless computing provide a lot of useful opportunities where large companies use serverless computing for their operations. The Coca-Cola Company uses serverless for their IT operations to reduce costs and deploy services faster, such as in their vending machines and their loyalty program. Other large companies using serverless are Expedia and Santander Bank [1]. These examples show that multiple companies benefit from using serverless computing, however, there are downsides to all technology.
Hellerstein *et al.* [5] mentions four different problems with serverless today, with a particular focus on the AWS Lambda. The first problem discussed is that function invocations have limited lifetimes, which boils down to a problem with keeping state management. The second problem is I/O bottlenecks (mainly network speeds), and when the functions are on a single VM the resources are shared, and thus slow everything down more. The third is communication through slow storage. This is most notable when trying to store state between multiple functions and it has to be read and written to slow storage for every call. The fourth and last problem is that there exists no specialised hardware that is optimized with serverless functions in mind, which forces both the users and providers to apply various resources allocation policies (such as pre-allocating CPU or RAM) in order to effectively manage the access to shared cloud resources. It should be noted that while the study focused on Amazon Web Services ecosystem, the problems are generalizable across other public providers as according to Figiela *et al.* [14] serverless functions perform in a nearly similar manner and consistent overheads within the environments, while possessing some differences in performance. Furthermore, the general use of the heterogeneous hardware in FaaS infrastructure has been explicitly confirmed for AWS Lambda and Google Cloud Functions (GCP). The problem of generalizability and differences in the infrastructure between the providers is important given the dependency of the given thesis on the data collected exclusively from the Azure Functions ecosystems, with further details provided in Section 2.4.

2.3 Common serverless metrics and triggers

Workloads contains information that both the user and the provider use which shares commonalities between the different providers. Common triggers that providers use are HTTP, which are any requests or responses over the protocol. Storage and database which are triggers to send requests to a service that stores data. Queues and timers which sends invocations either in a queue or in a defined manner. The HTTP trigger being the most commonly studied trigger type [15]. Common metrics that the different providers supports are: duration of the initialization of the function and the duration of the invocation. The number of cold starts, which is when the function is pre-loaded into memory, errors, executions, and time-outs. Which all matter for understanding how your functions are being executed. CPU and memory usage are also metrics that are frequently used and supported by multiple providers [16].

2.4 Workload characterization

Workload characterization has been a broad topic of discussion and study within the cloud computing domain, relative to similar studies within the serverless FaaS environments. Shahrad *et al.* [17] are the pioneers in characterizing serverless workloads at Microsoft Azure, and their released subset of traces is the data set used in this thesis. Their focus was to characterize FaaS workloads which are intrinsic to the applications and functions, in order to gain an understanding about their implications for cold starts and resource management. Their results show that the vast majority of applications are invoked infrequently, and that a fifth of the applications are responsible for almost all invocations.

Eismann *et al.* [18] conducted a study where they collected, characterized and analysed descriptions from 89 serverless applications from open-source projects, academic literature and industrial literature. Using these, 16 characteristics were analysed, the characteristics help to understand when and why serverless applications are successful and how they were built. These results were then compared to other studies and data sets to find a consensus and to find disagreements. These characteristics discuss about which provider is the most popular, the most popular programming language, the number of functions per application, which trigger type(s), how "bursty" (i.e. sharp spikes in activity or resource utilization within a short period of time) the workload is, the application type, the function run-time, and the motivation for why to use Serverless applications. This study builds on the classifications made, using one of the data sets [19] that were studied, and this thesis further describes the different functions in the serverless domain, by classifying different patterns that arise from the invocations.

Calzarossa *et al.* [20] talks about different types of workloads characteristics and how to characterize workloads. The authors present that the most common approach for workload characterization is an experimental approach, which is based on the analysis of measurements taken and collected from the workload. Similarly to what is being made in this thesis, descriptive statistics of parameters from the workload are said to be able to provide preliminary insights. Furthermore, clustering is also mentioned by the authors which is used to create groups and further describe the workload. The combination of clustering and descriptive statistics have been widely applied to workload characterization, which is one reason why this thesis uses the same structure for analyzing and characterizing. However, mostly static characteristics are accounted for in the previous studies, while this study will focus on characterizing the workloads themselves.

The same authors revisits and further complement their study [20] done 16 years prior in [21]. They discuss the current events in workload characterization using the same methodology to collect and analyse the data. A recurrent problem discussed for these kinds of studies is that gathering data for workload characterization is difficult, due to the lack of publicly available data. However, the authors conclude that the same methodology used in the previous study still is applicable in the present to characterize workloads. The same methodology works for workloads from multiple fields in the web domain, ranging from online social networks, video services and mobile apps, to cloud computing.

The previous research shows clearly that the method used in this thesis is the most popular way of analysing and characterising workloads, and further testing the method on time series data from a Serverless FaaS domain will show the method is possible for this kind of data. Furthermore, it can help gain an understanding of the different patterns that occur in a serverless environment, with the difficulties and problems that occur on the way. The limitation discussed in [21] is not applicable due to the open source nature of the sampled data from [19], however, as mentioned it is just a sample from the real-world data that was gathered. Since Shahrad *et al.* [17] are the pioneers and owners of the data set used in this thesis, this is a step forward in their work by further characterizing just the invocation patterns themselves, and not the entire production FaaS workload of Azure Functions.

2.5 Time-series and related work

While Serverless ecosystems possess differences relative to the more classical environments, the fundamentals behind the executed processes remain similar to the regular computer programs. Serverless functions are processes that are executed over a period of time while consuming a certain amount of allocated hardware resources . The data that describes how such processes perform over time fits the format and definition of *time-series*. The serverless function traces provided by Shahrad *et al.* [17] are used in this thesis as principal data, making working with time-series analysis central to the study. Time-series analysis has been studied as early as 1961 [22] and is used extensively with data that falls this format, e.g. data collected within the fields of public health and biomedical [23], social sciences [24], economy [25], and geophysics [26].
Using time-series for pattern identification with different techniques has been widely studied, with multiple studies [27], [28], [29] summarizing the common methods, techniques, and limitations when it comes to dealing with the time-series data. Gentle and Wilson [27] focused on finding patterns in time-series, using the *Alternating Trends Smoothing* (ATS) algorithm. The focus of the study centered on finding a way to cluster or classify the time-series to better predict the future outcomes. Furthermore, Gentle and Wilson discussed the challenge of overfitting that often occur when trying to use a clustering or classification model on large data sets of time-series data.
Wang *et al.* [28] focused on clustering time-series data, using the structural characteristics of the data set itself, rather than the trends or change points like [27]. With this method, the authors have shown that they could achieve high accuracy, even with few available characteristics. The demonstrated method comes with an additional benefit of being able to reduce the dimensionality of the time-series, and a significantly lower sensitivity to the missing and noisy data.
Alqahtani *et al.* [29] expanded upon the possibilities of clustering time-series with the focus on *deep time-series clustering* (DTSC). The method for clustering used a *deep convolutional auto-encoder* (DCAE) which clusters and reconstructs objective functions. The three main techniques used and compared by the authors were *K-means*, *deep auto-encoder* (DAE), and *1D-convolutional layer for deep convolutional auto-encoder* (1D-DCAE). The results show that 1D-DCAE completely outperforms

the k-means clustering method, and scores better than DAE.
In summary, time-series analysis has been widely applied to identify patterns in the time-series data, with application of different scopes, measures, and techniques in order to group and characterize the time-series differently. The data used in the thesis was collected over a predefined timeline from a single data-center [17], reducing the difficulty of performing the study by removing the need for synchronization of observations. Furthermore, while k-means clustering has been widely used and discussed by all authors as a fair baseline to perform the analysis, Alqahtani *et al.* [29] suggests that there may exist better solutions. In the context of the thesis, the application of k-means and k-medoids clustering algorithms are explored using euclidean distance (ED) and dynamic time warping (DTW) similarity measures, as well as several multivariate time-series clustering approaches described by Alqahtani *et al.* [29]. Lastly, at the time of study, no related literature could be identified covering the application of clustering techniques for characterization of the serverless workloads, representing the gap in knowledge and the focus of the given thesis.

3
Time-series analysis

This chapter serves as an introduction to the theoretical background behind the time-series analysis. The aim is to establish the problem domain, highlight the peculiarities of time-series as a data artefact, and present both the machine learning and validation techniques that were used in the exploratory data analysis performed by this study. The choice of the algorithms presented in this chapter is based on the relevance and popularity of said methods in the field, as well as considerations for constraints posed by the spacial complexity of the principal data.
The chapter is organized in several sections, each covering a fundamental principle or technique used in the study. Section 3.1 introduces both the general definition of clustering, as well as clustering as a subset of time-series analysis. Next, Section 3.2 presents the core challenges within time-series analysis, followed by Section 3.3 which establishes the relevant terms, techniques, and approaches within the domain, including the common similarity measures used within the clustering tasks. Lastly, Section 3.4 describes the algorithms that were chosen for the study, followed by Section 3.5 which presents metrics used for validating the quality of produced clusters, specifically the internal evaluation metrics that do not depend on the a priori domain knowledge of the used data.

3.1 Time-series clustering

The literature in machine learning domain defines clustering as a data mining technique which allows for categorizing the data without prior knowledge of the basis for the categories [30], most commonly achieved through an algorithmic set of computations. Clustering is considered to be one of the most popular approaches within the exploratory data analysis, as it enables the discovery of the structures, patterns, and relationships between different samples within the unlabeled data. The key principle in clustering centers around the systematic evaluation of homogeneity across the analyzed data artefacts using various similarity (or dissimilarity) measuring defined methods. Once a set of similarity scores is produced, the closest homogeneous data objects are grouped and labeled as belonging to the same "cluster".
Time-series clustering is a subset of clustering which focuses on the data arranged in an ordered, temporal sequence of observations that are commonly delimited by even time intervals, where observations can be univariate or multivariate. Mathematically, time-series T with n observations can be defined as:

$$T = (t_1, t_2, t_3, \dots, t_n), t_n \in \mathbb{R} \tag{3.1}$$

The univariate time-series are series that consist of single time-dependent observations, while multivariate time-series can contain multiple types of such observations. For example, a timeline of price fluctuations for a currency represents univariate time-series, as each time unit of the timeline maps exactly to one metric. Conversely, a weather station that simultaneously collects information from different types of sensors can generate multivariate time-series, if the collected data is synchronized according to the same timeline.
Time-series can be further split into different smaller subsets called sub-sequences. The sub-sequence S is a sequence of length $m \leq n$ consisting of values from the original time-series T.

$$S = (t_k, t_{k+1}, \ldots, t_{k+m-1}), \text{ where } k \text{ is } 1 \leq k \leq n - m + 1. \tag{3.2}$$

Formally, the process of time-series clustering can be defined as the following; Given a data set of n time-series data $D = F_1, F_2, ...F_n$, the process of unsupervised partitioning of D into clusters $C = C_1, C_2, ..., C_k$, in a way that congruent time-series are grouped together based on a selected similarity measure, is called time-series clustering [30].

3.2 Challenges in the time-series analysis

When it comes to applying clustering techniques to time-series, the literature identifies two major forms of challenges [30].
The first challenge is linked to the size and dimensionality of the time-series data [31]. The variable scope in observations, especially in cases of multivariate time-series or a collection of grouped univariate time-series, increases the total dimensionality of data sets. With the measurement of similarity between the data points being the main principle in clustering, the highly dimensional data increases computational demands at an exponential rate [32]. Consequently, the demands on the hardware resource management, and the application of task parallelization techniques becomes paramount in order to cluster large data sets within a reasonable amount of time. This problem can be further exacerbated in situations when time-series data requires more space than the memory available on computational platforms. As similarity scores need to be tracked alongside the time-series data points, the need for chunking and buffering data between memory and persistent disk storage can lead to exponential decreases in the speed of the process.
The second notable challenge is linked with the identification of similarities within or between the clustered time-series. Time-series are described as being naturally noisy and filled with shifts and outliers [33]. This, alongside the potential differences in the length of the target time-series, complicates the process further, as the similarity measure between the time-series needs to be matched and computed across the whole length of the target sequences. Such a misalignment can lead to erroneous separation of data objects, with clusters computationally matching the used similarity criteria, yet not holding to the ground truth under critical examination. As such, time-series clustering needs to be approached with care and consideration for the target data and particular techniques to be applied to such data.

Thus, clustering time-series generally consists of two primary ways of approaching challenges [34]. The first way maximizes the data details by processing raw data via adaptive similarity measures. As highlighted by the aforementioned challenges, such a step is considered time and resource consuming, and depending on the chosen similarity, the measures can be influenced by the noisiness of the data. Alternatively, the second way attempts to reduce computational and time requirements via dimensionality reduction techniques. Instead of using the raw data, the time-series is pre-processed in order to extract an aggregated set of features which uniquely describe different attributes of the time-series data. The reduction of dimensionality improves the efficiency of the clustering processes, especially in the cases of large data sets, but comes at a cost of lost details due to obfuscation of the real data via the modeled representations.

3.3 Time-series clustering techniques

3.3.1 Common approaches

The time-series analysis domain contains several methods to clustering that are defined by the intrinsic characteristics of time series data. The comprehensive review of the time-series clustering techniques by Aghabozorgi *et al.* [30] identified three prevalent directions within the field, which are further visualized in Figure 3.1.

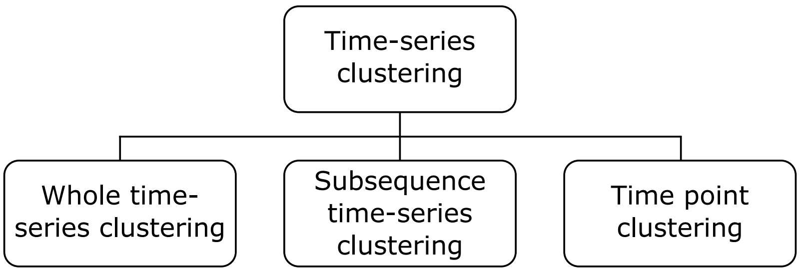


Figure 3.1: Overview of the time-series clustering approaches as described by Aghabozorgi *et al.* [30]

The first approach is known as *whole time-series clustering* and is considered to be one of the most prevalent methods in clustering time-series. The method treats individual time-series as discrete, self-contained objects, with clustering establishing similarity or dissimilarity using whole time-series, rather than parts of them.
The second prevalent method is called *subsequence time-series clustering.* Unlike the previous approach, this method is commonly used for clustering single, long time-series. The target time-series is divided into multiple sub-sequences via splicing techniques such as sliding windows [30], [35], with clustering performed between the subsets of such sub-sequences. It should be noted that Keogh and Lin [36] brought arguments against the usage of sliding windows, as the mean between all sub-sequences results in a constant vector, essentially rendering the clusters produced through that approach as random.
The last approach is known as *time point clustering.* Similar to sub-sequence clustering, this method commonly targets the sub-sequences within the individual time-

series. The clustering is based on establishing similarity between the individual data points, with similarity/dissimilarity measures derived from the proximity between neighboring time points.
Out of the three approaches, the whole time-series clustering aligns closely with objectives of this study, as the identification of recurrent patterns targets self-contained function invocations.

3.3.2 Whole time-series clustering

When it comes to application of the whole time-series clustering, the literature further specifies four primary approaches used in the field [30] and are illustrated in the Figure 3.2.

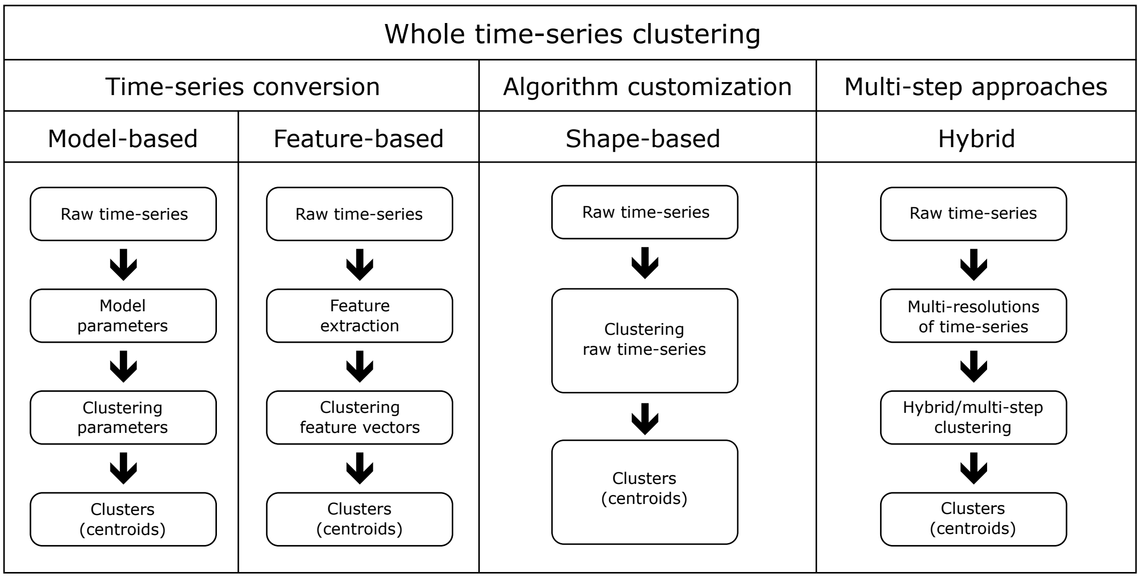


Figure 3.2: Overview of the methods used in the whole time-series clustering as described by Aghabozorgi *et al.* [30]

The *time-series conversion* methodology attempts to address the problem of the time-series dimensionality through the means of summarizing the original raw-data with less descriptors, which is achieved through application of *model-based* or *feature-based* approaches.
The *model-based method* uses statistical and/or mathematical models (e.g. Kohonen self organizing maps [37], ARIMA [38], Hidden Markov models [39], etc.) as an intermediary layer between the raw time-series data and the clustering process. Instead of clustering the time-series data directly, the data is processed through the chosen model first, producing parameters that are uniquely specific to the selected model. Then, the produced parameters are grouped into sets that are used as input vectors for the clustering techniques. Finally, the clusters and parameters that were assigned to them are traced back to the original data, resulting in an overview of the clustered time-series.
The *feature-based method* applies a similar dimensionality reduction idea, with a focus on producing descriptors that statistically describe the time-series. The raw

data is analyzed in order to produce a set of statistical features, such as the mean, variance, minima, maxima, etc. Then, similar to the model-based approach, the produced features are grouped into input vectors that are processed through clustering techniques. While this method offers a great potential for reducing the dimensions and some of the noisiness of the clustered data, the process naturally reduces the descriptive potential of the clustered input relative to the original data. As such, this method heavily depends on the careful feature selection and evaluation of their descriptive qualities before clustering.

Unlike the time-series conversion methods, the *shape-based method* applies the raw data directly, comparing the time-series through computational, algorithmic means. This is commonly achieved through distance-based measures, such as Minkowski distances or dynamic time warping, that compute similarity or dissimilarity based on the closeness of time-series and their constituting data points. While this method offers full descriptive clarity of the original data, this comes as a steep computational costs as described in Section 3.2, as each data point across all time-series needs to be compared with one another.

The *hybrid-approach methods* attempt to mix-and-match the previously described methodologies, using the shape-based, feature-based, and/or model-based methods together in different combinations. The resulting multi-resolution data produced through a chain of methods and processing techniques is clustered and traced back to the original time-series.

3.3.3 Feature extraction methods

The literature indicates that feature extraction can be implemented using different approaches and methods [40]–[42], however generally, the overall process can be structured into four conceptual phases as visualized in Figure 3.3.

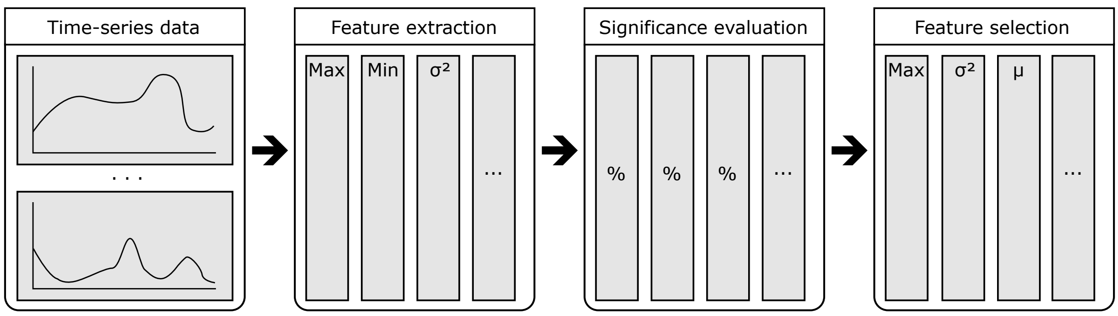


Figure 3.3: Overview of the main phases of feature extraction and selection process.

First, the time-series data is prepared in a format compatible with statistical methods used for feature extraction. Next, each series is examined through the lens of the chosen techniques, with statistical features aggregated into a new set of vectors per each time-series. Finally, in order to ensure that the features used for clustering possess a good descriptive potential of the original data, each feature needs to examined via significance evaluation techniques. Feature selection is used to determine features with the most significant descriptive potential, while filtering out the minor features that may detrimentally affect the clustering results - be it through the introduction of noise or an increase in the computational and/or memory demands.

3.3.4 Principal component analysis

Principal component analysis (PCA) is an adaptive data analysis technique used for reduction of the dimensionality in large datasets while explaining the principal variance-covariance structure of the data through computation of linear combinations [43]–[45]. Principal components represent the original variables in the data, linearly, and in order by maximal sample variance along all combinations of variables. The following components represent linear combinations that were not explained by the preceding component, and all components with the exception to the first one must be orthogonal to the last component [44].
Suppose X is a set of p features composed of n observations where $X = \{X_1, X_2, ...X_p\}$. The key idea behind PCA is that each observation that exists within the p-dimensional space of X has varying descriptive value between different dimensions. Thus, PCA seeks to assess the value of said dimensions using variance [45]. As such, the first principal component of X is a vector Z_1 where factors ϕ adhere to the normalized state $\sum_{j=1}^{p} = \phi_{j1}^2 = 1$. The linear combination of the constituting features in Z_1 is:

$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + ... + \phi_{p1}X_p. \tag{3.3}$$

Factors ϕ represent the loadings of the principal component, with loadings of Z_1 constituting the first principal component's loading vector $\phi_1 = (\phi_{11}\phi_{21}...\phi_{p1})^T$. The loadings are constrained by the sum of squares of 1 as to limit the maximum ceiling of the sought variance. With X composed of $n \times p$ observations, linear combinations between sample feature values are examined in a manner of

$$Z_{i1} = \phi_{11}X_{i1} + \phi_{21}X_{i2} + ... + \phi_{p1}X_{ip} \tag{3.4}$$

in order to discover a combination with the largest sample variance subjected to the same normalization constraint as Z_1. As such, the purpose of the loading vector of the principal component is to solve the optimization problem,

$$\underset{\phi_{11},\dots,\phi_{p1}}{\text{maximize}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} \phi_{j1} x_{ij} \right)^2 \right\} \text{ subject to } \sum_{j=1}^{p} \phi_{j1}^2 = 1 \tag{3.5}$$

through eigenvector decomposition, maximizing the sample variance of n observations of z_{i1}, where $z_{i1}, ..., z_{n1}$ represent the scores of the first principal component.
Once the first principal component Z_1 is identified, the following component Z_2 is determined through a process of finding the maximal variance out of all linear combinations of features in X that are uncorrelated with Z_1. This sequence is further repeated through the number of desired principal components.
In the end, PCA produces a set of components that are organized by their level of explained variance. If the data used in PCA has a strong underlining structure of dependencies between the constituting variables, the degree of said significance would be expressed in the variance of the first components in the vector. Assuming that this is the case and the number of the components with significant variance is

less than p, the subset of these components can be clustered as features - a method that was applied by this thesis which is further explained in Section 4.3.5.

3.3.5 Similarity measures

3.3.5.1 Euclidean distance

The commonly used distance based similarity measures in time-series clustering are derived from the Minkowski distance [46],[47]. The Eq. 3.6 showcases the general way of finding a distance between two vector sets of samples, where Minkowski p-metric defines the dimensionality of the given vectors. For two N sized time-series $x(t) = (x_v)$ and $y = (y_v)$, where $x_v = \{x(1), x(2), ..., x(N)\}$ and $y_v = \{y(1), y(2), ..., y(N)\}$, the following Eq. 3.6 determines the distance.

$$d_p(x, y) = \left(\sum_{v=1}^{M} |x_v - y_v|^p \right)^{1/p} \tag{3.6}$$

In cases of $p = 1$ (i.e. one-dimensional vectors), the metric is reduced to the Manhattan distance, while $p = 2$ represents the Euclidean distances. Thus, the same equation for Euclidean distance can be re-defined as illustrated in Eq. 3.7

$$d_{ED}(x, y) = \sqrt{\sum_{\mathrm{v}=1}^{\mathrm{M}} |\mathrm{x_v} - \mathrm{y_v}|^2} \tag{3.7}$$

The linear computational time complexity makes Euclidean distance a popular choice as a similarity measure within the broader domain of cluster analysis. When it comes to time-series analysis, however, the measure's 'lock-step' nature presents several limitations, such as an inability to handle sample sets with different dimensions, data sampling rates, phase-shifts between time axis, and noise, which are common in time-series data. As such, the direct application of it on raw samples without data pre-processing, such as de-noising, normalization, and phase-realignment, often yields skewed results.

3.3.5.2 Dynamic time warping

The dynamic time warping (DTW) is a modification of the Euclidean distance measure which permits 'stretching' and 'compression' of differently-shaped sequences in order to improve similarity detection. The algorithm takes the time shifts that may occur into consideration, finding an optimal alignment ratio between two sequences. When applied in conjunction with time-series clustering, DTW commonly results in more representative centroids and clusters when compared to Euclidean distance. The algorithm works by computing a cost matrix for one of the examined sequences and then calculating the minimal cost through the path. In order to apply DTW, the process depends on three conditions. First, the *boundary condition* stipulates that the first and the last points of both time-series must be temporally aligned. The second *monotonicity condition* posits that time-series data points needs to follow each other. If an element in one of the series precedes another one, then this should also hold for the second series. The last *step size condition*, states that no

point can be ignored in the sequences and that repetitions must not be acceptable in the final path [48].
It should be noted that the DTW algorithm cannot be directly applied to the clustering process. Instead, a derivative algorithm based on DTW called *DTW barycenter averaging* (DBA) is used instead [49]. The DBA algorithm is presented in [50] and the aim of the algorithm is "[...] to minimize the sum of the squared DTW distances from the average sequence to the set of sequences." [50]. The algorithm calculates a sum of the Euclidean distances for a coordinate and the coordinates that was computed from the DTW algorithm for the given coordinate. Then, the coordinates of the average sequence is updated using one of more coordinates from each sequence. Then the minimum of the partial sum for each coordinate of the average sequence is calculated, by taking the barycenter of that set of coordinates. The DBA algorithm repeats this set of operations for each coordinate in the average sequence, producing the 'minimized' sequence which is consequently used in clustering.

3.4 Clustering techniques

3.4.1 K-means algorithm

The K-means algorithm, also referred to as Lloyd's algorithm, is considered to be one of the oldest and most utilized methods in clustering [51]. The algorithm aims to reduce the intra-cluster sum of squares by partitioning the n number of samples into k number of clusters via the chosen similarity measure and the mean of cluster samples which represent the center point of the cluster called centroids [34]. The similarity is measured using distance-based metrics, such as Euclidean distance, dynamic time warping, soft dynamic time warping, and alike. The algorithm operates in an iterative manner, by recalculating the similarity between samples and cluster centroids and adjusting their positions until all clusters reach the state of parity, i.e. become static between iterations. The pseudocode for the algorithm is shown in Algorithm 1, and consists of four steps.

Algorithm 1: K-means Clustering Algorithm

```
Input: k number of clusters
       measure used for finding similarity
       max_iter maximum number of iterations
Step 1: Initialise k centroids using seeding heuristic
Step 2: Assign all the points to the closest centroid according to measure
Step 3: Recompute the centroids of the formed clusters
Step 4: Evaluate
if No updated centroids or points OR max_iter then
 |  The algorithm has converged
else
 |  Repeat step 2 and 3
end
```

First, the algorithm seeds centroids of the target number of k clusters, which will be iteratively updated in the following steps. The centroids normally represent the mean of samples belonging to a certain cluster. Given that at this stage the samples have not been assigned to the clusters yet, the centroids are selected using an initialization methods, which range form a predetermined set of centroids, randomly selected centroids, to algorithmic selection based on specialized heuristics, such as k-means++ [52], Maxmin [53], Luxburg [54], and more [55]. The choice of the initialization heuristic depends on the the clustering needs, with predetermined sets useful for ensuring repeatability and reproducibility of clustering rounds, whereas the specialized methods propose ways for improving the total runtime of clustering sessions. Most specialized heuristics optimize the positioning of the initial centroids, as the closer the centroids can be initialized to the optimal centroid positions of the final clusters, the less iterations it will take for the algorithm to converge with them.

Once the initial centroids are initialized, a similarity-measure score is calculated between each sample of the target set and centroids, as well as the within-cluster sum of squares for each cluster. Samples that share highest similarity with the target centroid, i.e. closest centroid, are gradually assigned to respective clusters, until the target set is exhausted and all samples are assigned to one of the clusters.

Next, the algorithm iterates over each cluster and calculates the mean of all samples that were assigned to the cluster in the previous step. The mean is derived mathematically and does not directly represent any discrete sample within the cluster. Rather, the arbitrary variable represents the new potential centroid point of the cluster, which potentially minimizes the within-cluster sum of squares between target samples.

The final step compares the newly calculated centroids of each cluster with their previous counterparts. If no such differences are identified for each centroid pair, the centroids have converged with the existing samples, and the algorithm terminates. Otherwise, the previous sample re-assignment and centroid re-calculation steps are repeated until convergence can be reached. Certain edge-cases may result in centroids not reaching parity, hence most implementations of the k-means algorithm include alternative termination conditions. One of the popular used conditions is the maximum number of permitted iterations, which terminates the algorithm if convergence cannot be reached within the given number of iterations. Furthermore, in the case of randomized initialization heuristics, the results may not always converge with the true optimal partitioning of the given set. In such situations, clustering may be performed several times, and the comparison of the final results made in order to identify the cluster configurations with the minimal within-cluster sum of squared distances across different runs.

The classic K-means algorithm uses crisp sample partitioning [51], which guarantees that each n sample of the target set will be exclusively assigned to exactly one k cluster. Such an assignment may not always be representative of the actual grouping that may exist between samples. Moreover, given that distance measures commonly capture the magnitude of the time-series data points, the calculated centroids become sensitive to outliers, which may then skew the mean of samples.

3.4.2 K-medoids algorithm

The k-medoids is a clustering model that in many aspects resembles the k-means algorithm. Much like k-means, the baseline k-medoids uses distance-based similarity measures between all samples in order to iteratively assign samples to clusters [56]. Similarly, each cluster composition is evaluated and iterated upon in order to find the optimal cluster composition for the given sample set. The key difference lies in the method of selecting the center points of the clusters. While k-means mathematically derives centroids from the sample mean of each cluster, k-medoid selects an actual discrete sample point that belongs to the cluster, called ’medoids’ or ’exemplars’. The pseudocode for the algorithm is shown in Algorithm 2, and consists of five steps. First, the algorithm selects the k number of medoids. Similarly to k-means, this selection can be made based on the predetermined set of medoids, random selection from the set of samples, or specialized heuristics tailored for medoid selection. Then, the similarity is measured for each sample relative to existing medoids, with each sample assigned to the cluster represented by the closest medoid.
The next step introduces the main difference between k-medoids and k-means algorithms. Instead of the mean of samples, the algorithm iterates over each sample of the cluster and measures distances relative to other sample points. Then, all distances are summed up to produce the total within-cluster distance score for each sample point. The sample with the smallest score, i.e. the most centrally located sample within the cluster, is selected as the new medoid. This process is repeated, until a set of new medoids is calculated for each initialized cluster. While this selection technique improves the algorithm’s tolerance towards the outliers and noise, especially when applied to the time-series data, this improvement comes at a steep computational cost. While sample mean calculation in k-means can be performed as a linear $\mathcal{O}(n)$ rate, the comparison between all n samples within cluster subsets increases it in a quadratic manner $\mathcal{O}(n^2)$.
The final, evaluation step bears a resemblance to the k-means algorithm, with each cluster medoid compared with the previous iteration’s counterparts. The algorithm repeats previous steps until parity is reached between the new and previous medoid sets, upon which the process terminates and returns the clustering results.

3.4.3 Kohonen self-organizing maps

A self organising map (SOM) is a type of artificial neural network (ANN), which belongs to the subset of unsupervised machine learning techniques. Artificial neural networks are a model consisting of neurons (nodes) that connect to each other. The nodes represents a specific function with a specific output, usually referred to the activation function. The connections between the nodes is called weights, which carry the signal between the network neurons and can act as the model’s memorization. The output of the network depends entirely of the setup of the network and what the expected use of the network is [57]. What makes SOM differ from other ANNs is the lack of the back-propagation correction steps.
The SOM algorithm was introduced in 1982 [58], and instead of learning from error-correcting, it is trained by competitive learning. In Figure 3.4 an overview over how the SOM-ANN is setup is shown, while Algorithm 3 details the basic training

Algorithm 2: K-medoids clustering algorithm

Input: k number of clusters
measure used for finding similarity
Step 1: Initialise k medoids
Step 2: Assign all the points to the closest medoid according to *measure*
Step 3: For each n sample in k cluster, find sum of *measure* relative to other samples in k cluster
Step 4: Select new k medoids, where sum of *measure* among n samples is lowest
Step 5: Evaluate
if No updated medoids **then**
| The algorithm has converged
else
| Repeat step 2, 3 and 4
end

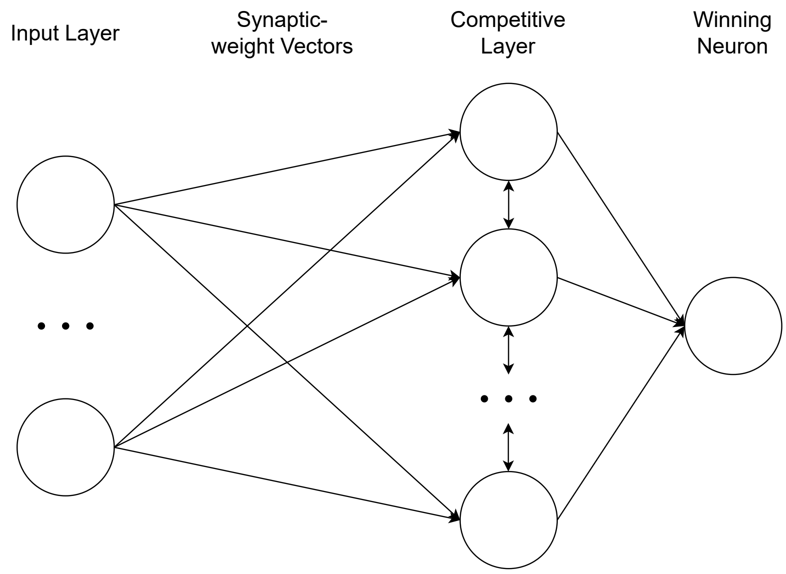


Figure 3.4: Overview of the self organizing map's network structure

sequence of SOM.
Firstly, the algorithm randomizes the synaptic-weights, then randomly selects an input vector from the data set, which is the input layer. Then in the output (Competitive) layer, the algorithm calculates the *best matching unit* (BMU), using the Euclidean distance, which is the neuron closest to the input vector. The competitive layer neurons are competing and are thus connected, which is called negative feedback. The BMU, the *winning neuron* in Figure 3.4, is in this figure a separate node, but in practice the competitive layer only activates one node as the winning one and outputs the result. The radius of the neighbors of the BMU is calculated, and then the BMU and the neighbors weights are updated towards the input vector. The neighbors are updated in inverse proportion of their distance, and furthermore a set learning rate η is used for the updates. The algorithm repeats the aforementioned steps for all vectors in the input data set, with each iteration decreasing the neighborhood radius and the learning rate applied to the node weights, until the map's topology converges [59].

Algorithm 3: Baseline self-organizing map algorithm

```
Input: S set of data vectors
       M grid-space with SOM nodes
       i initialization heuristic
       σ neighborhood radius
       η learning rate
       T number of training epochs
Step 1: Initialize all nodes in M using heuristic i (e.g. random weights)
while t < T do
    foreach sample vector ∈ S do
        Step 2: Find the node that best matches the vector
        Step 3: Adjust the node towards vector by inverse proportion of
          relative distance and η
        Step 4: Determine neighborhood of nodes N within σ radius
          from the node
        foreach neighbor ∈ N do
            Step 5: Adjust neighbor towards vector by inverse
              proportion of relative distance and η
        end
    end
end
```

3.5 Cluster evaluation

3.5.1 Caliński-Harabasz index

The Caliński-Harabasz index (also known as Variance ratio criterion, or VRC) is an internal cluster validation measure introduced by Tadeusz Caliński and Joachim Harabasz as an "[...] informal indicator of the 'best number' of clusters [...]"[60]. The index compares sample similarity relative to the cluster that they were assigned to, i.e. the degree of sample cohesion, relative to the external clusters, i.e. the degree of separation. The index is calculated for every K number of clusters as indicated in Eq. 3.8.

$$VRC(K) = \frac{BGSS(K)}{WGSS(K)} \frac{N-K}{K-1} \tag{3.8}$$

The N variable in Eq. 3.8 represents the total number of samples across all clusters, $BGSS$ represents the aforementioned degree of separation, or between-group (cluster) sum of squares, $WGSS$ represents the degree of cohesion, or within-group (cluster) sum of squares, and K specifies the total number of clusters. For a set S that contains all observations S_i, where $S = \{S_1, S_2, ..., S_N\}$, $BGSS(K)$ and $WGSS(K)$ can be described as,

$$BGSS(K) = \sum_{k=1}^{K} \sum_{i=1}^{n_k} \|S_i - c_k\|^2 \tag{3.9}$$

$$WGSS(K) = \sum_{k=1}^{K} n_k \|c_k - c\|^2 \tag{3.10}$$

where, c represents the global centroid between all samples, S_i indicates a particular sample in S, and c_k with n_k specify the centroid and the number of samples of the k^{th} cluster, respectively.
The intuition behind the index is linked with the balance between $BGSS$ and $WGSS$ relative to the number of K clusters and N, with large index values indicating a higher degree of cluster separation and/or inner cohesion. However, given that the index is a statistical construct without an enforced upper-bound limit, the measure does not prescribe a specific threshold of significance. Thus, when used as part of identification of the optimal k number of clusters, the index values should be compared relative to one another. CH indices that contain the maximal value at the peak of a well-defined 'elbow' suggests k with better cohesion and/or separation relative to other CH indices. Conversely, the proportional, linear growth across CH indices suggests minimal changes between cluster cohesion and/or separation, resulting in no identifiable combination of the "best-case" k cluster combination using the method.

3.5.2 Davies-Bouldin index

The Davies-Bouldin index is a distance-measure based validation metric that is commonly used to identify the optimal number of clusters for the given data set [61]. The score is inversely linked with how distant and scattered the constituting clusters are, decreasing when the rate of two (or more) clusters increases, and, vice versa, increasing when clusters and their partitioned samples 'clump up' closely together while having a high rate of scattering between external clusters. The intuition behind the metric is that the clusters that are positioned close to one another (while having scattered boundaries) have a higher likelihood of having n number of samples incorrectly partitioned between one another. Conversely, the larger the distance

between cluster boundaries and the less their samples scattered from one another, the less of a chance that the border-adjacent samples could belong to a neighboring cluster.
The distance measure used by the index utilizes Minkowski metric defined in Eq. 3.6, Within the context of Davies-Bouldin index, the given equation can be reinterpreted as M_{ij} which represents the distance measure between clusters i and j. The measure is derived by calculating the distance between sample vectors that characterize the clusters. Eq. 3.6 can be rewritten as Eq. 3.11, where a_{ki} and a_{kj} represent the kth member of the sample the M-dimensional vectors a_i and a_j which represent the averaged within-cluster similarity measure.

$$M_{ij} = \left(\sum_{k=1}^{M} |a_{ki} - a_{kj}|^p \right)^{1/p} \tag{3.11}$$

Furthermore, if S_i and S_j represent the individual rates of dispersion of cluster i and j, the rate can be calculated via T representing the total number of vectors constituting the given cluster, X which represents individual vectors, and A representing the centroid vector of a given time-series cluster. For cluster i, where T_i, X_n, and A_i, the dispersion rate equals:

$$S_i = \frac{1}{T_i} \left(\sum_{n=1}^{T_i} |X_n - A_i|^q \right)^{1/q} \tag{3.12}$$

The combination of the distance M_{ij} and dispersion S_i and S_j allows calculating the ratio between two measures as defined by Eq. 3.13.

$$R_{ij} = \frac{S_i + S_j}{M_{ij}} \tag{3.13}$$

The ratio represents the quintessential relationship between the total dispersion rate of compared clusters and the distance measure between them. The Davies-Bouldin index $\overline{R}$ is calculated as a mean of ratio maxes $R_i = max\{R_{ij}\}$, where cluster $i \neq j$,

$$\overline{R} = \frac{1}{N} \sum_{i=1}^{N} R_i \tag{3.14}$$

The final index metric derived from the aforementioned process represents the system-wide mean of similarity measures between pairs of clusters deemed most similar. The low score indicates the overall lesser likelihood of cross-mixed, mispartitioned samples. It is important to note that the likelihood does not guarantee certainty, as even clusters with low index scores may still possess relevant outliers that may be incorrectly partitioned into other clusters, or conversely, the high score be representative of the highly specific cluster boundaries. As such, the Davies-Bouldin index is commonly calculated and used together with other validation metrics.

3.5.3 Silhouette score

The Silhouette score is a measure used to determine the quality of the clusters by comparing similarities of all data samples relative to partitioned boundaries of the clusters that the samples were assigned to. The intuition behind the score posits that perfectly partitioned clusters would contain only relevant samples, and that the within-cluster similarity between all samples would be more pronounced than similarity between samples from different clusters. Vice versa, it also follows that a higher degree of similarity between samples from different clusters suggests erroneous partitioning and, potentially, unrepresentative clusters. In order to construct an unbiased, representative silhouette score, it is recommended to use identical similarity measures that were used to generate the clusters.

If a sample set s represents the silhouette score for every sample that was partitioned into a set of clusters C, then the algorithmic loop iterates through samples $s(i)$, generating metrics used to produce the silhouette score. First, a sample i that was assigned to cluster $C(a)$, is compared with all samples within the same cluster, using the chosen similarity measure (e.g. Euclidean distance). The aim of this step is to produce the mean of all similarity scores $a(i)$, representing the average similarity of i relative to own cluster $C(a)$.

The aforementioned procedure is repeated, calculating the mean between i and every other cluster $C(n)$, where $C(a) \neq C(n)$. The resulting set of means is processed in order to identify the most similar cluster $C(b)$ relative to i, e.g. the $C(n)$ with the shortest averaged Euclidean distance between i and the external cluster. The cluster with the most significant similarity is called a *neighbor* of sample i, representing the second-best cluster that i could have been partitioned into. As such, the average similarity between sample i and $C(b)$ is defined as $b(i)$. Finally, the silhouette score of sample i is calculated by comparing both $a(i)$ and $b(i)$, where,

$$s(i) = \begin{cases} 1 - \frac{a(i)}{b(i)} & \text{if } a(i) < b(i) \\ 0 & \text{if } a(i) = b(i) \\ \frac{b(i)}{a(i)} - 1 & \text{if } a(i) > b(i) \end{cases} \tag{3.15}$$

Alternatively, Rousseeuw [62] suggests simplified definition of Eq. 3.15 as a single formula:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{3.16}$$

The listed equations establish the maximum and minimal limits for $s(i)$, where $-1 \leq s(i) \leq 1$. The equation hints that the silhouette score of -1 informs that the within-cluster similarity $a(i)$ is greater than neighbor-cluster similarity $b(i)$, meaning that sample i is located much closer to the cluster $C(a)$ that it was partitioned into. The reverse of 1 indicates that sample i, for all intents and purposes, lies outside the boundaries of $C(a)$, with a high likelihood of belonging to $C(b)$.

The silhouette score provides a more robust approach to cluster evaluation, as the quality of resulting clusters can be assessed not only on the aggregated level of the entire clustering process, but also via providing an overview of each individual cluster. This comes at the cost of $\mathcal{O}(n^2)$ time complexity, as the listed procedure requires a comparison between all samples, making this method less applicable in assessment of clusters formed from raw time-series data, and more fitting for feature-based models.

4

Methodology

This chapter describes the fundamental design steps, techniques, and methodology used for conducting the research study. The chapter starts with defining the overall structure of the undertaken study in Section 4.1. Then, the chapter goes through the methodology defined for the conducted study phases, starting off with Section 4.2 which lists the principal data set and the protocol used for the preliminary data investigation performed on it. Section 4.3 showcases the design used for clustering the time series from the principal data set, detailing the data preprocessing routines, choice of the clustering algorithms, their open source implementations, and hyperparameters used to conduct the experiment. Section 4.4 covers the steps and techniques used to systematically analyze the results from the previous research phases in order to address the research objectives stated in Section 1.2. Lastly, section 4.5 covers the package created to reproduce the data-analysis and clustering parts of the thesis.

4.1 Exploratory research study

To achieve the aforementioned goals, the study was structured into three paths as outlined in figure 4.1. The blue-coded path covered the principal data investigation steps, consisting of data preparation, assessment, and analysis used to gain a better understanding of the principal data such as general characteristics and biases. Knowing these aspects would help to predict how different clustering techniques may behave and what measures need to be taken to mitigate the potential problems. This path denotes the beginning of addressing RQ1 as defined in Section 1.2.

Concurrently, the orange-coded path focused on implementation and benchmarking of different clustering techniques, with the goal of investigating the applicability of the existing algorithms to the principal data set. This path answered the second half of RQ1 through demonstration of functional clustering techniques that could process the serverless invocation traces. The orange path also provided insights into RQ2 as defined in Section 1.2, exploring the applicability of the existing cluster evaluation methods relative to the implemented clustering techniques.

Finally, the green-coded path focused on the systematic analysis, where collected results were sampled, analysed and invocation patterns visually inspected, defined and summarized. The results of which answered RQ3 as defined in Section 1.2 - what characterizations can be derived from the serverless function invocation traces.

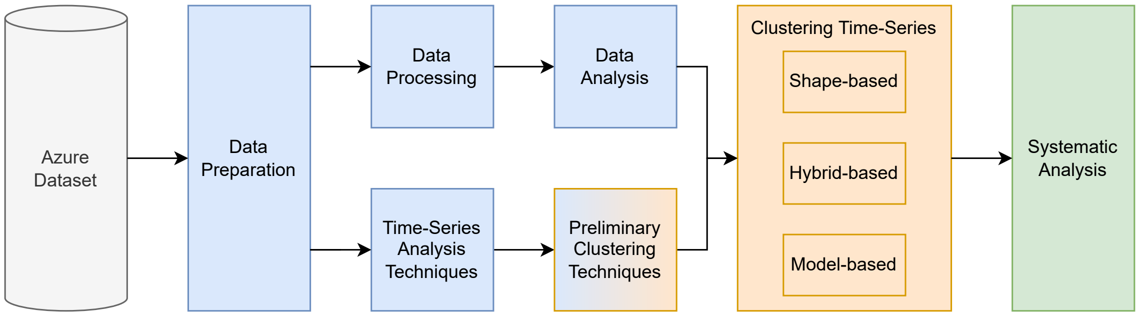


Figure 4.1: A high-level overview of the different research steps undertaken during the thesis.

4.2 Data investigation

This section expands upon the principal data set used in the study as well as methodology used for importing, processing and analyzing the stored samples. Subsection 4.2.1 provides a broad description of the data set, the schema format, key metrics, and the context behind its collection. The subsection 4.2.2 describes ways of accessing the data in the set, as well as the challenges and limitations associated with performing the commonly available operations on the set. Lastly, subsection 4.2.3 covers the steps and statistical tools used to perform the descriptive data analysis on the available data.

4.2.1 Principal data set

The data set used in this thesis is a subset of a data set collected from Microsoft Azure under two weeks in July 2019. The original data set is described and analysed in the paper *Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider* [17].
The data set is a multivariate time-series consisting of invocation counts from multiple applications, function execution time, and memory allocation. These three are split into different files, which all contain an application owner, and an application name. This owner is consistent among all files, which means that they can be cross-referenced and used to analyse the different sets for all owners. In the function invocation count and function duration, the function names are also included, which are the functions that are contained in a specific application that is owned by the owner. Furthermore, in the invocation counts set, the trigger group is included which indicates a specific group that the invocation was triggered from. The owners, application, and functions are hashed using HMAC-SHA256 with secret salts, where each column uses a different salt.
The different trigger groups described in [17] have been grouped into seven different groups: *http, timer, event, queue, storage, orchestration and others.* HTTP is any requests or responses over the HTTP protocol, and timer triggers can be compared to cron jobs, which will invoke functions at a determined interval. Event triggers include the Azure Event Hub and Azure Event grid, where individual or batch processing is used. The trigger group queue includes responses to message insertion in

applications such as Azure Service Bus, Azure Queue Storage, Kafka, MQTT and RabittMQ. Storage triggers indicate database and file systems invocations which were in response to changes in the stored data; it also includes Azure Blob Storage, CosmosDB and Redis. Orchestration triggers are grouped for all invocations triggered that are related to Azure Durable Functions [63], these triggers are used to create native, complex function chaining, and orchestration. Finally, others is the category for the rest of the triggers which do not fit any of the groups described previously such as IoT Hub.
The first set, function invocation counts and triggers, consist of the above mentioned hashed ids for the owner, application, and functions, and 1440 fields of number of invocations per minute for a specific function. The 1440 fields are the number of invocations per minute during each day, which is recorded after the functions execute. This set is split into 14 files, one for each day of the two weeks of the recording. An example of the data is shown in table 4.1.
The second set, function execution time distribution, consists of the above mentioned multiple other fields. The average execution time (ms) across all invocations of the 24−h period, the count of number of executions used for calculating the average, the minimum and maximum execution time during the 24−h period, and the weighted percentile execution time averages. However, the number of invocations per minute in this set may, in some cases, differ from the observations recorded in the first set, as the data for both sets was collected from two spearate log files. Which means that the count could differ and thus the count in this set should just be used in this set. This set is also split into 14 files, one for each day of the two weeks of recording.
The third set, application memory allocation distribution, consists only of the hashed owners and applications, the number of samples used to calculate the average, the average allocated memory across the samples during the 24−hour period, and the weighted percentile of the average allocated memory. For this and the second set, the weighted percentile is over the average, and not the true allocation or execution time. For these sets of files, two files went missing, and there only exists the 12 first days of the two weeks.

HashOwner	**HashApp**	**HashFunction**	**Trigger**	**1.0001**	**...**	**1.1440**
45ae8c...	486e1c...	47a98c...	HTTP	24	...	44
8c64ac...	42c43a...	90c123...	Timer	1	...	1
8a72ce...	5ba32a...	8cd512...	Event	2	...	10
...	...	...	...	...	...	...

Table 4.1: An example table for Function Invocation Counts

4.2.2 Data processing

The data set [19] contains, as mentioned, 42 different files which contain a lot of data. To ease the analysis, with inspiration by our supervisor the files were combined into a single large file, which matched the hashed functions against each other, and prepended the number of the day to each minute, in order to be able to find it. This function was executed for each of the 14 files for each set (12 for memory allocation)

which created three large sets.
When starting to analyse these combined sets a problem became apparent. The data sets were incredibly large, and it would completely fill up the available memory when doing large calculations or loading most of the set. This problem really complicated the analysis of the data and the problem was solved by transforming the data into an HDF5 [64] file, where you are able to load sections of the data set without loading the entire set into memory. Furthermore, to ease the analysing and memory allocation, the HDF5 file was split into multiple NumPy [65] arrays.

4.2.3 Exploratory data analysis

The data analysis was conducted via a descriptive analysis, studying the central tendencies, and the variability. This was to analyse the data, and ensure that the data set is fair and no bias existed. Visualising was used to gain a visual understanding of the data and observe patterns that could be found in a later stage. The entire data set was analysed, however, as mentioned there was greater focus on the different trigger groups and how owners, applications, and functions behave and relate to each other and the trigger groups; especially in the files containing the function invocation counts. However, the same analysis was performed over all of the sets.
First, some preliminary exploratory code provided by our supervisor was used, and the code was further explored and used to combine the single day files into larger $14-$day files ($12-$day for allocated memory). The files were combined on the owners and application, and unlocked the opportunity to study the trigger groups against the function execution time(s) and memory allocation.
Using the larger combined files, the analysis was split into different classes. The classes to analyse are the trigger groups, hashed owners, hashed applications, and the hashed functions, which were all analysed separately. The classes were analysed from most to least invocations, the total and average run-time, the total and average run-time ratio for each class, the average allocated memory, and the ratio of function ownership.
Lastly, weak stationarity, which implies that the time series covariance, mean and variance does not change over time, was analysed to further ensure how different patterns affect each other while clustering.
An algorithmic way of analysing the stationarity of a time-series is the Augmented Dickey Fuller Test [66], which is a unit root test [67] wherein you fit a regression model based on the mean. The test returns a p-value and a statistic, which determines if the pattern is stationary or not. The p-value of less than 5% indicates that concrete root unit could not be established in the examined time-series, suggesting stationarity. Another way for examining stationarity is to compare the statistic against the table of critical values, with the 1% statistic used for the purposes of this study. The value returned is a negative number, and the critical values in the table are also negative. The more negative the value is, and less than the critical value, the stronger the evidence that the null hypothesis should be rejected and thus the data is defined as stationary. Comparing the test statistic against the critical value determines if the series should reject the null hypothesis or not, using this

knowledge the series were grouped into stationarity or non stationarity.
The variance was studied for stationarity where the patterns were split into 10 different buckets in order to gain widespread insight into the diversity of variances, and also, how the different trigger groups fluctuate over time; which further enhances the understanding of the triggers, and their affect on the outcome from the clustering techniques.

4.3 Clustering invocation traces

4.3.1 Data set scopes

Given that the principal data set is a highly dimensional construct [17], considerations had to be made in relation to the common challenges in time-series clustering as detailed in Section 3.2. In order to minimize the computational impact, the principal scope of 2 weeks (20160 observations per time series) of observations was narrowed down to two time-series intervals: (1) the first day of the principal data set, consisting of the first 1440 observations per time-series, and (2) the first hour of the principal data set, consisting of the first 60 observations of each time-series. This deliberation allowed for focusing the clustering process on invocation patterns that arise at those levels of observations.
Using the aforementioned scopes of data, the principal data set was sliced in order to produce two data subsets; Dataset I and Dataset II, both matching with the 1 day and 1 hour time scopes (respectively). The observations outside of the defined temporal boundaries were removed from the subsets, which greatly reduced the dimensionality at the loss of the informational value of the resulting time-series. Moreover, the process results in static time-series being created in Dataset II, as some series contain no observation invocation activity within the first 60 minutes. Such series were, subsequently, dropped out of the data set.

4.3.2 Data preprocessing

Normalization and standardization are two very common techniques used when processing data. Normalization is the process of scaling the data into a range between 0 and 1. The normalization used in this thesis is the Min-max feature scaling, shown in equation 4.1. This process has some effects, the most impactful for this study is if that there exists one or more much larger or smaller outliers the majority of points will be either very small or large and could yield unexpected results, however, it could be detected quite easily by visualizing the data. Another effect is that the variance is not preserved and thus normalization does not work for all clustering methods, such as PCA as presented in section 3.3.4. However, by using normalization, the shape of the time-series will be preserved and the data is bounded by the range $[0, 1]$, which introduces robustness through small standard deviations in the calculated features.

$$X_{MinMax} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{4.1}$$

Standardization is the process of scaling the data into an unbounded range, but to a mean, as denoted by μ, of 0 and a standard deviation, as denoted by σ, of 1 shown in equation 4.2). For standardization, the data loses its shape if that is to be used for example, clustering; however, the variance stays and could thus still be used.

$$X_{Standard} = \frac{X - \mu}{\sigma} \tag{4.2}$$

This thesis uses both normalization and standardization, implemented in the library **sci-kit learn** [68] as MinMaxScaler and StandardScaler, for different parts of pre-processing depending on the algorithm used.

4.3.3 Clustering techniques and implementations

For the purposes of this study, the whole time-series clustering methodology was used to examine the serverless invocation traces, as categorizing time-series as discrete entities can reveal matching patterns individual functions. In particular, shape-based, model-based, and hybrid-approach methods were used (with their definitions provided in Section 3.3.2). These approaches were deemed suitable for reaching the goals of the study, as they allow not only to identify the potential patterns through clustering, but also examine the results and used techniques through the lens of the common performance challenges in the time-series analysis that are described in Section 3.2.

In total, 8 clustering techniques were defined using similarity measures, clustering algorithms, and initialization heuristics pertinent to the selected approaches. Furthermore, open source Python-based packages that incorporated said techniques were used to implement the benchmark cases. Table 4.3.3 summarizes the combination of said techniques, with $M1$-$M4$ using the shape-based approach, $M5$-$M7$ using hybrid-approach through a combination of feature-extraction and PCA, and $M8$ using the model-based approach through the application of the Kohonen's self-organized map with the resulting distance U-matrix used as the input for the clustering algorithms.

The following sections 4.3.4, 4.3.5, and 4.3.6 describe the defined techniques in further detail.

Table 4.2: Summary of the techniques defined for the purposes of the study using whole time-series clustering approaches and relevant algorithms.

Technique	Approach	Algorithm	Similarity measure	Initialization	Package
$M1$	Shape	K-means	Euclidean	randomized	faiss
$M2$		K-means	Euclidean	k-means++	tslearn
$M3$		K-means, DBA	DTW	k-means++	dtaidistance
$M4$		K-medoids	DTW	randomized	dtaidistance, pyclustering
$M5$	Hybrid	K-means, PCA	Euclidean	randomized	faiss
$M6$		K-means, PCA	Euclidean	k-means++	scikit-learn
$M7$		PAM, PCA	Euclidean	k-medoids++	scikit-learn-extra
$M8$	Model	SOM + K-means	Euclidean	PCA weights, k-means++	minisom, hyperopt, scikit-learn

4.3.4 Shape-based clustering

As described in Section 3.3.1, the shape-based clustering approach applies clustering directly to raw time series data. Distance-based measures such as Euclidean distance or DTW are used compute similarity between constituting observations within time series. Furthermore, these measures are used to adjust cluster boundaries with each clustering iteration, grouping time series with the nearest observations together. Implementing this approach was easier than others due to the abundance of the readily available open source implementations with minimal data pre-processing requirements.
The clustering techniques that were defined using this approach consist of 4 steps:

1. Pre-process the input data using with min-max normalization. Optionally, apply additional pre-processing routines depending on the technique-specific requirements, such as differencing ($M3$) and distance matrix pre-calculation ($M4$);
2. Cluster pre-processed input data with the technique's algorithm, such as K-means or K-medoids. For K-means, the number of maximum clustering iterations was limited to 100 for Dataset I and 50 for Dataset II;
3. Calculate internal evaluation indices such as Calinski-Harabasz, Davies-Bouldin, and silhouette score for the resulting clusters;
4. Save the results, including clusters, evaluation indices, and run-time performance metrics collected through each step;

Given the differences between some of the chosen libraries, certain techniques have used different hyperparameters and additional pre-processing routines. The list below provides a detailed description of the used packages and the mentioned differences in implementation:

- $M1$: this technique uses the **Facebook AI Similarity Search** (FAISS) [69] library which was created to perform a similarity search for very large data sets. In particular, the technique uses the library's implementation of a K-means algorithm that supports parallelization through GPU cores via CUDA framework. The results for this technique were produced using the min-max normalized input data, Euclidean distance measure, and randomized centroid initialization in-built into the algorithm's implementation, with the clustering performed using GPU.

- $M2$: this technique is similar in its execution structure to $M1$, with the main difference being the used library and centroid initialization heuristic. The technique uses **Tslearn** [70] library, which implements a customized version of the K-means algorithm adapted for time series analysis. Furthermore, the k-means++ [52] initialization method is used to seed the initial cluster centroids. The implementation is executed exclusively on CPU cores; while the library provides an option for specifying the number of parallel clustering jobs, no significant increase in performance or CPU core utilization was observed from specifying it during the preliminary tests.
- $M3$: this technique uses the **DTAIDistance** [71], a library that is tailored for computing DTW measures. The library also implements a version of K-means clustering algorithm which uses DBA [72] as a time-series averaging strategy. The method computed DBA measure using the maximal DTW shift constraint of `window = 30` and maximal number of iterations for computing DBA of `max_dba_it = 50`. The input data is additionally pre-processed using time series differencing [73] with a low-pass smoothing filter of $\phi = 0.1$ in order to lower the accumulation of noise during measure calculations. Similarly to $M2$, the technique uses a k-means++ cluster initialization heuristic and is processed on CPU cores. The library implements parallelization through C-based worker instances, with significant utilization of multiple CPU cores, observed during the preliminary tests.
- $M4$: the last technique uses the K-medoids algorithm as implemented by the **Pyclustering** [74] library. The library's k-medoids algorithm allows one to use pre-computed distance matrices as the input - a functionality that was used by this technique to shorten the time it takes to perform clustering. **DTAIDistance** [71] was used to compute the DTW-based distance matrix of Dataset I and II using `dtw.distance_matrix_fast` using the maximal DTW shift constraint of `window = 30`. Each resulting matrix was used by this technique for clustering all k iterations. Lastly, clustering was executed on CPU cores with no parallelization.

4.3.5 Hybrid-approach clustering

The inclusion of the hybrid-approach clustering into the study was motivated by the method's ability to utilize the best aspects available to other methodologies in the whole time-series clustering domain, specifically reduction of the computational demands and intrinsic noisiness of the time series data through multiple statistical generalizations. The method used in this study combines the feature-based clustering described in Section 3.3.3, together with the model-based approach based on the PCA in Section 3.3.4. The feature selection component of the technique is a method where specific features of a time series are selected to represent the entire series as statistical generalizations in the feature space. This feature space is represented by a feature matrix, which commonly possesses less dimensions than the large input data, reducing computational demands [75] with a possibility for further adjustments (via inclusion or exclusion of feature) to further increase the clustering performance [76].

The hybrid-approach techniques $M5$, $M6$ and $M7$ are structured in the following sequence of steps, with Step:

1. Transform the input data into the format compatible with the feature extraction methods;
2. Perform feature extraction and selection using efficient parameters;
3. Perform data cleanup on the extracted features in order address problematic values, e.g. infinity and NaN;
4. Apply PCA on the extracted features;
5. Identify a set of the most significant principal components that reach the minimal explained variance ratio;
6. Cluster selected principal components with the technique-specific algorithms, e.g. K-means or K-medoids;
7. Calculate internal evaluation indices such as Calinski-Harabasz, Davies Bouldin, and Silhouette scores for the resulting clusters;
8. Save the results, including clusters, evaluation indices, and runtime performance metrics collected through each step;

The process begins with transforming Dataset I and II into a format compatible with an automated feature selection implement chosen for this study — the **TSFresh** [77] library. Next, the feature selection is performed using `EfficientFCParameters` [78], which is an expansive list of statistical features curated by **TSFresh** that excludes the most computationally demanding features. The omitted parameters have high computational and spacial demands, which was observed to cause termination of the entire process due to the lack of available memory. Another parameter offered by the library — `minimal` — disables all but the most basic features, and is intended to be used for quick testing and validation [79]. The feature vector produced by the library may contain values incompatible with the clustering process, such as NaNs or values indicating positive/negative infinity. As such, clean up procedure is performed in order to remove such features from the set before proceeding with the next steps. The resulting feature set still contains 466 features after the clean-up.

Next, the PCA model is applied to decompose the extracted features into smaller dimensions, using methods as described in Section 3.3.4. The initial minimum target threshold was set to 80% explained variance for both input sets. The threshold was set according to the studies that used PCA; an analysis of cDNa [44] and clustering of financial time series data [76], to strike the balance between the prevention of training set overfitting and loss of information.

The PCA was performed using the implementation provided by the **Scikit-learn** [68] library. The library uses the Single Value Decomposition (SVD) to project the input vectors into smaller dimension, using the `full` setting for SVD's solver. The number of investigated components was set to 60, matching the maximum length of the Dataset II samples.

The explained variance of each principal component is visualized in Figure 4.2, with the 1st component describing ≈ 40% and 38% of explained variance of the feature vectors extracted from Dataset I and Dataset II (respectively). The graph further indicates that the preliminary threshold variance threshold of 80% can be reached with 40 components for Dataset I, and 16 components for Dataset II. For the purposes of the study, input vectors of 49 and 20 principal components were selected for Dataset I and Dataset II, in order to equalize the explained variance of used vectors at ≈ 82.8%.

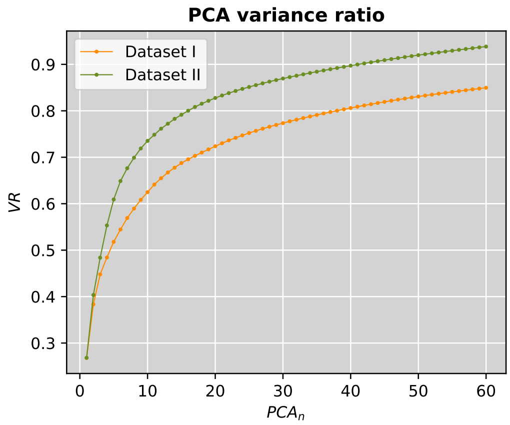


Figure 4.2: Overview of the explained variance ratio VR of the most representative subsets of principal components PCA$_n$ derived through PCA.

Finally, clustering is performed according the the clustering techniques defined for this study:

- $M5$: this technique reuses the **Facebook AI Similarity Search** (FAISS) [69] library, similar to the shape-based technique $M1$. Instead of the normalized time-series, the aforementioned subset of principal components is used as the input, otherwise reusing the same hyperparameters as $M1$: randomized centroid initialization, Euclidean distance measure, and clustering using the GPU.
- $M6$: given that the subset of the selected principal components is not a time-series, this technique could not reuse the implementation used by the $M2$ technique. Instead, the technique uses **skikit-learn** [68] library's generic implementation of the K-means algorithm. Otherwise, the technique shares similar hyperparameters as $M2$ technique.
- $M7$: this technique uses **skikit-learn-extra** [80] implementation of the general K-medoids algorithm called Partitioning Around Medoids (PAM) [81]. The technique uses the Euclidean distance measure and the k-medoids++ initialization heuristic inspired by k-means++ [52]. Similar to $M4$, the implementation is executed on CPU cores.

4.3.6 Model-based clustering

As defined in Section 3.3.1, the model-method clustering parses the raw time-series data through a statistical model first, deriving the model-specific parameters. Then, the parameters are organized into an input set that is processed through the clustering algorithm. The $M8$ technique implemented in this study uses the Kohonen self-organizing map (SOM) as such model, and applies the K-means clustering algorithm to the parameters derived from it, i.e. the distance U-matrix.

The overall process is structured into 9 steps, with steps 2-4 describing the initialization and training of SOM, while steps 5-7 cover K-means clustering and linking the results back to the input time series:

1. Pre-process the input data with min-max normalization;
2. Approximate the optimal number of nodes and grid size of the self-organizing map;
3. Perform hyperparameter tuning to find the optimal σ neighborhood radius and η learning rate relative to the map's size and sample-size of the input data;
4. Train the self-organizing map using pre-processed input data;
5. Cluster SOM's distance matrix (also known as U-matrix) using K-means algorithm;
6. Prune clusters which contain only the SOM nodes with no assigned time series;
7. Trace the remaining clustered SOM nodes back to the time series samples that were assigned to them;
8. Calculate internal evaluation indices such as the Calinski-Harabasz, Davies Bouldin, and Silhouette scores for the resulting clusters;
9. Save the results, including clusters, evaluation indices, and runtime performance metrics collected through each step;

The technique uses **minisom** [82] library for representing the self-organizing maps. The optimal map topography is calculated using the heuristic proposed by Vesanto and Alhoniemi [83]

$$M = 5\sqrt{S_N} \tag{4.3}$$

where S_N represents the total number of samples in the input set, i.e. time series, and M is the suggested number of SOM nodes to be used for mapping said samples.

The next step after establishing the general topography is focused on the optimization of SOM's key hyperparameters in order to improve the training outcomes. Specifically, the σ neighborhood radius that is thoroughly used to select the qualified neighboring nodes, and the η learning rate that is used to control the magnitude of the adjustments to the node weights throughout the SOM's training. The $M8$ technique achieves this through the use of the **hyperopt** [84] library which implements the Tree-structured Parzen Estimator (TPE) [85], enabling sequential hyperparameter optimization by estimating hyperparameter improvements through Bayesian inference. Figure 4.3 visualizes the aforementioned optimization process using TPE, with two test trials focusing on finding σ neighborhood radius and η learning rate by minimizing SOM's quantization or topography-based error rate, while Table 4.3 details the optimization specifications used in the $M8$ technique and identified hyperparameters used in training SOM.

Table 4.3: Results for the SOM hyperparameter optimization relative to the selected error metric. The σ neighborhood radius and initial η learning rate were derived over the course of the specified optimization epochs. The last column showcases the total time of the process.

Optimization metric	Dataset	Map size	Epochs	σ	η	Total time
Quantization error	I	29x29	200	0.4071	0.5460	18m. 42s
Quantization error	II	26x26	200	2.2643	0.5354	1m. 23s

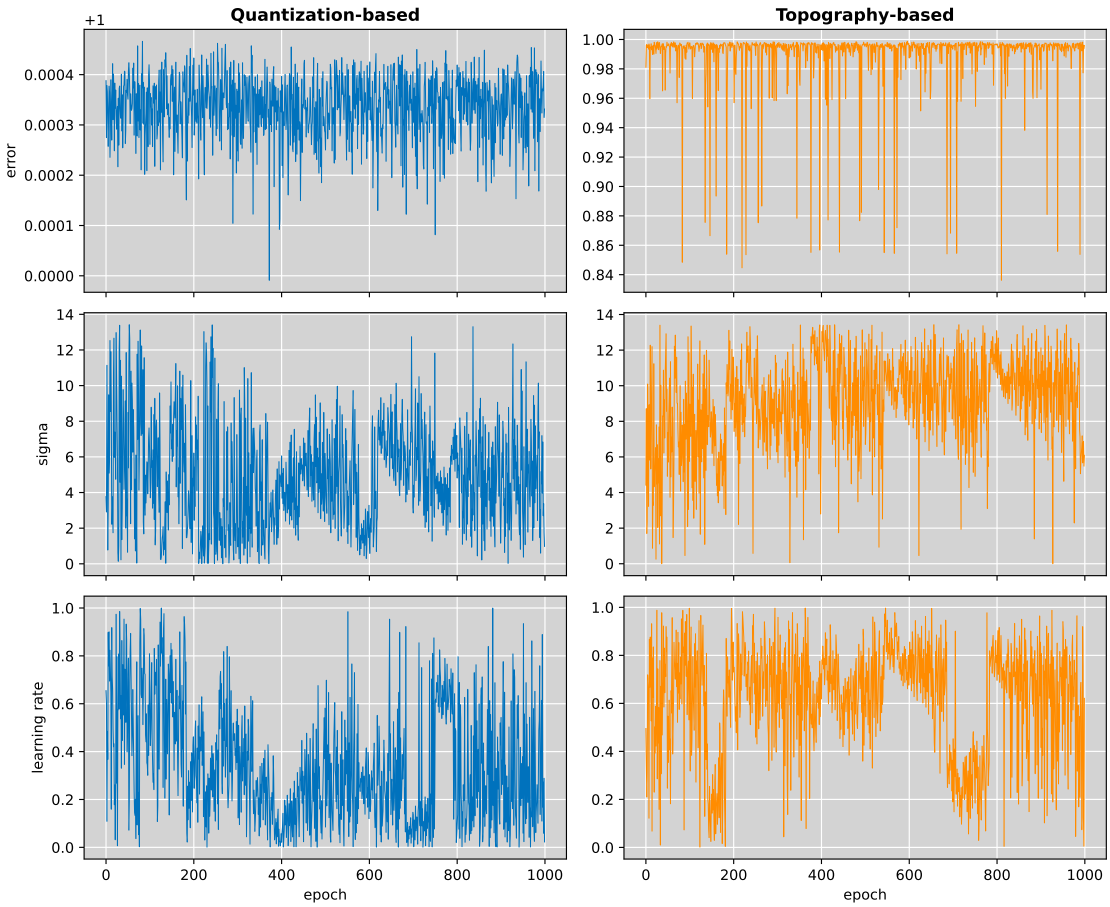


Figure 4.3: An example of SOM hyperparameter optimization for the σ neighborhood radius and η learning rate using **hyperopt** library. The optimization is performed in two separate trials over 1000 epochs, with each trial observing either quantization or topographic error metric. The process discovers a combination of hyperparameters which produce the lowest error metric over the course of optimization.

Using different specifications discovered through optimization and map size estimation for Dataset I and II, the map is trained on the input data over the course of 500 epochs. Rather than using randomized node weight initialization, $M8$ uses PCA-based initialization in-built into **minisom** library. Additionally, the default `gaussian` neighborhood function is used for identification of the node's neighborhood during the weights' adjustment phase of each training iteration. The resulting distance matrices (also known as U-matrices) of both datasets are visualized in Figure 4.4. Each cell represents the node in the SOM's topography, with the color-coded values indicating the normalized sum of all distances between the node and its neighbors. Darker colors indicate regions with greater separation between neighboring nodes, while brighter colors represent closer proximity. Large, similarly colored regions suggest relative uniformity in the distances between nodes.

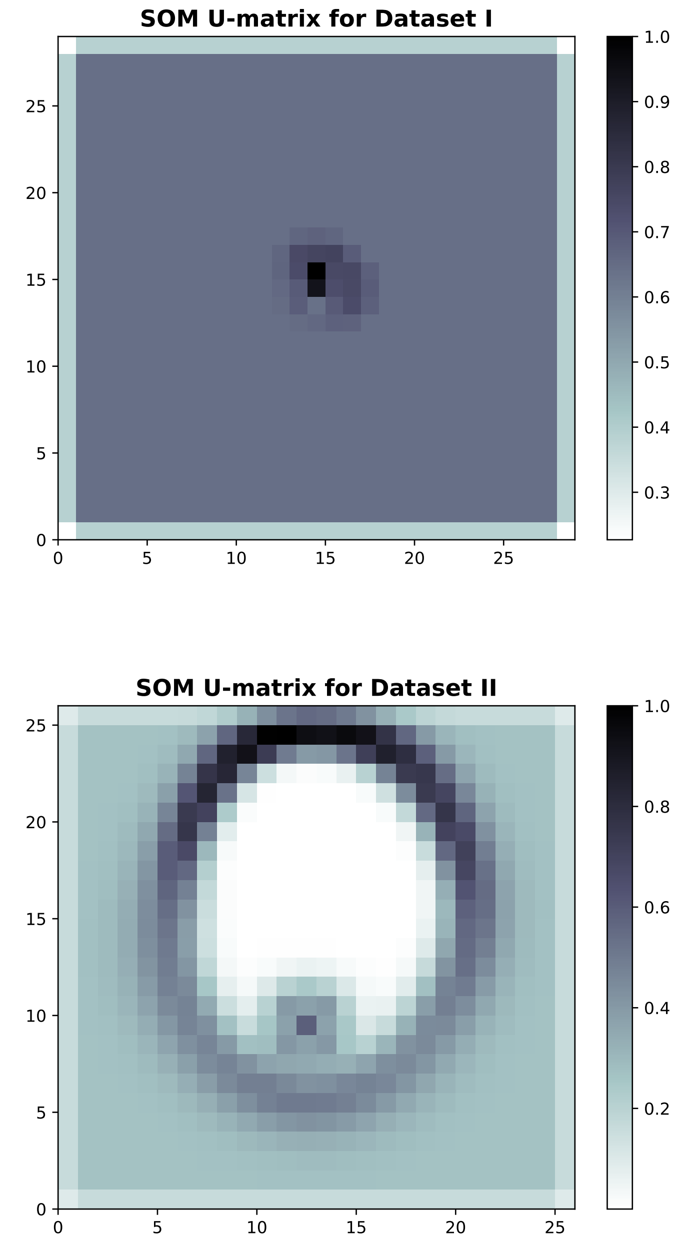


Figure 4.4: Overview of the resulting distance map derived from processing Dataset I and II through their versions of self organizing maps.

The final step in $M8$ technique uses the SOM U-matrices as input sets for the K-means algorithm taken from **scikit-learn** [68]. Similar to other techniques, the process is performed using Euclidean distance measures with the maximum number of clustering iterations limited to 100 for Dataset I and 50 for Dataset II, all computed on CPU cores with minimal parallelization. Lastly, time series used as input sets are linked with clusters by matching clustered SOM nodes with series that were assigned to them during SOM training.

4.3.7 Test environment

All clustering sessions were carried out on a local machine. The environment was hosted on Windows 10, powered by Intel i7-11800H Core 2.30GHz CPU (8 cores / 16 logical threads), with 64GB of RAM. The second environment was hosted on Windows 11, powered by AMD Ryzen 5900X 3.90GHz CPU (12 cores / 24 logical threads) with 48GB of RAM. Given the CPU-intensive nature of the calculations involved with the selected techniques, especially the ones dealing with the raw time series data, the total data case execution time could have been impacted by side processes. In order to mitigate the magnitude of that impact, the test environment was stripped of all external applications which could interfere with clustering runs. To further minimize the impact of the system processes linked with network services of the operating system, the test environment was isolated from the internet for the duration of the experiment. It should be noted that such interference could not affect the end results produced by the clustering techniques; rather, the measures were taken to minimize the total execution time of all data cases.

4.4 Systematic data analysis

To analyse the results from the clustering and to find valuable characteristics a systematic data analysis has to be performed on the results in order to gain better insights gained from the clustering process. As such, the analysis was performed in three steps:

1. Linking the collected cluster results with the original data and insights gained from the exploratory data analysis;
2. Summarize commonalities and dissimilarities observed in the gathered data into patterns;
3. Compose the identified patterns into characterizations, in the form of recurrent workload shapes;

Once all results from the executed clustering techniques are collected, the "best-case" techniques are selected based on the previously calculated internal validation metrics. The selected techniques are further analysed by examining the time series assigned to the clusters according to their data characteristic, i.e. the trigger groups, owners, distribution between clusters, the rate of invocation activity across the clusters and more. To provide a better classification of the varying cluster sizes observed in the results, a reference schema with size definitions was created and is

presented in Table 4.4. Furthermore, up to 10 time-series samples are selected from each technique's cluster, and plotted with their corresponding function ID for both the normalized and non-normalized (original) data. This is followed by a visual inspection of the generated plots, comparing samples within similar clusters in order to gain a greater understanding between the normalized and non-normalized time series samples and their shared and/or distinct characteristics.
The notes collected from the previous step are summarized as a list of features which describe each cluster and their constituting time-series, with anomalous findings noted down for further analysis (in order to identify the source of the apparent distinctness). Individual samples are further compared with the cluster invocation rate and activity metrics collected during the previous step. Finally, all observed patterns and corresponding information is collected and summarized into a set of shapes with a preliminary name and short description of the core characteristics that define them. The defined shapes are assigned to the clusters of the "best-case" techniques, anomalous findings are presented, and the results are further discussed in Chapter 6.

Table 4.4: Table defining the different cluster sizes, depending on the total number of functions clustered.

Definition	**Min samples**	**Max samples**
Empty	0	0 functions
Single-function	1	1 function
Scarce	2 functions	1% of total
Small	1% of total	10% of total
Medium	10% of total	30% of total
Large	30% of total	50% of total
Dominant	50% of total	90% of total
Super-cluster	90% of total	100% of total

4.5 Reproducibility

A package with all source code was produced and released to enable anyone to reproduce all finding in this thesis [8]. The package will include the results, the source code for data preparation and analysis in structured Jupyter notebooks. With instructions and requirements to reproduce it. It must me noted that hardware is a considerably limiting factor, and must be taken into consideration. Using a less powerful computer will yield a very long computational time.

5
Results

This Chapter presents the final results collected throughout the study, with respective sections listed in order of of the key research phases described in Chapter 4. First, Section 5.1 showcases the results obtained through the descriptive analysis of the principal data set as defined in Section 4.2. The analysis results are followed by Section 5.2 which describes the outcomes of the experiments and activities conducted according to the design protocol described in Section 4.3. The final Section 5.3 summarizes the findings via the results obtained during the systematic analysis phase (as defined in Section 4.4), bridging the contexts of the aforementioned research steps. Lastly, it merits mentioning that none of these results could be generalizable to another provider, due to the fact that all of the results were gathered from the data made public by Azure.

5.1 Exploratory data analysis

The result from the descriptive statistic analysis and the more specific analyses done on the data set both show some interesting results that could skew the results from the clustering. Descriptive statistics show that the distribution of values over all trigger groups are overall similar during all days, with a similar mean and standard deviation across all trigger groups. Similarly, the results are the same for memory allocation and function execution duration.

The number of invocations per minute was further analysed for each trigger group. Figure 5.1 shows invocations per minute on the y-axis and the minutes since the two weeks recording started on the x-axis. From these, one can observe distinct patterns on how the number of invocations differ during the weeks between the different groups. All patterns have their outliers from the apparent pattern but only HTTP, 5.1 g), have some outliers that can really matter in the clustering algorithm, seen in Figure 5.1 h) which occurs after 9 days. An explanation to why is not possible due to the anonymity of the data.

Studying the graphs, all have their own specific pattern that easily can be identified. 5.1 a) shows the expected behaviour of a timer showing multiple lines of different number of invocations, with a few exceptions that are above 100 invocations per minute. 5.1 b) shows a very disperse graph with a range of number of invocations between $0-200000$. Most of the invocations are up to 25000, and one can see a faint outline of a small peak each day. 5.1 c) has an interesting pattern, three distinct lines, one with a lot of smaller invocations rates, one that is steady in a line that abruptly in the end rises and grows, which similarly to 5.1 h) can not be deduced to why due to the anonymity, and one sine-shaped one with 10 peaks. 5.1 d) shows a large number of functions with fewer invocations per minute, and similarly to the others, a sine wave pattern with 14 peaks that has a much larger amplitude up than down from the "mid-line" of the wave. 5.1 e) shows similarly to the other ones that the majority of the functions are not invoked as frequently (in comparison to the others). This trigger group also has 14 peaks which peak up to around 12000. However, there also exists some invocations which go up towards 25000, but they are small in number (relatively) and thus should not bias the output. 5.1 f) shows the majority cluster of invocation rates are in the bottom, with a couple larger rates that could be seen. These larger ones are occurring once a day, and additionally, a handful of invocation number per minute rise above 2500 (and as with 5.1 e), nothing that will be of any significance). Lastly, 5.1 g) shows a lot of small number of invocations in the bottom, in between $0 - 10000$. Over that, a wave pattern occurs which has a spike during each day of the two weeks. Above that, there is a large cloud and one very thin cloud of invocations ranging from $50000 - 80000$ invocations per minute. However, this graph has omitted outliers, that under a 20 minute period the number of invocations grow exponentially larger up to around 300000 invocations per minute. These invocations are shown in better detail in figure 5.1 h).

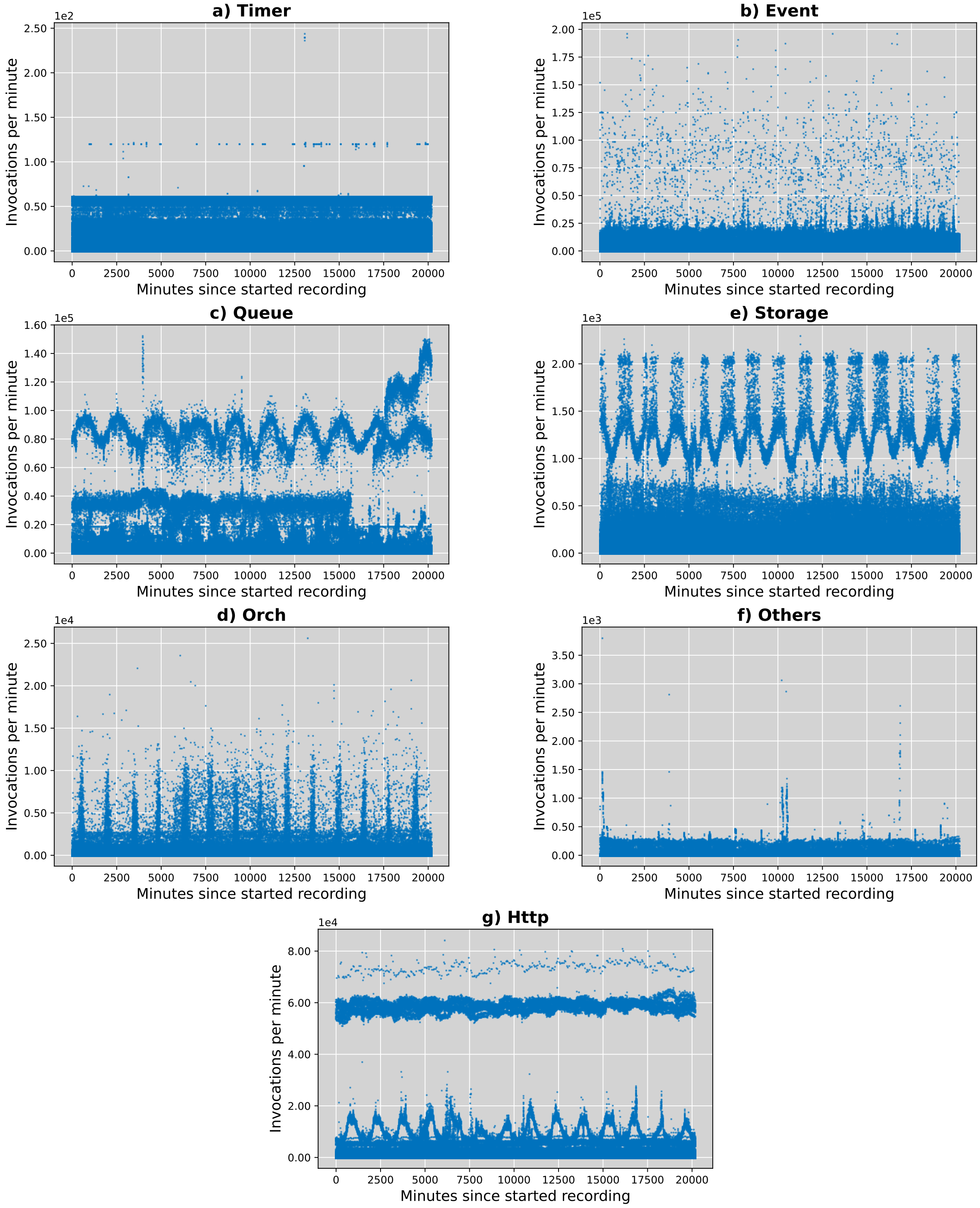


Figure 5.1: Plots of all invocations during all minutes of the two weeks for all trigger groups and the omitted outliers from the HTTP trigger group.

5.1.1 Distribution of function ownership

In Figure 5.2 the trigger groups are seen on the y-axis, and the percentage of the total amount of functions in orange, and invocations in blue. This shows that the timer trigger group held a majority of the functions in the data set, with HTTP and Queue following each other with a little over 20% and 10% respectively. The last groups are just over 10% collectively.

Further analysis of the different application owners and the applications themselves yielded the observation that no owners or applications own a majority or larger proportion of functions. If this had not been the case, disproportionate ownership or prominence of a small number of applications would have lowered the generalizability of the results due to biased data.

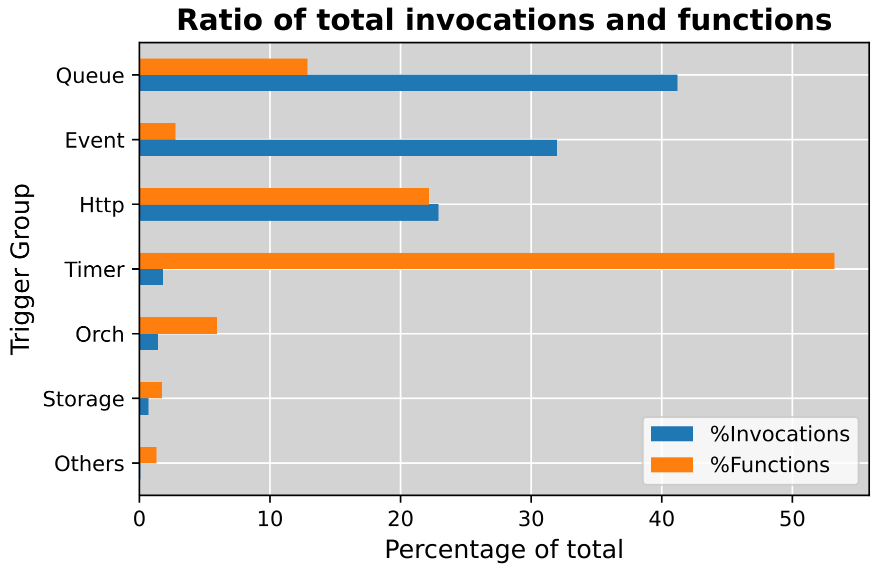


Figure 5.2: Comparison between the different trigger groups on what percentage of total number of invocations and total amount of functions in the dataset.

5.1.2 Run-time metrics

Figure 5.3 illustrates the percentage of the total run-time for the different trigger groups. The Timer trigger group is considerably larger than the others (with the majority of the total run-time), followed by Queue and HTTP. Similarly, in terms of function ownership these three groups also have a large majority.

Analysing run-time for the different owners, applications, and functions, calculated using the maximum execution time, indicate that the applications and functions do not have any considerable majority or larger proportion that could skew results. However, one owner owned a tenth of the total run-time in the set. This owner owned the most amount of applications and functions of all owners in the set. However, the functions' total invocations are not a large share of the total invocations. The owner include 355 Queues, 199 Timers, and 96 HTTP functions, with that amount of Timers and Queues (which has the most total run-time of the trigger groups), explains why it also has the most total run-time.

The subsequent observation that the invocation proportion is low thereby improves generalizability, since the algorithm focuses on invocations to characterise patterns.

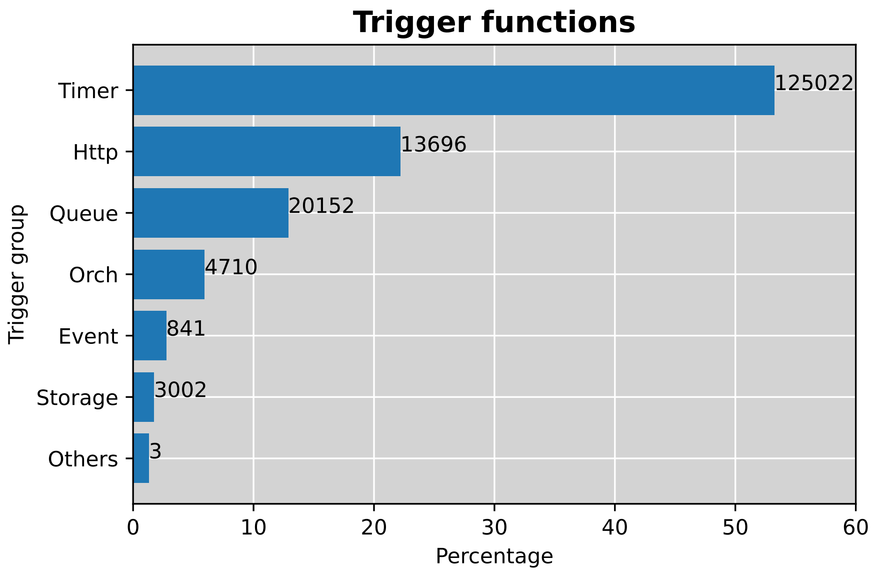


Figure 5.3: Percentage of the total run-time for the different trigger groups and where the numbers define the average runtime for each trigger group over all functions.

5.1.3 Function prominence factors

The blue bars on Figure 5.2 is the total number of invocations each trigger type has compared to the total number of invocations in the data set. This graph shows a different result than what have been shows above. The Timer trigger group, which had been the clear majority, are no longer as prominent. Queues, Events, and HTTP are instead the largest with more than 90% of the total invocations. Further analysing the different owners, applications and functions, shows that the top five owners owns 45% of total number of invocations. Which skew the results in their direction and decreasing generalizability for the results from the algorithm. The top five applications with most invocations share 41.5% of the total number of invocations, which share the same possibilities as the owner. Finally, for the functions, the top five share 38% of the total number of invocations during the two weeks of observations, and the trigger types for these functions are (in order): (1) Queue - 1683728143 invocations - 35.5% of total Queue invocations, (2) HTTP - 1187337050 invocations - 45.0% of total HTTP invocations, (3) Queue - 900734217 invocations - 19.0% of total Queue invocations, (4) Event - 336710817 invocations - 9.1% of total Event invocations, and (5) Event - 307327546 invocations - 8.3% of total Event invocations. This supports the findings from Figure 5.2 which displays that Queues, Events and HTTPs have the most number of invocations over the entire dataset.

5.1.4 Workload stationarity

Analysing workload stationarity begins with analysing the variance for all functions. Figure 5.4 shows the range of variance for all trigger groups split into 10 buckets, and also split into different periodic subsets: hourly, daily, and weekly. From the results we can see that the spread of the variance is much better for the weekly subset and the daily subset, rather than the hourly subset.

Most of the hourly variances are high, this result stemming from the invocations in this set either laying very far away from the mean of the invocations (and fluctuating a lot), or, a steady stream of higher invocations. The 1462 functions that have 0 variance indicate towards that it is a constant value of invocations at the mean. Comparing this to figure 5.5 for the hour subset, one can see that the majority of the functions are stationary, i.e a tiny fluctuation in invocation counts or a constant number of invocations during the hour. The very small subset of values in the *non stationary* category show that many of the functions do fluctuate in invocation counts, but also that many that have a high variance do not fluctuate that much — they only deviate from the mean.

The daily subset illustrates a better spread of variance, which indicates a better range of functions with different invocation counts. After observing the stationarity in figure 5.5, one concludes that $\approx 92\%$ of functions in this set are stationary. This result indicates that many functions are stationary above the mean, and subsequently that many functions could be timers (or behave like timers) with stationary values.

The weekly subset show a similar result as the daily subset, with a great spread and with almost all functions as stationary. This indicates towards a timer behaviour or only stationary values, however, with a great spread in variance.

Both the daily and weekly subsets that a majority of functions were being clustered with low variance and high stationarity, and few functions were clustered with high variance and low stationarity. Functions could, however, be clustered into "sine wave"-like patterns with different amplitudes and frequencies (which also would be considered stationary).

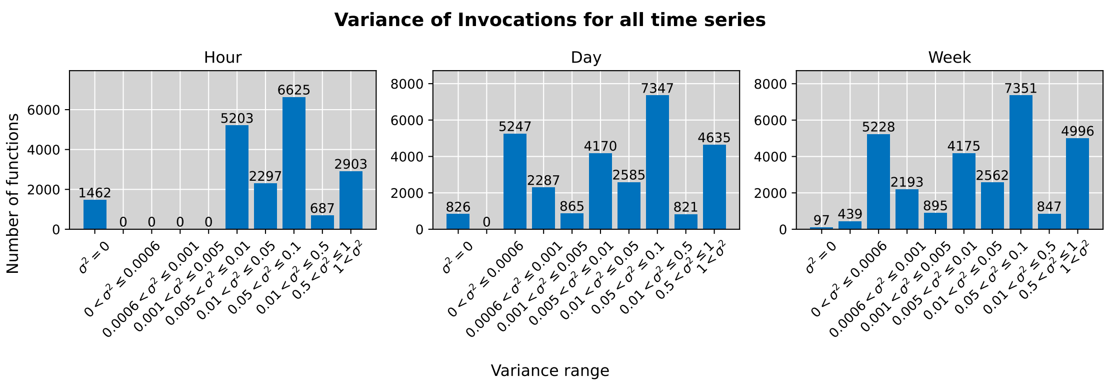


Figure 5.4: Variance of the invocation split into 10 buckets studied for subsets hour, day and week

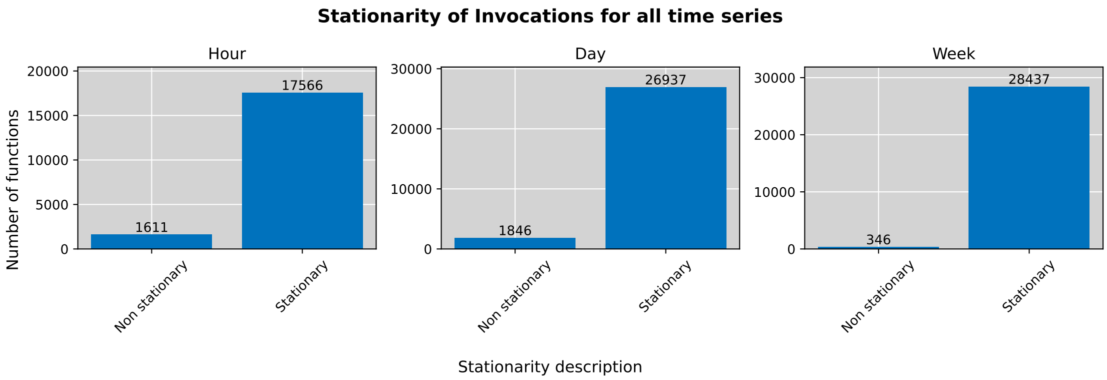


Figure 5.5: Ratio of stationary functions over all trigger groups for subsets hour, day and week

5.2 Clustering

5.2.1 Used data scopes

This subsection contains the statistical descriptors that were collected from spliced data subsets, which were derived according to the data scopes proposed in Section 4.3.1. The resulting scopes and the distribution of serverless function types within the subsets is presented in Table 5.1, with the difference in scale further visualized in Figure 5.6. Overall, Dataset I maintains a similar distribution of trigger types and the number of samples as the Principal dataset, with a dramatic reduction in the number of invocation activity represented by the subset. Moreover, Dataset II contains a smaller number of samples than other sets, while the overall distribution of the trigger types (with the exception of 'Other' and 'Storage' triggers) continues to match other data sets. This difference is linked with the pruning of inactive (i.e. zero invocation activity) functions as described in Section 4.3.1.

Table 5.1: General overview of the data sets considered for clustering activities. Dataset I and II were used in clustering, covering observations across 1 day and 1 hour respectively. The table depicts the total number of serverless functions, the total number of observed invocations (based on the time-span covered by the respective data sets), and the number of functions based on the type of the function triggers. The distribution of triggers changes between Dataset II and Dataset I, with storage becoming the least represented trigger type.

Dataset	Functions	Invocations	Event	HTTP	Orchestration	Other	Queue	Storage	Timer
Principal	28783	1.15e+10	791	6385	1706	376	3707	499	15319
Dataset I	28783	8.40e+08	791	6385	1706	376	3707	499	15319
Dataset II	19177	3.11e+07	633	2167	805	333	2167	265	11061

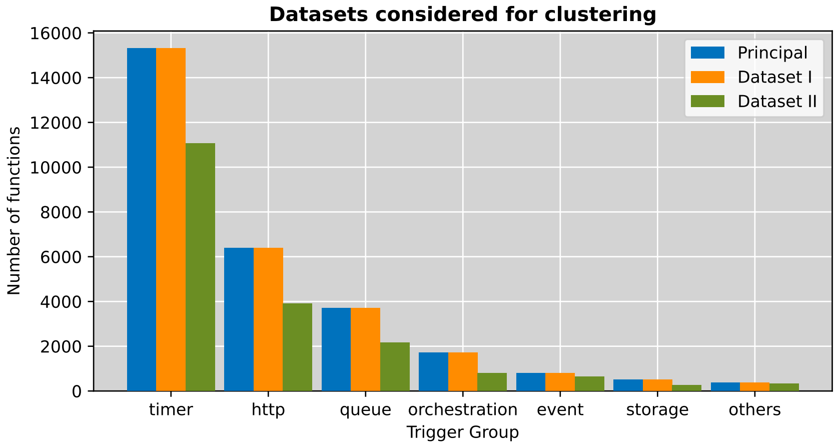


Figure 5.6: Overview of the serverless function distributions by trigger type across all data sets considered for clustering. All functions with missing observations and inactivity were pruned out of the selected data sets. It must be noted that pruning samples in such a way resulted in the reduction of the total number of function samples between Dataset II and I.

5.2.2 Benchmarking clustering techniques

This section presents the results for performance metrics collected during application of the clustering techniques defined in Section 4.3. the benchmarking trials consisted of executing each clustering technique in isolation, ensuring that the test environment's resources were allocated exclusively to the clustering process. As specified in the methodology, each technique iterates through the k number of target clusters, starting with 5 cluster groups. The process is incremented through $k = 30$ target clusters, covering 25 possible partitioning cases per dataset-technique combination.

The baseline clustering process that all techniques share is structured in three phases. First, the technique clusters the input data using the defined library implementation and provided hyperparameters. Then, the results produced by the implementation are transformed in line with the unified data format, standardizing the output results across all techniques. Lastly, the internal evaluation metrics are computed using the clustering results. All produced results and metrics are archived and saved as persistent artefacts in the test environment. The amount of time used to complete all the mentioned phases is presented in Figure 5.7, with each graph visualizing the summed run-time to process each target k cluster using the chosen techniques, distance measures, and datasets. Table 5.3 summarizes the run-time further, depicting the shortest, longest, and average run-time cases per k, as well as the total amount of time it takes to perform all trials for the selected clustering techniques.

The techniques based on k-medoids algorithms such as $M7$ and $M4$ demonstrate disproportionately longer clustering runs relative to other investigated methods, even with application of dimensionality reducing techniques, such as PCA. The model-based approach used in the $M8$ technique produces the results faster than other methods, as long as the optimized hyperparameters described in 4.3.6 are known before the trial runs.

Table 5.2: Overview of the computation times for each combination of chosen clustering techniques and datasets. Each technique iterated over the range of 5 through 30 k clusters. The table showcases the shortest, average, and longest clustering run for one of the k clusters in seconds, as well as the total runtime over the aforementioned range.

Method	**Algorithm**	**Measure**	**Dataset**	**Clustering per k, seconds**			**Total runtime**
				Shortest	Average	Longest	
$M1$	K-means	Euclidean	I	5.45	12.79	22.44	5m 33s
$M1$			II	0.39	2.22	7.38	58s
$M2$	K-means	Euclidean	I	16.61	37.31	55.11	16m 10s
$M2$			II	5.47	7.36	15.00	3m 11s
$M3$	K-means, DBA	DTW	I	200.94	1149.33	2110.11	8h 18m
$M3$			II	10.92	56.35	141.97	24m 25s
$M4$	K-medoids	DTW	I	857.12	8054.12	28578.50	58h 10m
$M4$			II	495.59	4391.81	9383.48	31h 43m
$M5$	K-means, PCA	Euclidean	I	0.78	3.16	9.06	1m 22s
$M5$			II	0.30	1.35	2.88	35s
$M6$	K-means, PCA	Euclidean	I	1.58	8.97	17.50	3m 53s
$M6$			II	1.30	5.21	9.80	2m 16s
$M7$	PAM, PCA	Euclidean	I	1305.41	14232.16	27878.17	<u>102h 47m</u>
$M7$			II	420.67	4803.20	13456.81	<u>34h 41m</u>
$M8$	SOM, K-means	Euclidean	I	0.25	1.08	2.14	28s
$M8$			II	0.20	0.53	1.58	14s

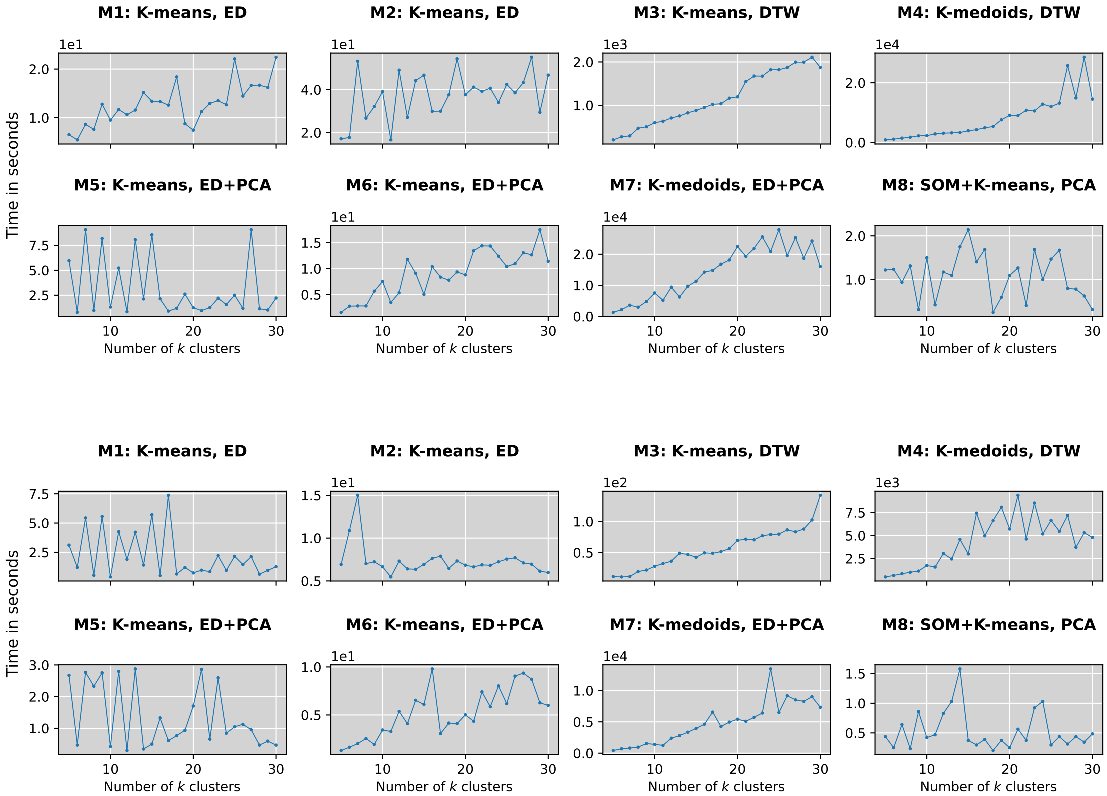


Figure 5.7: Overview of the performance metrics collected for all defined clustering techniques and dataset combinations. Each subfigure visualizes the total amount of time it takes to produce the results for k number of clusters using the selected techniques.

5.2.3 Determining the optimal number of clusters

This section summarizes the internal evaluation metrics that were defined in Sections 3.5.1, 3.5.2, and 3.5.3, and computed for each clustering technique. The optimal number of samples for each technique is based on a selection of k that maximizes the beneficial effects across all three indices. Both the selected k and the associated metrics are presented in Table 5.3, with the progression of changes in the metrics throughout clustering trials for Dataset I and II, visualized in Figure 5.8 and Figure 5.9 (respectively). Given that no clear determination could have been made regarding the shared optimal k across the techniques, the principle of maximizing the value across evaluation indices was reused between the techniques that applied similar clustering approach. In the end, three "best-case" techniques were selected for each of the applied approaches: $M2$ for shape-based approach, $M7$ for the hybrid approach, and $M8$ for the model-based approach (selected by default).

Table 5.3: Overview of the internal evaluation metrics and the associated optimal k number of clusters selected for each tested clustering technique.

Method	Algorithm	Dataset	Internal evaluation metrics			Optimal k
			Calinski-Harabasz	Davies-Bouldin	Silhouette	
$M1$	K-means	I	17618.70	1.75	0.94	9
$M1$	K-means	II	8410.18	1.27	0.88	14
$M2$	K-means	I	135576.33	0.39	0.98	19
$M2$	K-means	II	209740.21	0.31	0.98	21
$M3$	K-means, DBA	I	31721.48	0.73	0.99	13
$M3$	K-means, DBA	II	35172.24	0.51	0.98	20
$M4$	K-medoids	I	114212.55	0.89	0.93	15
$M4$	K-medoids	II	132599.49	0.59	0.93	13
$M5$	K-means, PCA	I	2076.12	1.72	0.18	10
$M5$	K-means, PCA	II	1488.57	1.27	0.51	28
$M6$	K-means, PCA	I	7341.77	0.68	0.98	5
$M6$	K-means, PCA	II	11808.17	0.58	0.49	19
$M7$	PAM, PCA	I	1043.25	1.51	0.25	22
$M7$	PAM, PCA	II	3937.11	1.18	0.55	28
$M8$	SOM, K-means	I	560.0	1.58	0.97	7
$M8$	SOM, K-means	II	3079.12	1.18	0.96	7

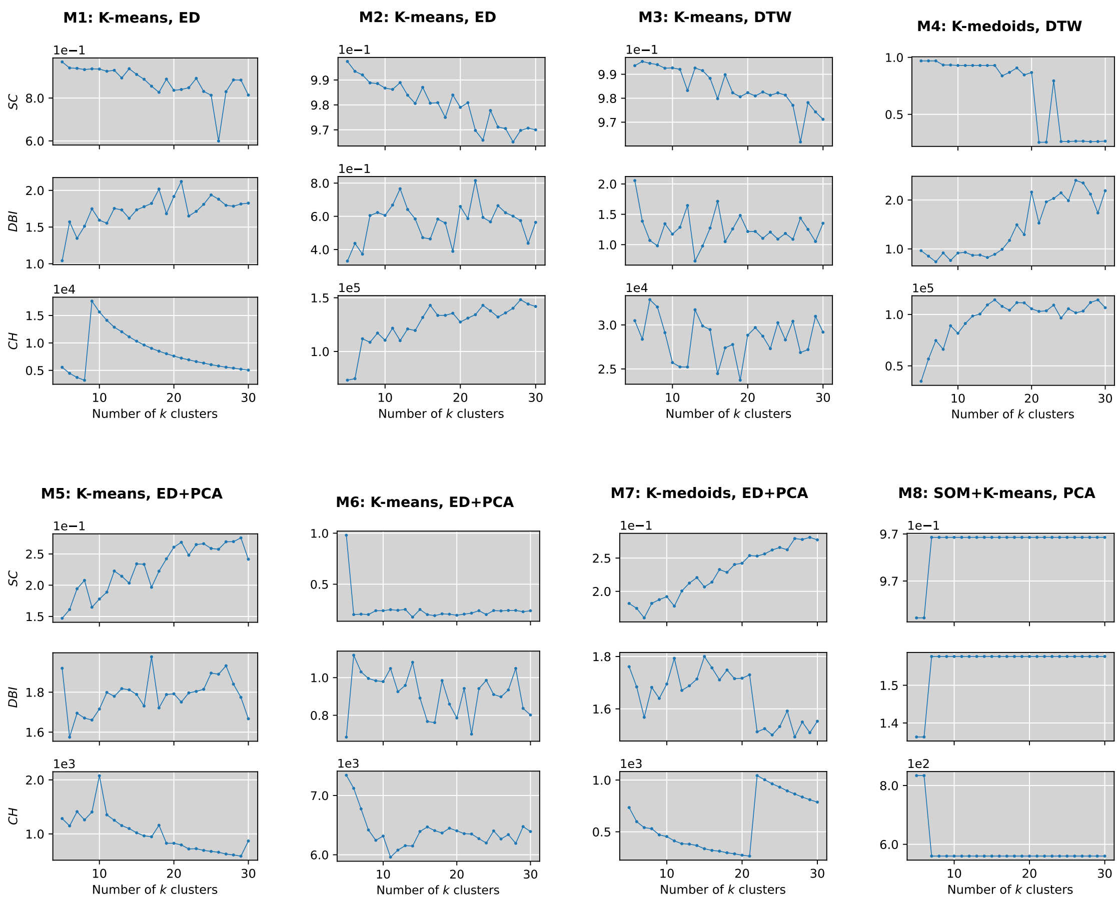


Figure 5.8: Overview of the internal validation metrics of the clusters derived from Dataset I (1 day). Three metrics were used to evaluate variations of the clustering techniques and the number of chosen k clusters: the Silhouette score (SC), Davies-Bouldin index (DBI), and Calinski-Harabasz index (CH).

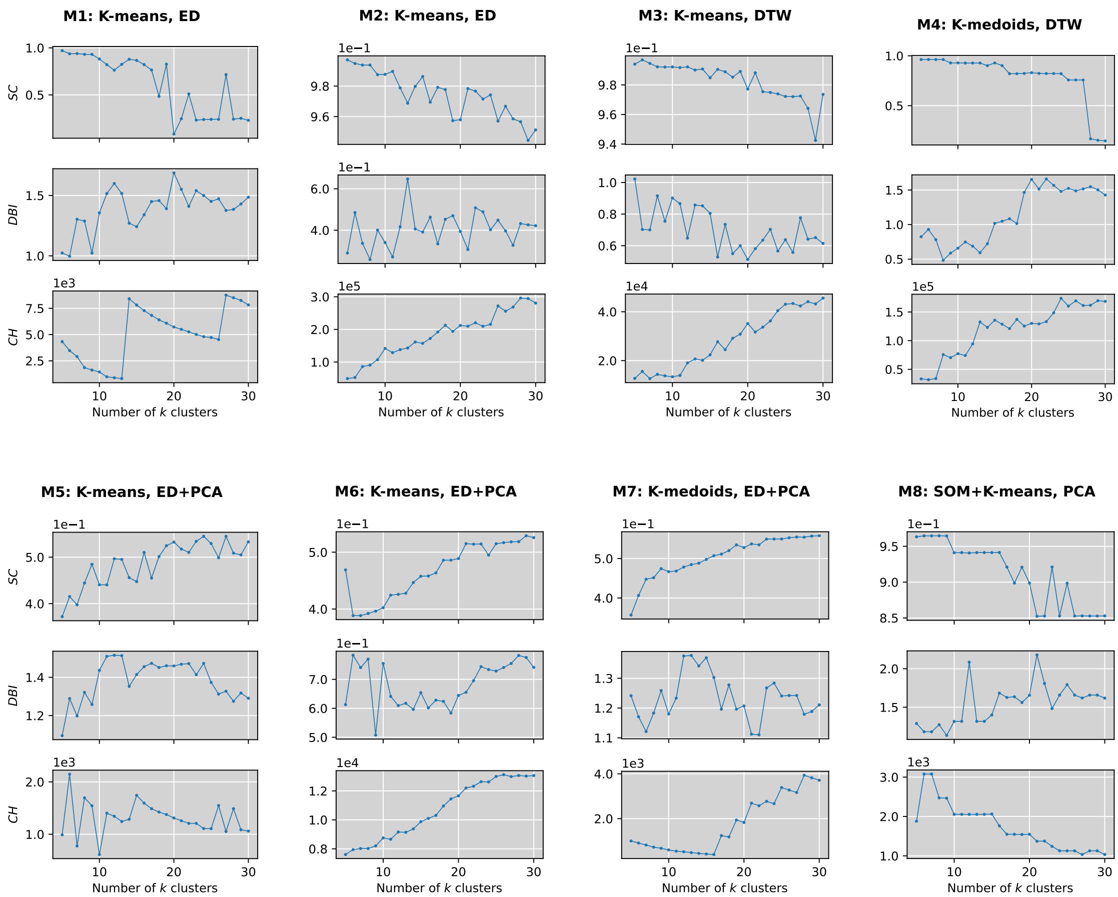


Figure 5.9: Overview of the internal validation metrics of the clusters derived from the Dataset II (1 hour). Three metrics were used to evaluate variations of clustering techniques and the number of chosen k clusters: the Silhouette Coefficient (SC), Davies-Bouldin index (DBI), and Calinski-Harabasz index (CH).

5.2.4 Shape-based clustering

The results for the shape-based $M2$ technique are presented in Table 5.4 and 5.5 The application of the technique produces a cluster profile with one dominant super-cluster representing the majority of the sampled serverless functions, and a large variety of smaller clusters, most of which represent a single assigned function.

The super-clusters $C0$ represent more than 99% of all samples, with an average invocation rate Ci_{avg} across all assigned serverless functions being no more than 4 invocations per minute. This characteristic is further reinforced by the periods of complete inactivity Ca_{min} of which samples belonging to $C0$ constitute 79% and 69.9% of total observations within super-clusters for the dataset I and II, respectively. In summary, while both super-clusters contain nearly all serverless functions, most of these functions have significant periods of inactivity.

Table 5.4: Summary of all clusters obtained by applying K-means-based $M2$ technique on Dataset I, showcasing the cluster size category, sample-count metrics, amplitude of invocation rate, and function activity describing cluster size, samples, invocation rate, activity and observed shape

Cluster	**Size**	**Samples**		**Invocation rate**			**Activity %**		
		Total	%	Ci_{min}	Ci_{max}	Ci_{avg}	Ca_{min}	Ca_{max}	Ca_{avg}
$C0$	Super-cluster	28634	99.482	0	26203	3.397	0.79	0.093	0.117
$C1$	Single-function	1	0.003	51196	75700	58049.397	0	0.001	0.999
$C2$	Scarce	2	0.007	4008	25613	15987.059	0	0.001	0.999
$C3$	Single-function	1	0.003	59414	111263	88522.003	0	0.001	0.999
$C4$	Single-function	1	0.003	0	151835	8435.328	0.831	0.001	0.168
$C5$	Scarce	103	0.358	0	86560	820.657	0.083	0.001	0.916
$C6$	Single-function	1	0.003	18689	56117	33991.543	0	0.001	0.999
$C7$	Single-function	1	0.003	281	33075	8650.631	0	0.001	0.999
$C8$	Scarce	3	0.01	6019	14027	10640.207	0	0.001	0.999
$C9$	Single-function	1	0.003	0	47226	3246.7	0.321	0.001	0.678
$C10$	Single-function	1	0.003	0	43505	6830.985	0.008	0.001	0.992
$C11$	Scarce	22	0.076	0	42284	2442.583	0.005	0.001	0.994
$C12$	Single-function	1	0.003	77	51286	7522.417	0	0.001	0.999
$C13$	Single-function	1	0.003	4198	27281	10264.147	0	0.001	0.999
$C14$	Scarce	2	0.007	0	12346	3773.064	0.126	0.001	0.873
$C15$	Single-function	1	0.003	1099	23780	15595.399	0	0.001	0.999
$C16$	Scarce	4	0.014	197	10661	5169.68	0	0.001	0.999
$C17$	Scarce	2	0.007	0	14721	1368.937	0.484	0.001	0.515
$C18$	Single-function	1	0.003	4546	48349	11627.749	0	0.001	0.999

The remaining serverless functions are split between two types of clusters. The larger number of clusters contain a single assigned sample, while the remaining clusters contain groups of samples no larger than 103 in total. The larger clusters are defined by having no cohesive cluster-wide pattern, even though many associated samples may be further subdivided into similarly-shaped sub-categories. At the same time, all single-function clusters reflect a particular pattern directly associated with the shape of the associated time series. Both types of clusters have a relatively large average invocation profile Ci_{avg} and maximas Ci_{max}, with most functions remaining consistently active with brief to no periods of inactivity Ca_{min}, while also not revisiting the maxima Ci_{max} more than once, which is indicated by a low and mostly shared ratio Ca_{max}. These characteristics indicate that the remaining clusters represent the non-static serverless functions with high rates of activity and spikes in invocation calls — significant enough to be considered as outliers due to the shape-based measures used in partitioning (as well as the low number of the assigned samples).

Table 5.5: Summary of all clusters obtained by applying k-means-based $M2$ technique on Dataset II, showcasing the cluster size category, sample-count metrics, amplitude of invocation rate, and function activity describing cluster size, samples, invocation rate, activity and observed shape

Cluster	Size	Samples		Invocation rate			Activity %		
		Total	%	Ci_{min}	Ci_{max}	Ci_{avg}	Ca_{min}	Ca_{max}	Ca_{avg}
$C0$	Super-cluster	19016	99.16	0	6259	3.553	0.699	0.184	0.117
$C1$	Single-function	1	0.005	53038	69730	57683.433	0	0.017	0.983
$C2$	Scarce	3	0.016	8455	12071	10561.567	0	0.017	0.983
$C3$	Single-function	1	0.005	26268	38160	33381.817	0	0.017	0.983
$C4$	Single-function	1	0.005	74892	85720	81335.733	0	0.017	0.983
$C5$	Single-function	1	0.005	0	151835	8815.667	0.883	0.017	0.1
$C6$	Scarce	11	0.057	838	7489	3886.423	0	0.018	0.982
$C7$	Single-function	1	0.005	8613	16817	13487.133	0	0.017	0.983
$C8$	Single-function	1	0.005	5402	48349	10585.733	0	0.017	0.983
$C9$	Scarce	32	0.167	0	4979	1579.267	0.003	0.017	0.981
$C10$	Single-function	1	0.005	0	38572	7796.183	0.2	0.017	0.783
$C11$	Single-function	1	0.005	1088	20134	5259.783	0	0.017	0.983
$C12$	Scarce	96	0.501	0	6477	557.287	0.055	0.021	0.924
$C13$	Single-function	1	0.005	91	12942	3568.583	0	0.017	0.983
$C14$	Single-function	1	0.005	0	18628	620.933	0.967	0.033	0
$C15$	Single-function	1	0.005	470	19058	1641.233	0	0.017	0.983
$C16$	Scarce	2	0.01	8011	17335	14953.583	0	0.017	0.983
$C17$	Scarce	2	0.01	1454	8576	6936.842	0	0.017	0.983
$C18$	Single-function	1	0.005	0	11711	1034.433	0.217	0.017	0.767
$C19$	Single-function	1	0.005	0	16127	925.233	0.8	0.017	0.183
$C20$	Scarce	2	0.01	0	5047	1333.8	0.608	0.017	0.375

5.2.5 Hybrid-based clustering

The hybrid based clustering resulted in a far longer computational time to adapt the data and select the actual features than expected, up to multiple days for the large dataset. As can be observed in table 5.2 method ID M7, which is k-medoids initiated with PAM and Euclidean distance, and took time to compute magnitudes longer than the others. For example, the 1 day dataset took 4 days and 7 hours to compute, and the 1 hour dataset took 1 day and 10.75 hours to compute.

Comparing random samples, 10 (or as many as possible), taken from each cluster we obtain the following information. The function distribution among the clusters in D1 appear uniform and D2 is more skewed towards clusters 3-5 (see figure A.18), this will impact the final patterns detected in the different clusters for both of the datasets.

Table 5.6 summarizes the random samples obtained from each cluster, summarizing the size, samples, invocation rate, and activity. From this one can see that there are a majority of small functions, some medium and scarce, and one single function cluster. From the analysis, 7 different shapes have been found over the 22 clusters that vary dependent on invocations and activity. The majority of clusters have activity that trends towards the minimum invocation rate, i.e. for all except cluster C10 the invocations tends to be 0. Very few functions also do not tend to fluctuate close towards their maximum, or around their activity average; only 3 clusters tend to have the majority of their activity towards its average (C5, C8 and C9). Compared to M2 and M8, all clusters tend towards the opposite, with their much lower average invocation rates, and appearing spikier (with more defined shapes) throughout.

Table 5.6: Summary of all clusters obtained by applying the hybrid based $M7$ technique on Dataset I.

Cluster	**Size**	**Samples**		**Invocation rate**			**Activity %**		
		Total	%	Ci_{min}	Ci_{max}	Ci_{avg}	Ca_{min}	Ca_{max}	Ca_{avg}
$C0$	Small	2101	7.299	0	14721	0.553	0.998	0.001	0.001
$C1$	Scarce	239	0.83	0	8911	0.101	0.999	0.001	0
$C2$	Single-function	1	0.003	0	151835	8435.328	0.831	0.001	0.168
$C3$	Small	2158	7.497	0	26203	1.272	0.863	0.007	0.13
$C4$	Small	448	1.556	0	688	0.012	0.995	0.001	0.005
$C5$	Small	759	2.637	0	10661	15.324	0.006	0.446	0.548
$C6$	Scarce	254	0.882	0	3	0.103	0.899	0.1	0.001
$C7$	Scarce	254	0.882	0	60	0.798	0.497	0.431	0.072
$C8$	Small	839	2.915	0	111263	508.476	0.233	0.007	0.76
$C9$	Small	1857	6.452	0	86560	64.546	0.352	0.006	0.642
$C10$	Small	826	2.87	1	60	3.335	0	1	0
$C11$	Small	923	3.207	0	60	0.02	0.983	0.017	0
$C12$	Medium	3033	10.537	0	3657	0.417	0.953	0.009	0.039
$C13$	Scarce	266	0.924	0	30	0.001	0.999	0.001	0
$C14$	Small	528	1.834	0	2	0.067	0.933	0.067	0
$C15$	Medium	3675	12.768	0	24843	0.69	0.971	0.002	0.027
$C16$	Small	1061	3.686	0	223	0.689	0.655	0.33	0.015
$C17$	Small	1409	4.895	0	126	0.254	0.797	0.2	0.003
$C18$	Small	2148	7.463	0	2706	0.497	0.898	0.048	0.055
$C19$	Scarce	239	0.83	0	16074	0.454	0.998	0.001	0.001
$C20$	Small	2651	9.21	0	9037	0.081	0.992	0.001	0.007
$C21$	Medium	3114	10.819	0	3984	1.073	0.655	0.162	0.184

For D2, the summary shown in table 5.7, the summary is very similar to D1. The majority of clusters still tend towards their minimum for their activity. This dataset results in 3 clusters, which have its included functions always staying on, and never having any invocations (C2, C8 and C13). The clustering finalizes with 9 distinct shapes for the 28 assigned clusters, all with varying activity and invocation rates. The average invocation rate trends towards the lower values, meaning that most of the functions do not have a high invocation count during the time frame. With some higher averages, however, these belong to the 3 aforementioned clusters which never return to 0 and they are either very small, or, a single-function cluster.

With 28 assigned clusters, is it expected to have some clusters that only include a small number of functions, and this is apparent in the summary. However, the spread of functions is even amongst the majority of clusters. The observed shapes for dataset II also tends towards having spikes in different forms and disjoint shapes (assigned to the majority of the clusters).

Table 5.7: Summary of all clusters obtained by applying the hybrid based M7 technique on Dataset II.

Cluster	Size	Samples		Invocation rate			Activity %		
		Total	%	Ci_{min}	Ci_{max}	Ci_{avg}	Ca_{min}	Ca_{max}	Ca_{avg}
$C0$	Small	951	4.959	0	441	0.186	0.932	0.067	0.001
$C1$	Single-function	1	0.005	0	151835	8815.667	0.883	0.017	0.1
$C2$	Single-function	1	0.005	74892	85720	81335.733	0	0.017	0.983
$C3$	Medium	2145	11.185	0	128	0.252	0.799	0.2	0.001
$C4$	Small	1462	7.624	1	200	4.245	0	1	0
$C5$	Medium	3248	16.937	0	6259	0.285	0.982	0.017	0.001
$C6$	Single-function	1	0.005	0	38572	7796.183	0.2	0.017	0.783
$C7$	Small	652	3.4	0	3114	0.479	0.962	0.017	0.021
$C8$	Single-function	1	0.005	53038	69730	57683.433	0	0.017	0.983
$C9$	Small	514	2.68	0	2848	2.084	0.857	0.033	0.109
$C10$	Small	310	1.617	0	580	15.734	0.067	0.428	0.505
$C11$	Small	527	2.748	0	103	1.143	0.509	0.464	0.027
$C12$	Small	515	2.686	0	322	1.092	0.749	0.198	0.053
$C13$	Scarce	4	0.021	1454	38160	17533.788	0	0.017	0.983
$C14$	Small	398	2.075	0	64	0.491	0.716	0.253	0.03
$C15$	Small	343	1.789	0	216	0.105	0.976	0.017	0.006
$C16$	Small	830	4.328	0	418	2.611	0.204	0.211	0.584
$C17$	Small	800	4.172	0	582	0.585	0.797	0.191	0.012
$C18$	Scarce	82	0.428	0	48349	2261.311	0.1	0.02	0.88
$C19$	Small	725	3.781	0	5	0.34	0.665	0.334	0.001
$C20$	Small	546	2.847	0	1660	86.189	0.038	0.029	0.933
$C21$	Small	582	3.035	0	2616	4.847	0.698	0.052	0.25
$C22$	Small	865	4.511	0	2073	5.276	0.392	0.07	0.538
$C23$	Small	799	4.166	0	119	0.206	0.888	0.104	0.008
$C24$	Small	988	5.152	0	722	0.497	0.898	0.059	0.043
$C25$	Small	946	4.933	0	1092	0.25	0.957	0.036	0.007
$C26$	Small	579	3.019	0	1400	0.551	0.947	0.017	0.036
$C27$	Small	362	1.888	0	2334	92.215	0.099	0.026	0.875

5.2.6 Model-based clustering

The results for the model-based $M8$ technique are presented in table 5.8. While the target k for the k-means component was set to 7 clusters, not all clusters contained function data from dataset I and II. This side-effect can be explained by the way the specificity of the self-organizing maps. Over the course of training epochs, the topography between the network nodes changes to reflect the dimensionality reduced time series that the network is being adjusted to. The $M8$ technique applies the resulting distance map to the k-means algorithm, clustering the network's topography directly. This process includes both the winning nodes that had the time series assigned to them, and regions that remained either unchanged and/or had no winning nodes. As such, the clusters produced by the k-means algorithm include both the data-dense nodes which can be back-traced to the original data (as reflected by the results in table 5.8), as well as the "empty" clusters representing the data-sparse regions, which for the purposes of this thesis were not considered for the analysis (and, as such, were discarded).

Table 5.8: Summary of all clusters obtained by applying the model-based $M8$ technique on Dataset I and II.

Cluster	Size	Samples		Invocation rate			Activity %		
		Total	%	Ci_{min}	Ci_{max}	Ci_{avg}	Ca_{min}	Ca_{max}	Ca_{avg}
$C0$	Single-function	1	0.003	0	16074	104.508	0.985	0.001	0.015
$C1$	Super-cluster	28454	98.857	0	24843	1.964	0.794	0.093	0.112
$C2$	Small	326	1.133	0	151835	1616.222	0.096	0.001	0.904
$C3$	Scarce	2	0.007	0	26203	124.28	0.909	0.001	0.09

(a) Dataset I

Cluster	Size	Samples		Invocation rate			Activity %		
		Total	%	Ci_{min}	Ci_{max}	Ci_{avg}	Ca_{min}	Ca_{max}	Ca_{avg}
$C0$	Scarce	30	0.156	0	19058	2180.602	0.021	0.017	0.962
$C1$	Super-cluster	18895	98.529	0	3114	2.48	0.702	0.185	0.113
$C2$	Scarce	188	0.98	0	18628	272.285	0.115	0.023	0.862
$C3$	Scarce	19	0.099	0	151835	16489.5	0.057	0.017	0.926
$C4$	Scarce	45	0.235	0	16127	929.82	0.154	0.021	0.825

(b) Dataset II

The overall sample distribution between clusters is comparable to the $M2$ technique, with the dominant super-cluster reflecting 98% of time series when applied to both datasets. With the number of sample-dense clusters being lower than other techniques, only one instance of a single-function cluster can be observed for dataset I. Additionally, nearly all clusters possess a disjointed pattern shape, a side-effect from multiple dissimilar time series being grouped up into the same cluster. The only outlier clusters that have an observable pattern are $C0$ and $C3$.

5.3 Systematic analysis

5.3.1 Common patterns

The analysis of results resolved in multiple patterns existing in the datasets which, depending on the used clustering technique, were easier or more difficult to describe. In order to classify the results, the overall centroid shape of the cluster's time series was used to establish a set of terms used in the systematic analysis. The common classification terms are defined as follows:

- **Constant**: this pattern is composed of a static invocation load above dormant rate that resembles a flat line stretching throughout the entire time series.
- **Disjointed**: this pattern is designed for cases where several sample functions (around 10 samples) may share similar characteristics within the cluster, however, a consensus between the samples can not be determined without decomposing the cluster into multiple sub-clusters.
- **Gradient**: a workload pattern which displays such a high degree of volatility in invocation calls that the visualized time series resemble a gradient. This pattern is especially noticeable for the Dataset I samples, since invocation call

spikes that cover 5 minute intervals would resemble a regular timer pattern on a 60-minute scale, i.e. Dataset II.

- **Mean adjacent**: the pattern of invocation calls closely follows the time series centroid line through the entire range of observations. This attribute varies from time-series to time-series, ranging in its uniformity, but overall, little to no deviations are commonly present.
- **Mountain**: this pattern is defined by the low number of invocation calls that follow a semi-stable flat-curve, such as flat-lined dormant functions or functions with a number of invocation calls that don't change, which at some point rapidly increases for a bearing of time — forming a peak or a plateau of high activity, before descending back to a lower state.
- **Mountains & valleys**: this pattern contains multiple mountain patterns throughout the range of observations that are interspersed with 'valleys' of lower activity.
- **Multiple spikes**: this pattern emerges when time-series are composed of multiple spikes with negligible number of other patterns in-between, i.e. mountains, valleys, constants, etc.
- **Spike**: this pattern resembles a mountain pattern with a key difference of how fast the peak of activity is reached. While the mountain pattern takes 5+ minutes to reach the peak, a spike in activity happens within 1-5 minutes.
- **Pulse**: a subset of multiple spikes with the key difference of possessing a uniform pattern of evenly spaced dormant states between following invocation spike(s).
- **Random**: this designation is used for an undecidable pattern, where sampled functions do not share much similarity other than the changes in invocation rates over time.
- **Sinusoidal**: this pattern exhibits a shape consistent with a sine wave curve. This includes samples with a high amplitude of invocation calls, changing in a volatile, rapid manner, while the centroid maintains the sinusoidal shape, e.g. $M7$ with Dataset I.

5.3.1.1 Shape-based approach

Table 5.9 describes the summary of the common shapes described during the systematic analysis done for dataset I and II, using K-means and euclidean distance.
Dataset I results in a majority of the identified shapes, and is very concentrated towards mean adjacent and disjointed shapes. This, especially the disjointed shapes, indicates that the clustering result is not optimal but still shows potential, and observing table 5.4 one can see that the results include a super-cluster (which samples over 99% of the total samples). This will create a lot of single function clusters, which subsequently creates a very sparse clustering result only describing a couple of functions — and not the entire dataset. Furthermore, disjointed is one of the largest shapes, which is one of the less descriptive shapes, where multiple shapes are included in the entire cluster, and thus results in a reduced or unclear understanding of the pattern the cluster creates.
Dataset II is more spread over the identified shapes, with disjointed as the largest identified shape. This set also contains a disjointed super-cluster which, as men-

tioned for Dataset I, indicates a suboptimal clustering, but some were more spread. This, and that the number of samples in the super-cluster being smaller than for DI's, indicates that the hour set was better clustered than the day set for this algorithm.

Table 5.9: Overview of the patterns identified for the shape-based $M2$ clustering method.

Shape	**Dataset I**		**Dataset II**	
	Count	Cluster ID	Count	Cluster ID
Constant line	-	-	-	-
Disjointed	4	C0, C5, C15, C16	6	C0, C2, C6, C9, C12, C17
Gradient	-	-	-	-
Mean adjacent	7	C1, C2, C3, C6, C8, C10, C18	1	C3
Mountain	3	C7, C14, C17	4	C5, C7, C8, C15
Mountains & valleys	-	-	3	C11, C13, C19
Multiple spikes	3	C4, C9, C15	3	C1, C10, C18
Pulse	-	-	2	C14, C20
Random	1	C12	1	C4
Sinusoidal	1	C13	-	-
Spike	-	-	-	-
Total	19	19	21	21

5.3.1.2 Hybrid-based approach

Table 5.10 shows a summary of the common shapes discovered through the systematic analysis for both datasets for k-medoids, initiated with PAM and euclidean distance.

For the 1 day dataset one can see a clear majority, 64%, of patterns including spikes in some matter. After these the very frequent gradients with 4 clusters assigned, and disjointed shapes with 3 clusters assigned. Finally the constant and sinusoidal with one cluster each. We can clearly see that the number of clusters impacted on how the clusters were fitted, especially with all different spikes that in many cases could be merged together.

For the 1 hour dataset, spikes also are the majority, 54% of all clusters. In this set of clusters, there were even more difficult samples to combine into a cluster and see a distinct pattern, therefore 6 clusters that were sorted into disjointed. The patterns for the hour pattern are more distinct and clear due to the much shorter time frame, thus the mountain and mountains & valley pattern more readily occur (which could also resemble a broad spike or saw pattern). There also exists a sinusoidal pattern, however it is much more spiky than an actual sinus wave. Furthermore, there exists a mean adjacent shape, which could be similar to the sinus wave. However, the revolving is not done in a sinusoidal pattern — more in a heartbeat or mountain/upside-down mountain pattern. Finally, a constant line also appeared for the 1 hour pattern, however with different amplitudes between the samples.

Table 5.10: Overview of the patterns identified for the hybrid-based $M7$ clustering method.

Shape	Dataset I		Dataset II	
	Count	Cluster ID	Count	Cluster ID
Constant line	-	-	-	-
Disjointed	3	C3, C5, C9	6	C9, C10, C16, C18, C22, C27
Gradient	4	C7, C16, C17, C21	-	-
Mean adjacent	-	-	1	C13
Mountain	-	-	2	C1, C20
Mountains & valleys	-	-	2	C2, C6
Multiple spikes	3	C2, C12, C14	7	C12, C14, C17, C19, C21, C24, C25
Pulse	3	C6, C11, C18	3	C0, C3, C23
Random	-	-	-	-
Sinusoidal	1	C8	1	C8
Spike	7	C0, C1, C4, C13, C15, C19, C20	5	C5, C7, C11, C15, C26
Total	22	22	28	28

5.3.1.3 Model-based approach

Table 5.11 describes the summary of the common shapes identified during the systematic analysis for Dataset I and II using the SOM + K-means algorithm, euclidean similarity measure and initialized using PCA weights.

The common shapes for both datasets are very sparse, due to the small chosen cluster size. The 1 day dataset shows some differences in identified shapes, with a spike and multiple spikes, however, since they are defined by a single-function and a scarce sample size it is not truly indicative of the shape of the different clusters. The datasets show similar distribution to M2, with super-clusters containing over 98% of the total samples. The majority of shapes resulting in disjointed shapes shows that the clustering process did not result in optimal clusters — even though the evaluation had pointed towards this number of clusters.

Table 5.11: Overview of the patterns identified for the model-based $M8$ clustering method.

Shape	Dataset I		Dataset II	
	Count	Cluster ID	Count	Cluster ID
Constant line	-	-	-	-
Constant line	-	-	-	-
Disjointed	2	C1, C2	5	C0, C1, C2, C3, C4
Gradient	-	-	-	-
Mean adjacent	-	-	-	-
Mountain	-	-	-	-
Mountains & valleys	-	-	-	-
Multiple spikes	1	C3	-	-
Pulse	-	-	-	-
Random	-	-	-	-
Sinusoidal	-	-	-	-
Spike	1	C0	-	-
Total	4	4	5	5

5.3.2 Workload characterization

Figure 5.10 depicts all the defined shapes found during the Systematic Analysis.
For the disjointed shape, it can be loosely defined as containing multiple defined shapes in one cluster — which renders a consensus impossible. Multiple clusters are plotted into one figure in order to show this inconsistency more clearly.
The constant shape is, as the name describes, a line of constant values over the entire time frame. The activity stays the same for the duration however the invocation count can differ between functions and it does not define the shape.
The gradient shape, which only can be seen for the day dataset (Dataset I), shows a high frequency of invocations, and during the longer period the pattern resembles a gradient. This shape in the 60-minute time frame would instead resemble the pulse shape (and vice versa — if the pulse shape would continue for an entire day). The invocation count has to explicitly stay on 2 values, often 0 and a max value that it fluctuates between.
The mean adjacent shape describes a pattern that follows a clear middle point with different invocation counts that continuously return to the same point. For this shape to be applied, the mean must be clearly defined, and not vary significantly during the specified period.
Mountain is a shape which is similar to a spike, with longer duration of invocations and often surrounded by either no, or very few, invocations. As seen in the figure, the function has a large mountain of invocations for 5 minutes and with much fewer invocations around it. In general, a mountain is defined as a short burst of invocations in a otherwise stationary function.
The next shape is mountains & valleys, which is a pattern of multiple mountains contained in a single function. The mountains show clearly, and then the invocation

count lessens, but does not necessarily go down to stationarity and, then goes back up to depict the signature mountain shape again.

Multiple spikes is a shape where the pattern spikes multiple times during the time frame, with a non-uniform pattern. It does not necessarily mean a single time spike, but could also stretch over a short period of time. The invocation count could also differ during this time frame.

The pulse shape is closely related to multiple spikes, however, with the difference that the pulse is uniform and the spikes similarly having the same amplitude throughout the entire time frame. The amplitude is not defining for the shape, and could differ between functions, and still be defined as a pulse.

Random is a shape where a clear pattern not could be found. However, were it to be split into multiple parts, it perhaps could result in defined patterns. The function does not act in any way that is coherent to any pattern, and seems to randomly invoke the function at any time, with random frequencies.

The sinusoidal shape follows the sinus curve, the example has some spikes with higher invocations than what a perfect sinus curve depicts, but the general shape is still sinusoidal.

The spike is self explanatory, it follows just a single spike with zeroes around it for the entire time frame. Where the spike occurs or the count does not matter.

The disjointed shape is a shape where the functions in the cluster does not share a consensus between each other, and thus the cluster can not be described with a specific shape. The included functions could, themselves, be defined with the described shapes as seen in the figure, where multiple mean adjacent or constant lines can be seen, followed by a mountain and mountains & valleys. The disjointed shape was observed more often in the shape-based and model-based methods, which also includes super-clusters, which leads to the inference that the remaining clusters would have been more difficult for the algorithm to cluster into defined patterns.

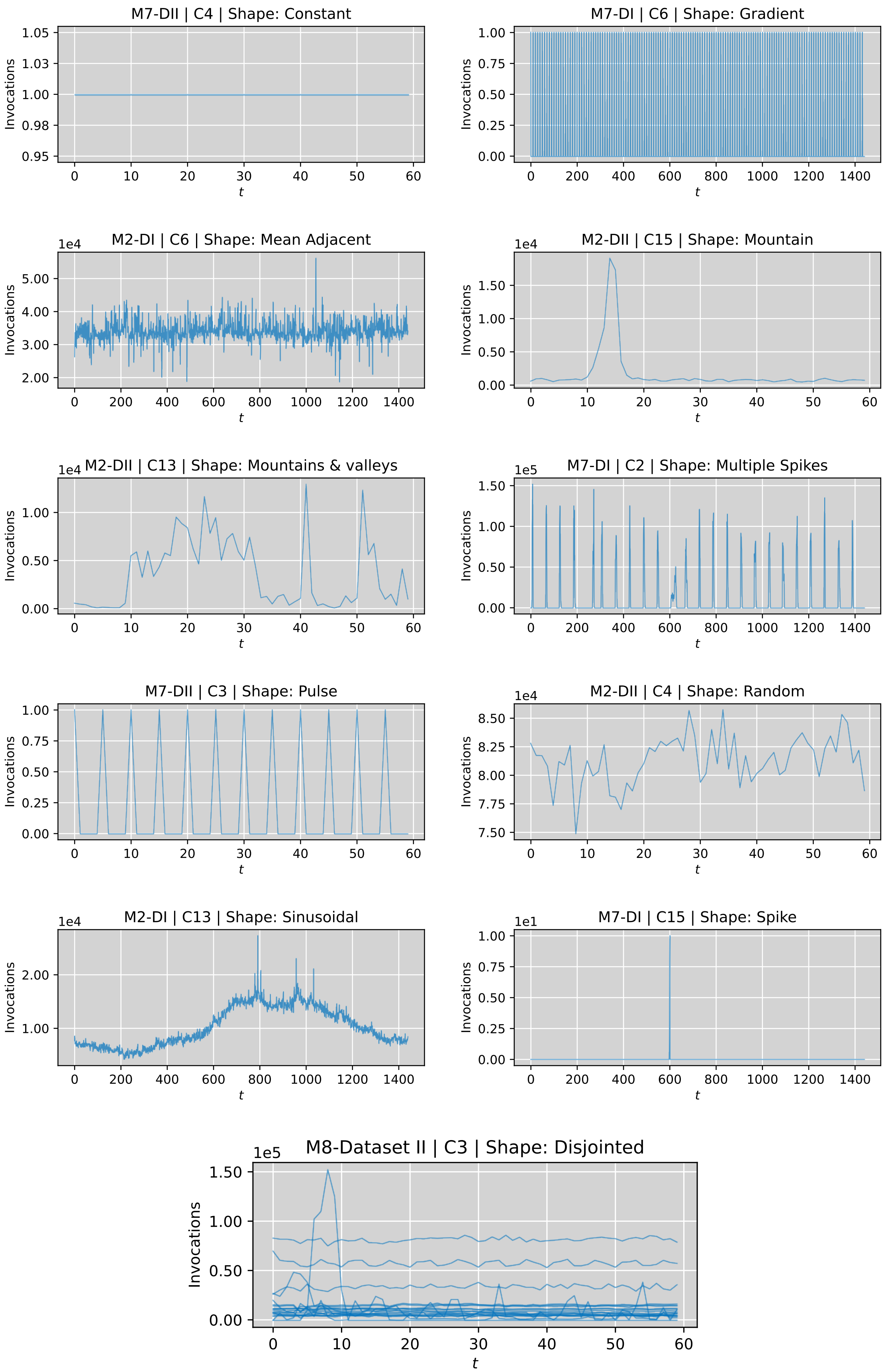


Figure 5.10: Figures depicting the different shaped defined during the Systematic Analysis.

5.3.3 Anomalies

During the data processing and clustering, different anomalies appeared that have made an impact on the results. One anomaly that is mentioned by the providers of the dataset is that during the logging process some counts were logged in the bucket before their the actual time, and thus a higher count than expected was shown. Figure 5.11 displays an example of this behaviour, where one can see two instances of a constant line having the fault, and shows a pattern where the invocation count goes from a constant count of 1, and then up to 2, and back down to 0 for the minute after — and then back to a constant count for the time's remainder.

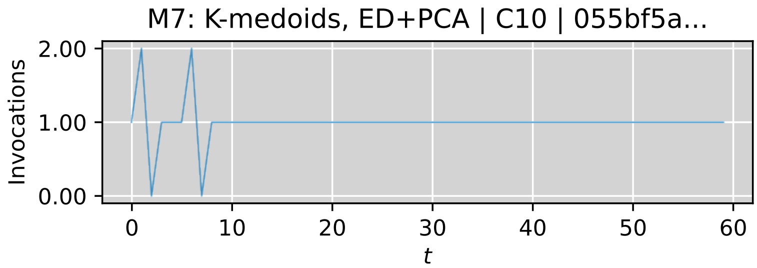


Figure 5.11: Example of a logging fault where a count is logged for the wrong minute.

Another anomaly, shown in figure 5.12, belongs to an HTTP-based function, wherein a spike in usage exponentially grew from a constant, timed value of 1 up to an average of around 300000 for a 20 minute period, only to return back to an inactive state for the remained of the two weeks. Since the principal dataset's functions are anonymous and no more information is available, the source of this sudden change in behavior remains unknown.

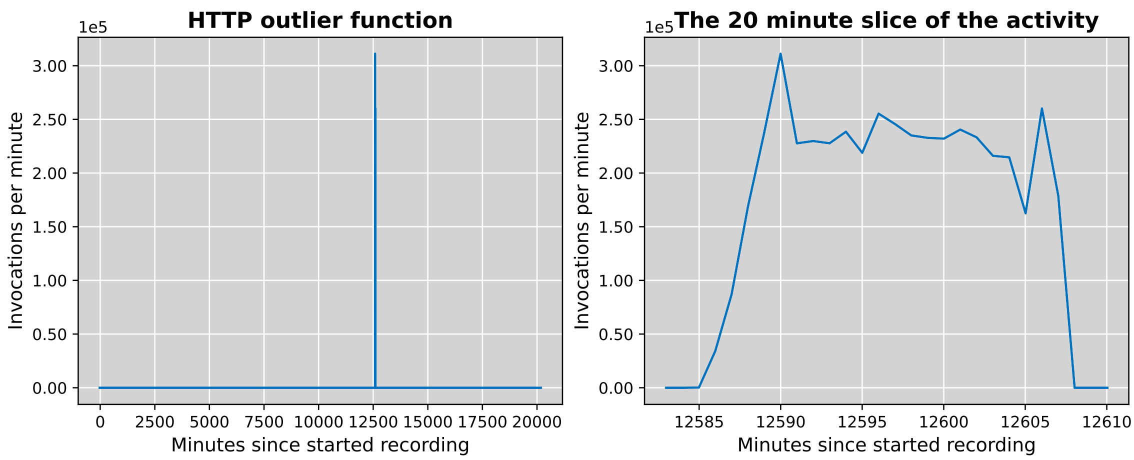


Figure 5.12: Overview of an anomalous spike in the activity detected for an HTTP-trigger based invocation function

6

Discussion

Based on the objectives established for this study, this chapter reviews the results presented in Chapter 5, covering each research question listed in Section 1.2 under Section 6.1. Furthermore, Section 6.2 reflects upon the challenges experienced throughout the study that lie outside of the aforementioned research questions, as well as presents a perspective on the lessons learned while conducting the research phases. Lastly, Section 6.3 describes the threats of validity to the results and findings discovered throughout the study, and details a set of measures that were implemented to minimize the negative effects on the conclusions.

6.1 Research questions

6.1.1 RQ1: How can existing clustering techniques be applied to Serverless workload pattern identification?

Serverless workload patterns consists of multiple time series and analyses using statistics and similar methods can be applied without any modification. Time series can be difficult to analyse and the method applied in this thesis may not be the optimal and should be studied further. Due to the size of the data, other methods may work better with larger amount of data than the method conducted throughout this thesis. This could yield in better preprocessing and understanding about the dataset and result in better clustering results and characteristics. Further studies using datasets equal in size to the dataset used in this paper have not been found and thus any comparisons between methodologies could not be conducted. However, the methods used in this thesis for data analysis can be used for any dataset, not exclusively time series.

Existing machine learning algorithms need to be adapted to work with time-series. Multiple libraries have been created and open-sourced for this specific purpose, and seem to work without faults. This, however, combined with the fact that these algorithms may not be optimized make it still an experimental feature that may not represent the full truth of the matter. The methodology used in this thesis was very limited by the size of the dataset, which made the use of more sophisticated algorithms for comparison and clustering not applicable (due to unobtainable computational complexity). This severely impacted the clustering negatively, possibly losing clusters and patterns that could have arisen otherwise.

The baseline model, K-means with Euclidean distance, appeared to be a suitable baseline to work with, with due consideration given to the scope and size of the data. Though inaccurate and simple, it was the best possible fit for the entirety of the data. As observed through the entirety of the work, the dataset was too large for more sophisticated algorithms which could have yielded more accurate and informative results. Another factor that could yield alternative results, or skew it towards a not as informative or realistic result, is whether the the dataset includes all trigger types or not. Clustering and observing the trigger types by themselves would result in more precise characteristics that, consolidating them all together would not; this is due to some trigger types being more prominent in the data.

6.1.2 RQ2: How can applicability of workload pattern identification techniques be empirically assessed?

The techniques that were implemented after preliminary tests did not impose any challenges outside of what was expected according Section 3.2 - large amount of input data putting high time, computational, and memory demands on the test environment. This resulted in some techniques taking days to compute, with some distance measures - such as DTW - taking unreasonable amount of time to compute for the original, principal data set. This was one of the reasons for limiting the scope of clustered time series to Dataset I (1 day) and II (1 hour), as well as implementing the hybrid-approach technique, combining the dimensionality reduction properties of the extracted features and PCA. While the automated feature selection process did take nearly a day to process the input subsets — the used time was incomparable to weeks-worth of potential calculations for certain techniques and DTW.

The reduced scope of used observations does introduce generalizability and validity concerns as only limited number of patterns were used in clustering. While the hybrid-approach $M7$ technique had several viable clusters, the patterns may be more representative of the selected slices (i.e. period of the week - 1s day; and period of the day - 1st hour) rather than generalize towards the entire time frame of 2 weeks. That said, patterns over multiple weeks may not be as interesting when it comes to workload analysis as a day or an hour pattern, where smaller changes may accumulate and incur significant affects over the cumulative, long-term workloads; and thus, after the two weeks made more impactful changes than what would been made for a shape assigned over the entire time.

As mentioned, PCA, though loosing some of the information after the chosen threshold, was a necessary step to decrease computational time for clustering which showed potential for being used as an initiator for multiple algorithms. This method show improvements in computational times over all algorithms; due to its lower dimensionality, however, it does not show the entire picture of the occurrence due to the loss of information. One clear indication that PCA looses some information is when observing the internal evaluation metrics, especially for $M7$ PAM, and PCA where the silhouette score is very low. Furthermore the score is much lower for M5, K-Means with PCA compared to the others without PCA. The Calinski-Harabasz score is also much lower for the algorithms using PCA compared to without PCA, which follows that PCA is reducing the dimensionality, and thus the Calinski-Harabazs score is lower. This suggests that PCA is a valid method for time series clustering without a great loss of information.

The computational time for many algorithms has been magnitudes longer than what many other studies show, partly depending on the implementation of the algorithm by the open source libraries used but also on the size of the dataset itself. The slowest clustering method compared to the fastest differ by 5-magnitudes of order, which does not include the time for the pre-processing done for the different methods. Thus for example for PAM, PCA ($M7$) and SOM ($M8$), which for $M7$ the feature extraction part before applying PCA takes a couple of days to finish, and the optimization for SOM hyperparameters takes about 10 minutes to finish. Computational time depends majorly on the hardware. The machines used during this thesis were both relatively powerful from the onset but were later upgraded; exclusively, the RAM was upgraded for both machines. For some operations during the thesis process, the upgraded RAM was a necessity to have, due to the high computational cost of some algorithms. This obstacle was occurring for many used algorithms until the scope of used data was scaled down to Dataset I and II; and even further through dimensionality reduction offered by PCA, allowing to use certain algorithms (especially k-medoids) without encountering RAM-related problems.
In spite of this result, studies show more interesting results using the same techniques. Thus these techniques were considered due to the extent they had been studied and used, however, not in context of this domain and the data size. The time and space complexity of all techniques were considered, and that the algorithms would take time was known; however due to the simplicity of the techniques, perhaps another technique or algorithm would have been a more apt choice. Compared to the baseline, it is still interesting to have a more simple technique to compare with, only using a more capable technique for the final clustering iteration and possible characterization. All these techniques are possible, and stumpy was only briefly glanced over in this thesis (but is suspected that it would work in the future). However, all techniques for both preparation and clustering showed no problem in completing and gathering some results. Only the computational factor has to be taken into consideration before using the entire Azure dataset (or another dataset of a similar size).

6.1.3 RQ3 What workload characterizations can be derived from the Serverless function invocation traces?

The systematic analysis yielded multiple patterns that could be discerned from the different clusters, however, the visual inspection of only a maximum of 10 samples does not convey the highest accuracy or the complete truth of the different shapes that could emerge in each cluster. Furthermore, the disjointed shape is an arbitrary shape (with multiple defined shapes included), and thus yielded the common shape through the samples undecidable, and in turn, being uncertain. The process of identifying the different common patterns included only visual inspection, which is not the most accurate form of analysis, but valid for beginning the work of defining common patterns. Due to time constraints, visual inspection had to be the only method of identifying patters, which definitely lowered the accuracy of finding common patterns that resonated well through all methods and clusters, and not only for the chosen methods or any general method chosen.

Cluster validation was based on some methods to evaluate the quality of the obtained clusters. The chosen evaluation methods were studied and used heavily, and did gain insight towards the number of optimal clusters. However, no method is perfect, and more methods could have been used to further understand the clustering results. Further evaluation could yield different shapes than what had been currently defined, due to the difference in optimal clusters. However, multiple shapes defined in this thesis are common for any data collected (such as spikes and constant lines), and could be generalized towards additional providers of cloud computing.

The defined shapes show multiple different characteristics that could be traced back to the included activity rates, general behaviour, and trigger type. Shapes such as constant, gradient, multiple spikes, pulse, and spike could all be linked to timers and HTTP requests with repeatable behaviour with uniform patterns; but not exclusively, since all clusters may include any trigger type with equal likelihood. However, trigger types such as timer tend to be clustered similarly due to the distinct behaviour inherent of the type. Deduction of this manner are difficult to draw for all shapes, however, since the behaviour of a certain trigger type or activity rate for a function is not readily known.

The fact that the majority of functions appear under the disjointed shape indicates that the optimal number of clusters were not found, and overlap between the clusters is apparent. This is most readily apparent for M5 and M7, where the silhouette score is low when compared to the others. This is due to the PCA method yielding a uniform sample size for the clusters, however, a downside to the uniformity is that the clusters include functions that are similarly shaped and thus samples that are similar could be clustered into different clusters instead of being clustered together. This will reduce the silhouette score, since samples are similar relative to the boundaries of the clusters to which they were assigned.

The HTTP outlier anomaly found, where the invocation exponentially grew, had no discernible reason for why it occurred, thus only speculations can be made. One possible explanation is a distributed denial-of-service (DDoS) attack attempt in which Azure tried to protect the functions, but the length of the "attack" renders this explanation not an extremely likely candidate for the cause. In this case, the lack of information about each function renders a concrete and verifiable explanation for its behavior impossible.

The other anomaly is only a fault during logging, where the invocation is logged for the wrong minute, which would definitely be clustered differently than the constant shape it would have had without the logging fault. The extent of this fault is not known nor was it investigated to the fullest extent, but it made a difference that not is to be neglected during this final consideration.

The findings of this thesis define a beginning of clustering and characterizing function invocation traces derived from cloud computing providers, establishing a baseline of what is and is not possible for a large amount of data using personal computers.

Considering the providers point of view, this thesis provides insight on possible directions to take considering the shapes found. However, it should be noted that all shapes may not be properly found for all applications if studied separately. Constant, mean adjacent, pulse and sinusoidal (considering it is known that it behaves in a matter such as day and night cycles) are most probable to occur for all applications, thus could be used for experiments and tests for new policies regarding, for example, cold starts and resource management by being able to predict when functions should warmed or pre-allocating resources.

Developers using the service provided by the providers and the information from this thesis, considering FaaS only costs when a function is run and the resources used, they should develop functions that are as efficient as possible with resources. Developing efficient functions which only evaluate one action and preferably executing in a pulse like matter to keep the resources assigned, this will ensure quick execution time since cold starts are avoided and resources more efficiently are allocated.

The shapes that provide insight for both developers and providers, such as constant and pulse, could be generalized to all different providers. However that could not be said for all shapes, due to how hardware and software policies are implemented and what physical hardware the different providers use. Different providers could have different implementations and triggers for functions which may behave differently than Azure. A lot of shapes, especially constant, mean adjacent, pulse and sinusoidal have great possibility of occurring for other providers since many of them resemble a repeating pattern, such as sinusoidal day and night.

6.2 Lessons learned

During the implementation of the algorithms, another library was found called stumpy [86], which includes an implementation of matrix profiles. Matrix profiles is another implementation of how to characterize time-series, and in a more time consuming way compared to the techniques used in this thesis. According to the documentation for the entire NaN dataset, which is around 581,000,000 data points, the matrix profile will calculate in about a week without the implementation time for the environment used. This would have sped up the implementation time in this thesis by a significant factor, however, the time frame did not allow for a transition to use it instead. If the entire dataset were to be used with the environment used for this thesis, even matrix profiles would be impossible to compute, as the only way to do so would be to use 8 or 16 NVIDIA Tesla V100 GPU cores (which are not readily accessible for the average person). Stumpy also features multiple interesting functions, such as snippets, which are smaller subsets of the larger set which similarly describes it, which could then be used for clustering or other algorithms (which are more computationally heavy). Furthermore, the usage of anomaly and novelty discovery algorithms to analyse different anomalies and features in the set could yield interesting, or alternate results than what were discovered in this thesis.

6.3 Threats to validity

We discuss potential threats to validity and how the we will mitigate the treats for internal validity, external validity and reproducibility.

6.3.1 Internal validity

The implementation of the techniques were made after thoroughly studying similar implementations. This was to mitigate the threat that the knowledge gained from the data preparation stage would bias the implementation towards any results found. Another threat to the validity is that, due to the shear size of the data, only one of the trigger groups was focused upon for optimising the code (which may have left some information or performance(s) hidden). This is an unfortunate side effect of the dataset's size, since the size rendered us unable to test other triggers efficiently and without taking multiple days for calculations. Another threat is the reduced size of the introduced datasets, where patterns were limited to the smaller set and some important patterns could been emitted.

6.3.2 External validity

Our study is made for clustering function invocation traces, especially from Azure where the data is made public from. Generalizability of the study is extremely difficult due to the fact that other providers have not made their data publicly available for study. The code base could be used for clustering from other providers, provided that the data is analysed before, and some parts of the implementation would be modified appropriately, for example, if there would exist differing or alternate fields. The mentions of different providers in the results is to address the generalizability of the method and results, however it can not be fully implemented nor provided here. Furthermore, limiting the dataset to smaller subsets introduce even less generalizability.

7
Conclusion

This thesis presents the findings from clustering and characterizing function invocation traces originating from Microsoft Azure. Analyzing the dataset and preparing it for clustering with existing techniques and establishing a baseline algorithm to be used. The data was split into multiple time scopes and the baseline implemented using K-Means with Euclidean distance which showed promising results. Shape-, model- and hybrid-based clustering techniques with different initializers were implemented to ensure validity and variation. The algorithms were evaluated using Calinski-Harabasz, Davies-Bouldin and the Silhouette indices to find the optimal number of clusters for each algorithm, ensure valid results and to identify any issues. We further identify workload characterizations using samples and visualization, and the invocation activity and rate from the derived clusters, describing and visualizing common patterns that arise. We find that eleven common patterns arise from the selected methods, ranging from constants, pulses to sinusoidal patterns, all with varying invocation counts and activity rate.

7.1 Future work

To further expand on this topic, individual clusters on the different trigger would be beneficial and interesting, furthermore analysing multiple time slices as some parts of this thesis have done. This would result in an improved understanding of function invocation traces in FaaS, as well as further evolution of the techniques used for gaining performance and efficiency.
Another interesting take on this thesis is to use stumpy and matrix profiles instead of basic K-means and K-medoids algorithms, this would result in another perspective of the FaaS invocation traces and its anomalies in the data. This could contribute to the fields of FaaS and cloud computing by means of offering additional insights into which tool would be best (or least) suited for certain types of analysis.
Lastly, forthcoming work on this subject could include using real-world data that had not been subject to anonymization. That is to say, working with datasets where the actual dates and times were present. With their inclusion, it could mean working with specific periods (for example, holidays or weekends) where the workload and usage may be expected to deviate from the standard operating norm. Therefore, it follows that a potential benefit which could be yielded from working with this type of real-world data would be in revealing patterns (or other characteristics) worthy of note. Thus, ensuing insights may also have practical applications, or utility, as well (depending on the subject or domain).

Bibliography

[1] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, “The rise of serverless computing,” *Communications of the ACM*, vol. 62, pp. 44–54, 12 Nov. 2019, ISSN: 0001-0782. DOI: 10.1145/3368454.

[2] G. C. Fox, V. Ishakian, V. Muthusamy, and A. Slominski, “Status of serverless computing and function-as-a-service(faas) in industry and research,” *ArXiv*, vol. abs/1708.08028, 2017.

[3] T. Lynn, P. Rosati, A. Lejeune, and V. Emeakaroha, “A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms,” IEEE, Dec. 2017, pp. 162–169, ISBN: 978-1-5386-0692-6. DOI: 10.1109/CloudCom.2017.15.

[4] I. Baldini, P. Castro, K. Chang, *et al.*, *Serverless computing: Current trends and open problems*, 2017. DOI: 10.1007/978-981-10-5026-8_1.

[5] J. M. Hellerstein, J. Faleiro, J. E. Gonzalez, *et al.*, “Serverless computing: One step forward, two steps back,” Dec. 2018.

[6] A. Mohan, H. Sane, K. Doshi, S. Edupuganti, N. Nayak, and V. Sukhomlinov, “Agile cold starts for scalable serverless,” USENIX Association, 2019, p. 21.

[7] J. Manner, M. Endreß, T. Heckel, and G. Wirtz, “Cold start influencing factors in function as a service,” IEEE, Dec. 2018, pp. 181–188, ISBN: 978-1-7281-0359-4. DOI: 10.1109/UCC-Companion.2018.00054.

[8] I. Pavlov and V. Vestlund, *Serverless workload characteristics*, 2022. [Online]. Available: https://github.com/Archilir/serverless-workload-characteristics.

[9] T. Kluyver, B. Ragan-Kelley, F. Pérez, *et al.*, “Jupyter notebooks – a publishing format for reproducible computational workflows,” in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt, Eds., IOS Press, 2016, pp. 87–90.

[10] “What is iaas? infrastructure as a service: Microsoft azure,” Microsoft. (2022), [Online]. Available: https://azure.microsoft.com/en-us/overview/what-is-iaas/ (visited on 05/18/2022).

[11] “What is paas? platform as a service: Microsoft azure,” Microsoft. (2022), [Online]. Available: https://azure.microsoft.com/en-us/overview/what-is-paas/ (visited on 05/18/2022).

[12] D. Ma, “The business model of "software-as-a-service",” IEEE, 2007, pp. 701–702, ISBN: 0-7695-2925-9. DOI: 10.1109/SCC.2007.118.

[13] “What is saas? software as a service: Microsoft azure,” Microsoft. (2022), [Online]. Available: https://azure.microsoft.com/en-us/overview/what-is-saas/ (visited on 05/18/2022).

[14] K. Figiela, A. Gajek, A. Zima, B. Obrok, and M. Malawski, "Performance evaluation of heterogeneous cloud functions," *Concurrency and Computation: Practice and Experience*, vol. 30, Aug. 2018. DOI: 10.1002/cpe.4792.

[15] J. Scheuner and P. Leitner, "Function-as-a-service performance evaluation: A multivocal literature review," *Journal of Systems and Software*, vol. 170, p. 110 708, Dec. 2020, ISSN: 01641212. DOI: 10.1016/j.jss.2020.110708.

[16] "Faas metrics," OpenTelemetry. (2022), [Online]. Available: https://opentelemetry.io/docs/reference/specification/metrics/semantic_conventions/faas-metrics/ (visited on 07/28/2022).

[17] M. Shahrad, R. Fonseca, I. Goiri, *et al.*, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," USENIX Association, Jul. 2020, pp. 205–218, ISBN: 978-1-939133-14-4. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/shahrad.

[18] S. Eismann, J. Scheuner, E. V. Eyk, *et al.*, "The state of serverless applications: Collection, characterization, and community consensus," *IEEE Transactions on Software Engineering*, pp. 1–1, 2021, ISSN: 0098-5589. DOI: 10.1109/TSE.2021.3113940.

[19] "Azure functions trace 2019," Mohammad Shahrad et al. (2019), [Online]. Available: https://github.com/Azure/AzurePublicDataset/blob/master/AzureFunctionsDataset2019.md (visited on 12/15/2021).

[20] M. Calzarossa, L. Massari, and D. Tessera, *Workload characterization issues and methodologies*, 2000. DOI: 10.1007/3-540-46506-5_20.

[21] M. C. Calzarossa, L. Massari, and D. Tessera, "Workload characterization," *ACM Computing Surveys*, vol. 48, pp. 1–43, 3 Feb. 2016, ISSN: 0360-0300. DOI: 10.1145/2856127.

[22] E. Parzen, "An approach to time series analysis," *The Annals of Mathematical Statistics*, vol. 32, pp. 951–989, 4 Dec. 1961, ISSN: 0003-4851. DOI: 10.1214/aoms/1177704840.

[23] S. L. Zeger, R. Irizarry, and R. D. Peng, "On time series analysis of public health and biomedical data," *Annual Review of Public Health*, vol. 27, pp. 57–79, 1 Apr. 2006, ISSN: 0163-7525. DOI: 10.1146/annurev.publhealth.26.021304.144517.

[24] R. McCleary, A. H. Richard, E. M. E, and D. McDowall, *Applied Time Series Analysis for the Social Sciences*. Sage Publications, 1980, p. 326.

[25] C. W. J. Granger and M. Hatanaka, *Spectral Analysis of Economic Time Series. (PSME-1)*. Princeton University Press, Dec. 1965, ISBN: 9781400875528. DOI: 10.1515/9781400875528.

[26] D. Gubbins, *Time Series Analysis and Inverse Theory for Geophysicists*. Cambridge University Press, Mar. 2004, ISBN: 9780521819657. DOI: 10.1017/CBO9780511840302.

[27] J. E. Gentle and S. J. Wilson, *Finding patterns in time series*, 2018. DOI: 10.1007/978-3-319-18284-1_6.

[28] X. Wang, K. A. Smith, and R. Hyndman, "Characteristic-based clustering for time series data," *Data Mining and Knowledge Discovery*, vol. 13, pp. 335–364, 3 Sep. 2006, ISSN: 1384-5810. DOI: 10.1007/s10618-005-0039-x.

[29] A. Alqahtani, M. Ali, X. Xie, and M. W. Jones, "Deep time-series clustering: A review," *Electronics*, vol. 10, p. 3001, 23 Dec. 2021, ISSN: 2079-9292. DOI: 10.3390/electronics10233001.

[30] S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah, "Time-series clustering – a decade review," *Information Systems*, vol. 53, pp. 16–38, Oct. 2015, ISSN: 03064379. DOI: 10.1016/j.is.2015.04.007.

[31] E. J. Keogh and M. J. Pazzani, *A simple dimensionality reduction technique for fast similarity search in large time series databases*, 2000. DOI: 10.1007/3-540-45571-X_14.

[32] H. Zhang, T. Ho, Y. Zhang, and S. Lin, "Unsupervised feature extraction for time series clustering using orthogonal wavelet transform.," *Informatica (Slovenia)*, vol. 30, pp. 305–319, Aug. 2006.

[33] J. Lin, M. Vlachos, E. Keogh, and D. Gunopulos, *Iterative incremental clustering of time series*, 2004. DOI: 10.1007/978-3-540-24741-8_8.

[34] M. Ruizhe, "Clustering of time series data: Measures, methods, and applications.," Computer Science, Aug. 2020.

[35] C.-S. J. Chu, "Time series segmentation: A sliding window approach," *Information Sciences*, vol. 85, no. 1, pp. 147–173, 1995, ISSN: 0020-0255. DOI: https://doi.org/10.1016/0020-0255(95)00021-G. [Online]. Available: https://www.sciencedirect.com/science/article/pii/002002559500021G.

[36] E. Keogh, J. Lin, and W. Truppel, "Clustering of time series subsequences is meaningless: Implications for previous and future research," IEEE Comput. Soc, pp. 115–122, ISBN: 0-7695-1978-4. DOI: 10.1109/ICDM.2003.1250910.

[37] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990. DOI: 10.1109/5.58325.

[38] M. Corduas and D. Piccolo, "Time series clustering and classification by the autoregressive metric," *Computational Statistics & Data Analysis*, vol. 52, no. 4, pp. 1860–1872, 2008, ISSN: 0167-9473. DOI: https://doi.org/10.1016/j.csda.2007.06.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167947307002368.

[39] M. Bicego, V. Murino, and M. Figueiredo, "Similarity-based clustering of sequences using hidden markov models," in May 2003, vol. 2734, pp. 95–.

[40] X. Wang, K. A. Smith, R. J. Hyndman, and D. Alahakoon, "A scalable method for time series clustering," *Computer Science*, 2004.

[41] M. Christ, A. Kempa-Liehr, and M. Feindt, "Distributed and parallel time series feature extraction for industrial big data applications," *ArXiv*, vol. abs/1610.07717, 2016.

[42] H. Zhang, T. B. Ho, Y. Zhang, and M. S. Lin, "Unsupervised feature extraction for time series clustering using orthogonal wavelet transform," *Informatica (Slovenia)*, vol. 30, pp. 305–319, 2006.

[43] I. Jolliffe and J. Cadima, "Principal component analysis: A review and recent developments," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, p. 20 150 202, Apr. 2016. DOI: 10.1098/rsta.2015.0202.

[44] R. Cangelosi and A. Goriely, "Component retention in principal component analysis with application to cdna microarray data," *Biology Direct*, vol. 2, p. 2, 1 2007, ISSN: 17456150. DOI: 10.1186/1745-6150-2-2.

[45] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning: With applications in r*, 2013. DOI: 10.1007/978-1-4614-7138-7_10.

[46] R. C. de Amorim and B. Mirkin, "Minkowski metric, feature weighting and anomalous cluster initializing in k-means clustering," *Pattern Recognition*, vol. 45, pp. 1061–1075, 3 Mar. 2012, ISSN: 00313203. DOI: 10.1016/j.patcog.2011.08.012.

[47] A. Kianimajd, M. Ruano, P. Carvalho, *et al.*, "Comparison of different methods of measuring similarity in physiologic time series," *IFAC-PapersOnLine*, vol. 50, pp. 11 005–11 010, 1 Jul. 2017, ISSN: 24058963. DOI: 10.1016/j.ifacol.2017.08.2479.

[48] *Dynamic time warping*, 2007. DOI: 10.1007/978-3-540-74048-3_4.

[49] "Dynamic time warping," Romain Tavenard. (2017), [Online]. Available: https://tslearn.readthedocs.io/en/stable/user_guide/dtw.html (visited on 03/28/2022).

[50] F. Petitjean, A. Ketterlin, and P. Gançarski, "A global averaging method for dynamic time warping, with applications to clustering," *Pattern Recognition*, vol. 44, pp. 678–693, 3 Mar. 2011, ISSN: 00313203. DOI: 10.1016/j.patcog.2010.09.013.

[51] J. MacQueen, *Some methods for classification and analysis of multivariate observations*, 1967.

[52] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," N. Bansal, K. Pruhs, and C. Stein, Eds., SIAM, 2007, pp. 1027–1035. [Online]. Available: http://dl.acm.org/citation.cfm?id=1283383.1283494.

[53] T. F. Gonzalez, "Clustering to minimize the maximum intercluster distance," *Theoretical Computer Science*, vol. 38, pp. 293–306, 1985, ISSN: 03043975. DOI: 10.1016/0304-3975(85)90224-5.

[54] U. von Luxburg, "Clustering stability: An overview," Jul. 2010. DOI: 10.1561/2200000008.

[55] P. Fränti and S. Sieranoja, "How much can k-means be improved by using better initialization and repeats?" *Pattern Recognition*, vol. 93, pp. 95–112, Sep. 2019, ISSN: 00313203. DOI: 10.1016/j.patcog.2019.04.014.

[56] L. Kaufmann and P. Rousseeuw, "Clustering by means of medoids," *Data Analysis based on the L1-Norm and Related Methods*, pp. 405–416, Aug. 1987.

[57] Y.-c. Wu and J.-w. Feng, "Development and application of artificial neural network," *Wireless Personal Communications*, vol. 102, pp. 1645–1656, 2 Sep. 2018, ISSN: 0929-6212. DOI: 10.1007/s11277-017-5224-x.

[58] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, pp. 59–69, 1 1982, ISSN: 0340-1200. DOI: 10.1007/BF00337288.

[59] G. Chicco, R. Napoli, and F. Piglione, "Application of clustering algorithms and self organising maps to classify electricity customers," IEEE, pp. 373–379, ISBN: 0-7803-7967-5. DOI: 10.1109/PTC.2003.1304160.

[60] T. Caliński and J. Harabasz, "A dendrite method for cluster analysis," *Communications in Statistics*, vol. 3, no. 1, pp. 1–27, 1974. DOI: 10.1080/03610927408827101.

[61] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, pp. 224–227, 2 Apr. 1979, ISSN: 0162-8828. DOI: 10.1109/TPAMI.1979.4766909.

[62] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, Nov. 1987, ISSN: 03770427. DOI: 10.1016/0377-0427(87)90125-7.

[63] "What are durable functions?" Microsoft. (2022), [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview?tabs=csharp (visited on 05/04/2022).

[64] "Hdf5," The HDF Group. (2022), [Online]. Available: https://portal.hdfgroup.org/display/HDF5/ (visited on 05/04/2022).

[65] C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, "Array programming with numpy," *Nature*, vol. 585, pp. 357–362, 7825 Sep. 2020, ISSN: 0028-0836. DOI: 10.1038/s41586-020-2649-2.

[66] D. A. Dickey and W. A. Fuller, "Distribution of the estimators for autoregressive time series with a unit root," *Journal of the American Statistical Association*, vol. 74, pp. 427–431, 366a Jun. 1979, ISSN: 0162-1459. DOI: 10.1080/01621459.1979.10482531.

[67] S. Glen. "Unit root: Simple definition, unit root tests." (Dec. 2016), [Online]. Available: https://www.statisticshowto.com/unit-root/ (visited on 05/05/2022).

[68] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in python," Jan. 2012.

[69] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2019.

[70] R. Tavenard, J. Faouzi, G. Vandewiele, *et al.*, "Tslearn, a machine learning toolkit for time series data," *Journal of Machine Learning Research*, vol. 21, no. 118, pp. 1–6, 2020. [Online]. Available: http://jmlr.org/papers/v21/20-091.html.

[71] wannesm, khendrickx, A. Yurtman, *et al.*, *Wannesm/dtaidistance: V2.3.5*, version v2.3.5, Jan. 2022. DOI: 10.5281/zenodo.5901139. [Online]. Available: https://doi.org/10.5281/zenodo.5901139.

[72] F. Petitjean, A. Ketterlin, and P. Gançarski, "A global averaging method for dynamic time warping, with applications to clustering," *Pattern Recognition*, vol. 44, no. 3, pp. 678–693, 2011.

[73] D. Kwiatkowski, P. C. Phillips, P. Schmidt, and Y. Shin, "Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root?" *Journal of Econometrics*, vol. 54, no. 1, pp. 159–178, 1992, ISSN: 0304-4076. DOI: https://doi.org/10.1016/0304-4076(92)90104-Y. [Online]. Available: https://www.sciencedirect.com/science/article/pii/030440769290104Y.

[74] A. Novikov, "Pyclustering: Data mining library," *Journal of Open Source Software*, vol. 4, p. 1230, Apr. 2019. DOI: 10.21105/joss.01230.

[75] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr, "Time series feature extraction on basis of scalable hypothesis tests (tsfresh – a python package)," *Neurocomputing*, vol. 307, pp. 72–77, Sep. 2018, ISSN: 09252312. DOI: 10.1016/j.neucom.2018.03.067.

[76] M. Nordlinder, "Clustering of financial account time series using self organizing maps," KTH Royal Institute of Technology, 2021. (visited on 08/26/2022).

[77] "Tsfresh," Maximillian Christ and Blue Yonder GmbH. (2022), [Online]. Available: https://tsfresh.readthedocs.io/en/latest/index.html (visited on 07/28/2022).

[78] "Tsfresh," Maximilian Christ and Blue Yonder GmbH. (2022), [Online]. Available: https://tsfresh.readthedocs.io/en/latest/text/feature_extraction_settings.html (visited on 07/28/2022).

[79] "Tsfresh," Maximilian Christ and Blue Yonder GmbH. (2022), [Online]. Available: https://tsfresh.readthedocs.io/en/latest/text/feature_extraction_settings.html#for-the-advanced-how-do-i-set-the-parameters-for-all-kind-of-time-series (visited on 08/26/2022).

[80] C. Aridas, J.-O. Joswig, T. Mathieu, and R. Yurchak. "Scikit-learning-extra: A machine learning extension of the scikit-learn." (2019), [Online]. Available: https://scikit-learn-extra.readthedocs.io.

[81] L. Kaufman and P. J. Rousseeuw, *Partitioning around medoids (program pam)*, Mar. 1990. DOI: 10.1002/9780470316801.ch2.

[82] G. Vettigli, *Minisom: Minimalistic and numpy-based implementation of the self organizing map*, 2018. [Online]. Available: https://github.com/JustGlowing/minisom/.

[83] J. Vesanto and E. Alhoniemi, "Clustering of the self-organizing map," *IEEE Transactions on Neural Networks*, vol. 11, no. 3, pp. 586–600, 2000. DOI: 10.1109/72.846731.

[84] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ser. ICML'13, Atlanta, GA, USA: JMLR.org, 2013, I–115–I–123.

[85] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, Eds., vol. 24, Curran Associates, Inc., 2011. [Online]. Available: https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

[86] S. Law, "Stumpy: A powerful and scalable python library for time series data mining," *Journal of Open Source Software*, vol. 4, p. 1504, 39 Jul. 2019, ISSN: 2475-9066. DOI: 10.21105/joss.01504.

A

Appendix

A.1 Optimal-k results for tested techniques

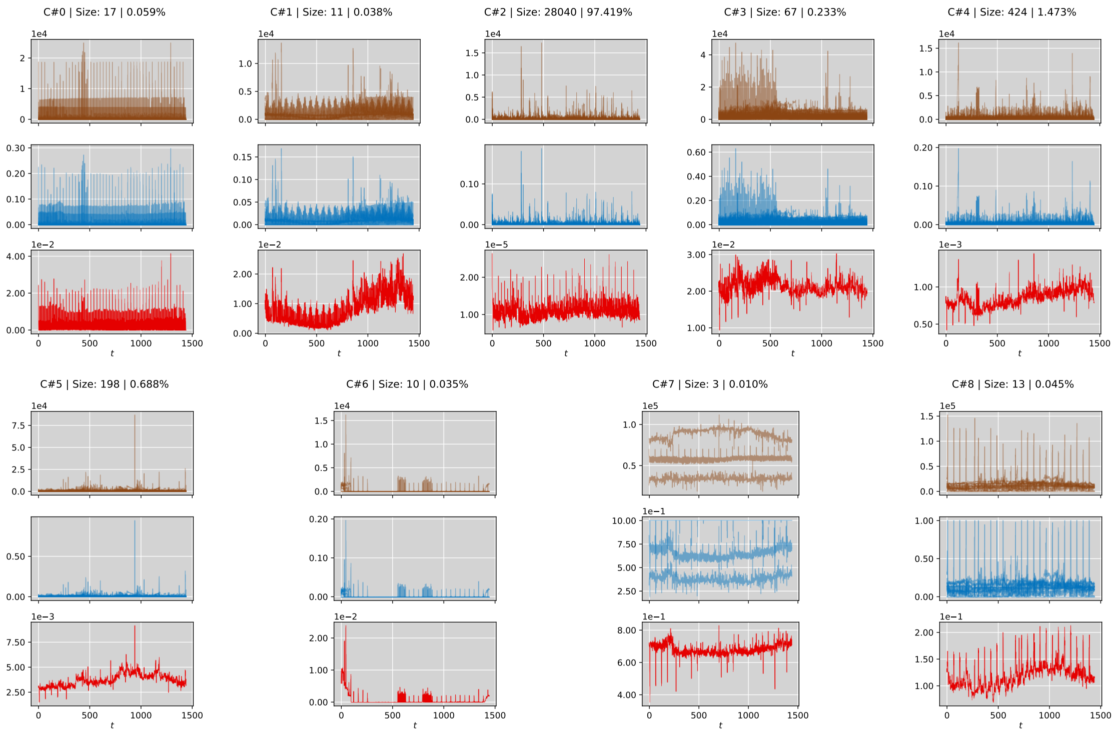


Figure A.1: Clusters produced by the $M1$ shape-based method via k-means algorithm using normalized time-series from **dataset I**. The graphs depicts the invocation time-series of the clustered serverless functions in the following order: (1) original time-series assigned to the cluster, (2) normalized time-series assigned to the cluster, (3) cluster's centroid derived from the normalized time-series. Additionally, each graph depicts the cluster's label (C), the total number of time-series assigned to the cluster ($Size$), and the percentage share of the assigned functions relative to the entire dataset. Clusters $C2$, $C4$, and $C5$ contain the majority of dataset, representing 97.419%, 1.473%, and 0.688% of the functions respectively.

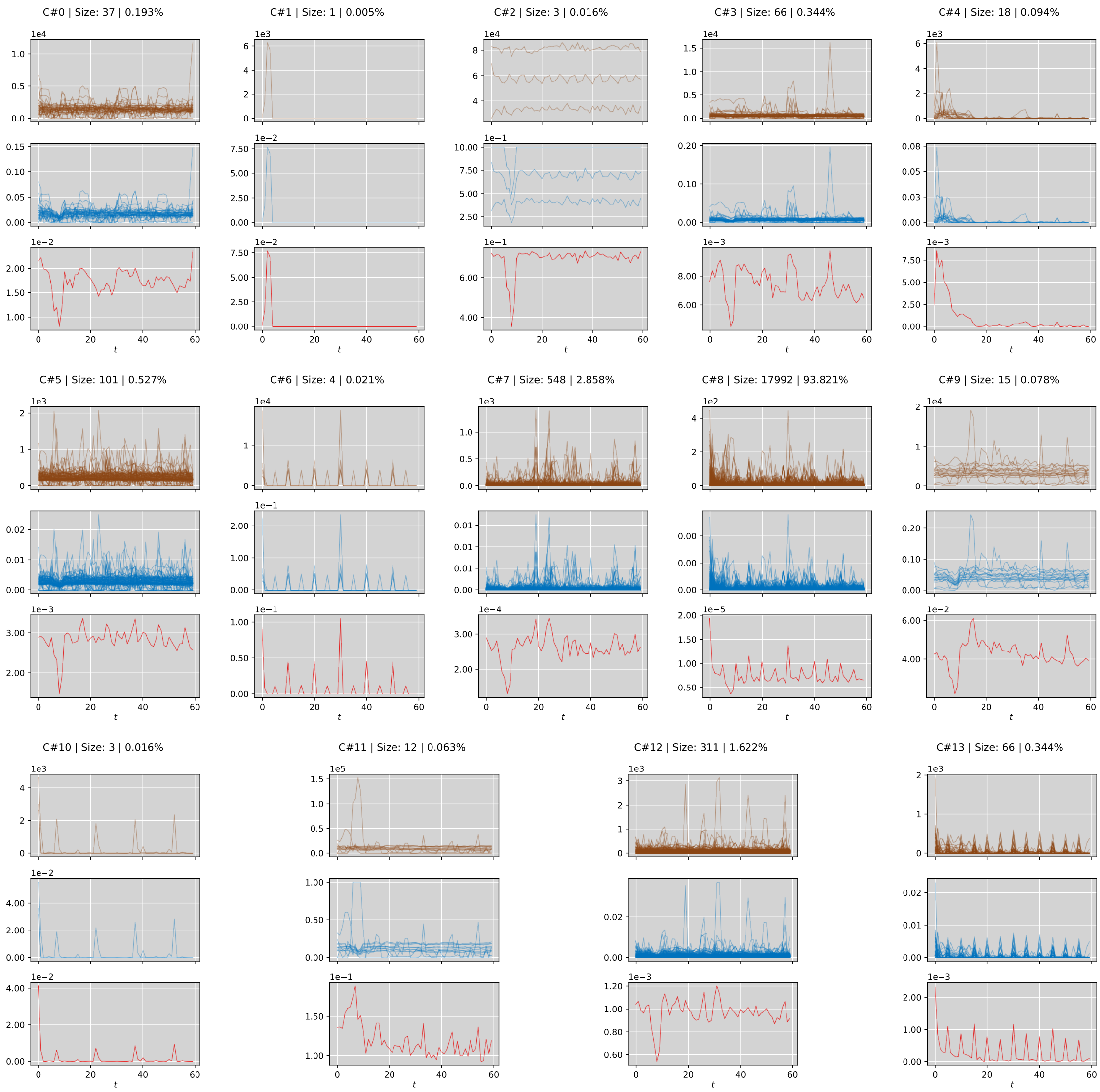


Figure A.2: Clusters produced by the $M1$ shape-based method via k-means algorithm using normalized time-series from **dataset II**. The graphs depicts the invocation time-series of the clustered serverless functions in the following order: (1) original time-series assigned to the cluster, (2) normalized time-series assigned to the cluster, (3) cluster's centroid derived from the normalized time-series. Additionally, each graph depicts the cluster's label (C), the total number of time-series assigned to the cluster ($Size$), and the percentage share of the assigned functions relative to the entire dataset. Clusters $C8$, $C7$, and $C12$ contain the majority of dataset, containing 93.821%, 2.858%, and 1.622% of the functions respectively.

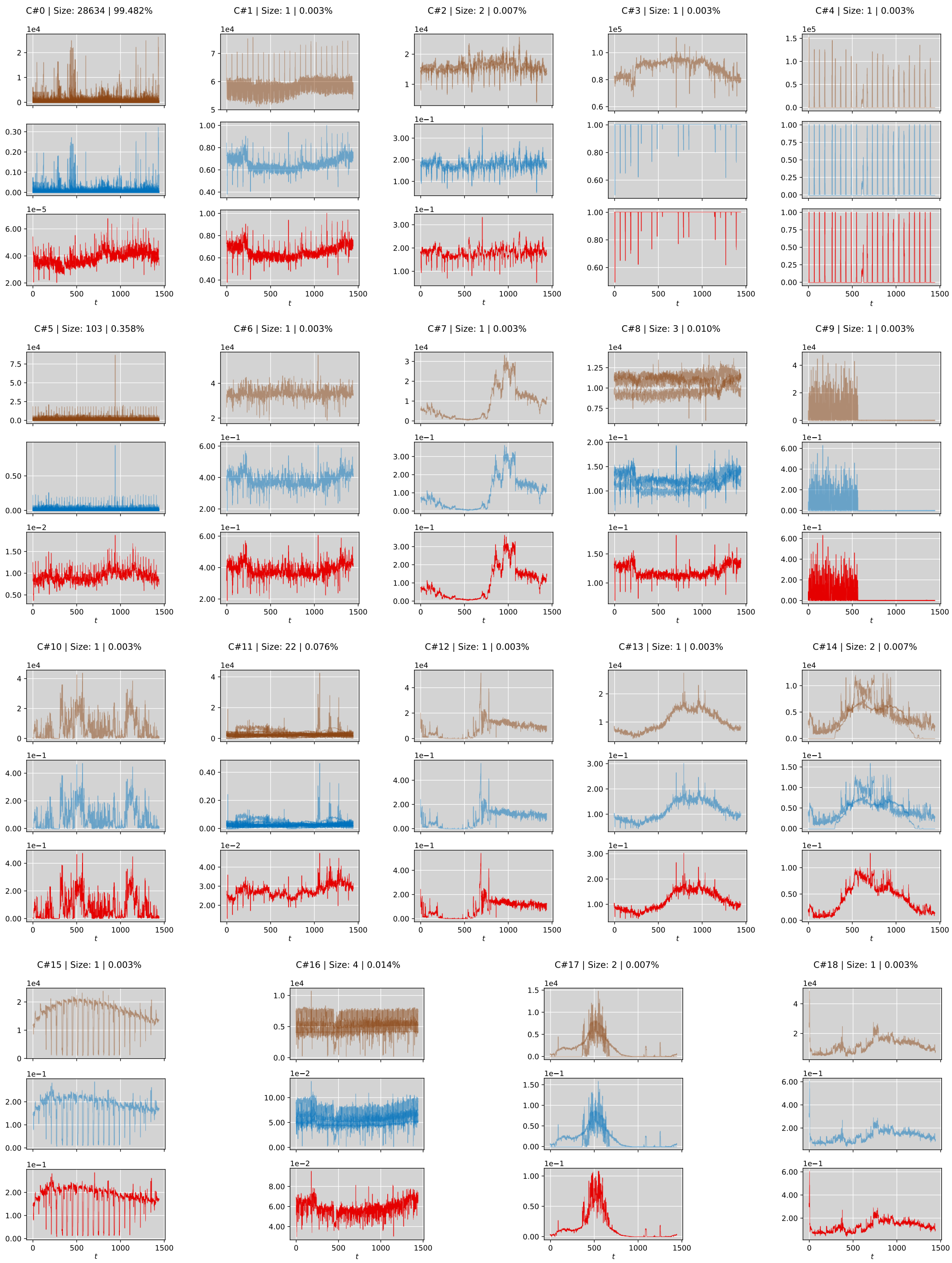


Figure A.3: Clusters produced by the $M2$ shape-based method via k-means algorithm using normalized time-series from **dataset I**. The graphs depicts invocation time-series assigned to the cluster in the following order: (1) original series (2) normalized series, (3) cluster's centroid. Additionally, each graph depicts the cluster's label (C), the total number of time-series assigned to the cluster ($Size$), and the percentage share of the assigned functions relative to the entire dataset. Clusters $C0$, $C5$, and $C11$ contain the majority of dataset, containing 99.482%, 0.358%, and 0.076% of the functions respectively.

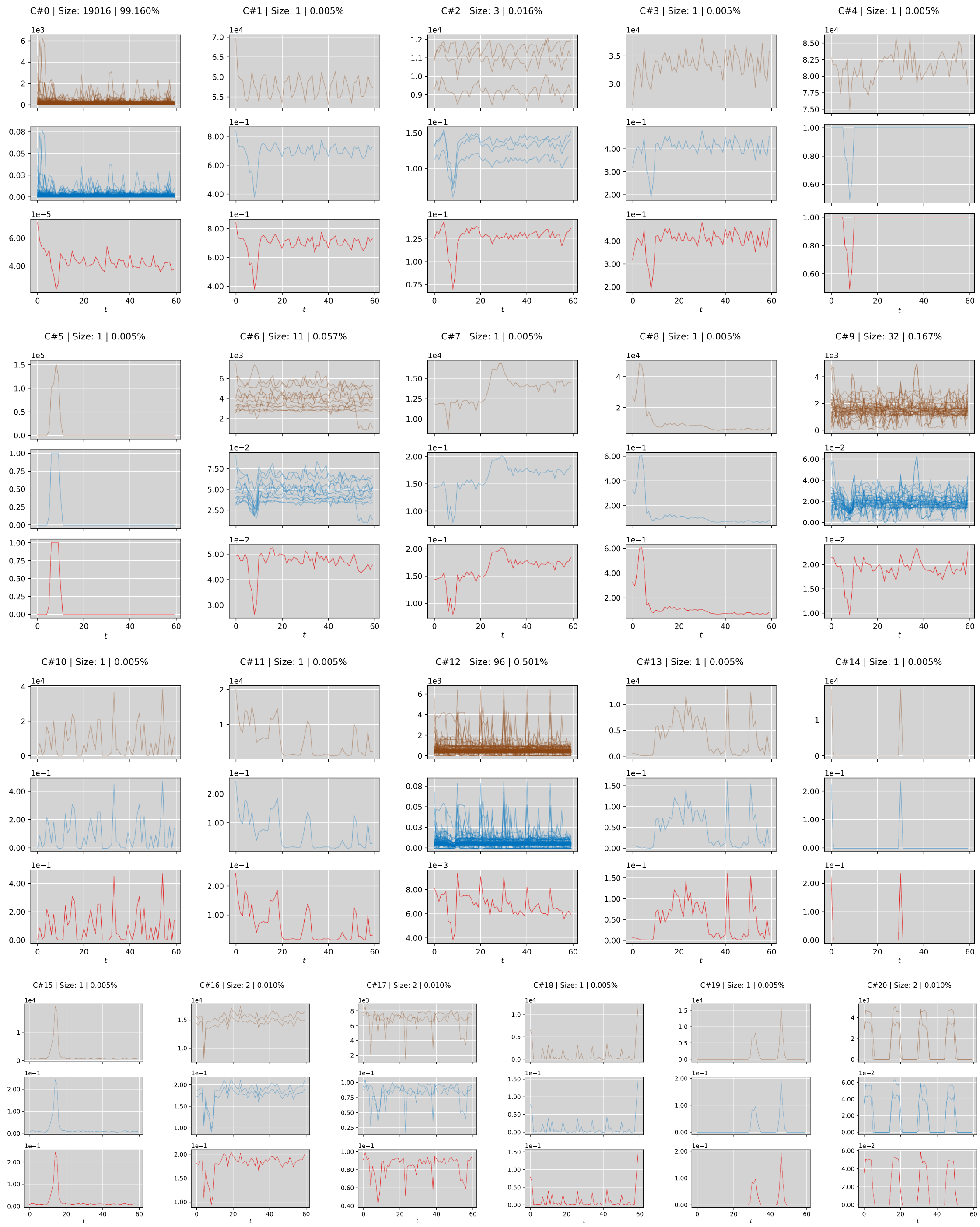


Figure A.4: Clusters produced by the $M2$ shape-based method via k-means algorithm using normalized time-series from **dataset II**. The graphs depicts invocation time-series assigned to the cluster in the following order: (1) original series (2) normalized series, (3) cluster's centroid. Additionally, each graph depicts the cluster's label (C), the total number of time-series assigned to the cluster ($Size$), and the percentage share of the assigned functions relative to the entire dataset. Clusters $C0$, $C12$, and $C9$ contain the majority of dataset, containing 99.160%, 0.501%, and 0.167% of the functions respectively.

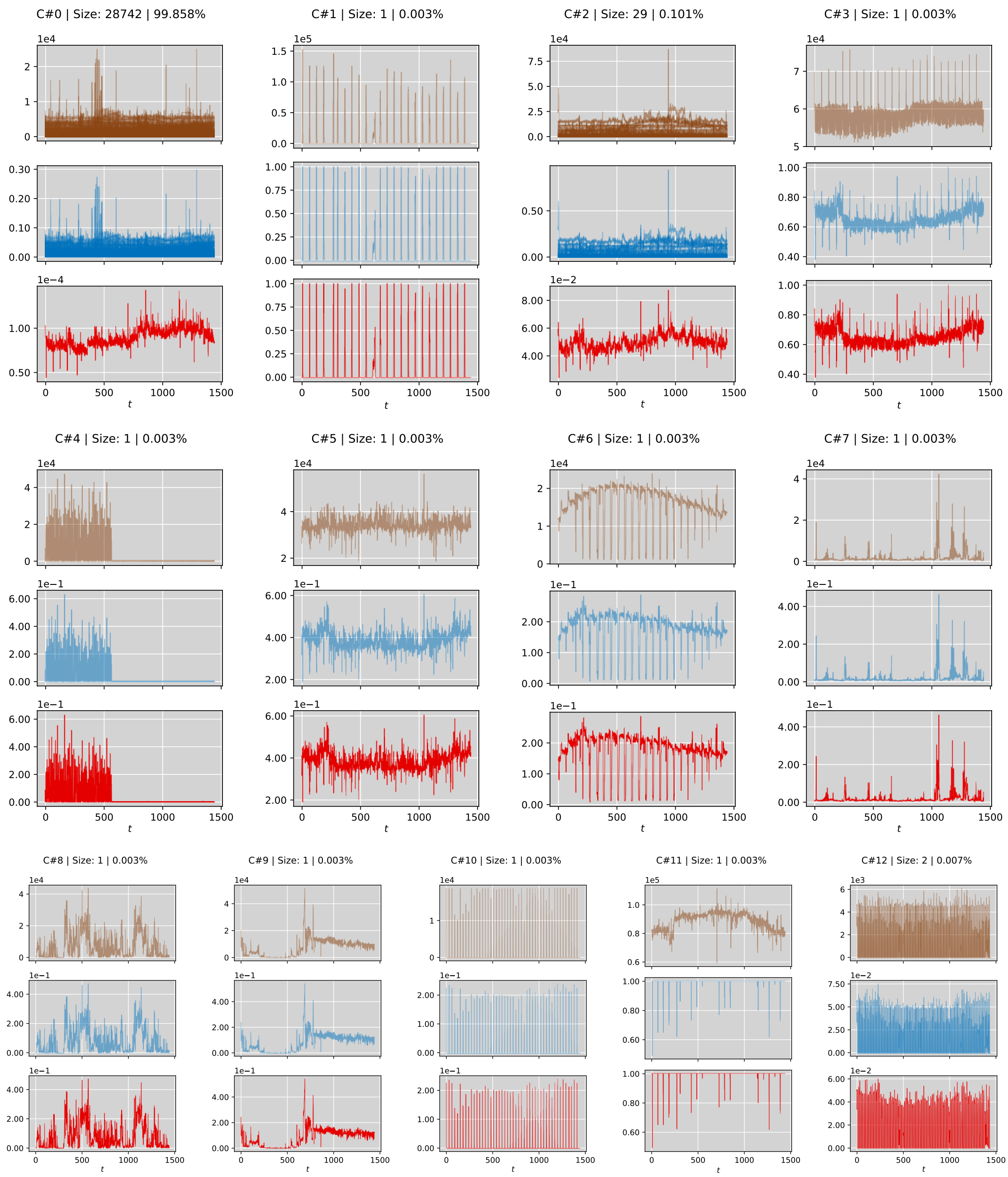


Figure A.5: Clusters produced by the $M3$ shape-based method via k-means algorithm using DBA and normalized time-series from **dataset I**. The graphs depicts invocation time-series assigned to the cluster in the following order: (1) original series (2) normalized series, (3) cluster's centroid. Additionally, each graph depicts the cluster's label (C), the total number of time-series assigned to the cluster ($Size$), and the percentage share of the assigned functions relative to the entire dataset. Clusters $C0$, $C2$, and $C12$ contain the majority of dataset, containing 99.858%, 0.101%, and 0.007% of the functions respectively.

Figure A.6: Clusters produced by the $M3$ shape-based method via k-means algorithm using DBA and normalized time-series from **dataset II**. The graphs depicts invocation time-series assigned to the cluster in the following order: (1) original series (2) normalized series, (3) cluster's centroid. Additionally, each graph depicts the cluster's label (C), the total number of time-series assigned to the cluster ($Size$), and the percentage share of the assigned functions relative to the entire dataset. Clusters $C0$, $C19$, and $C9$ contain the majority of dataset, containing 99.426%, 0.339%, and 0.073% of the functions respectively.

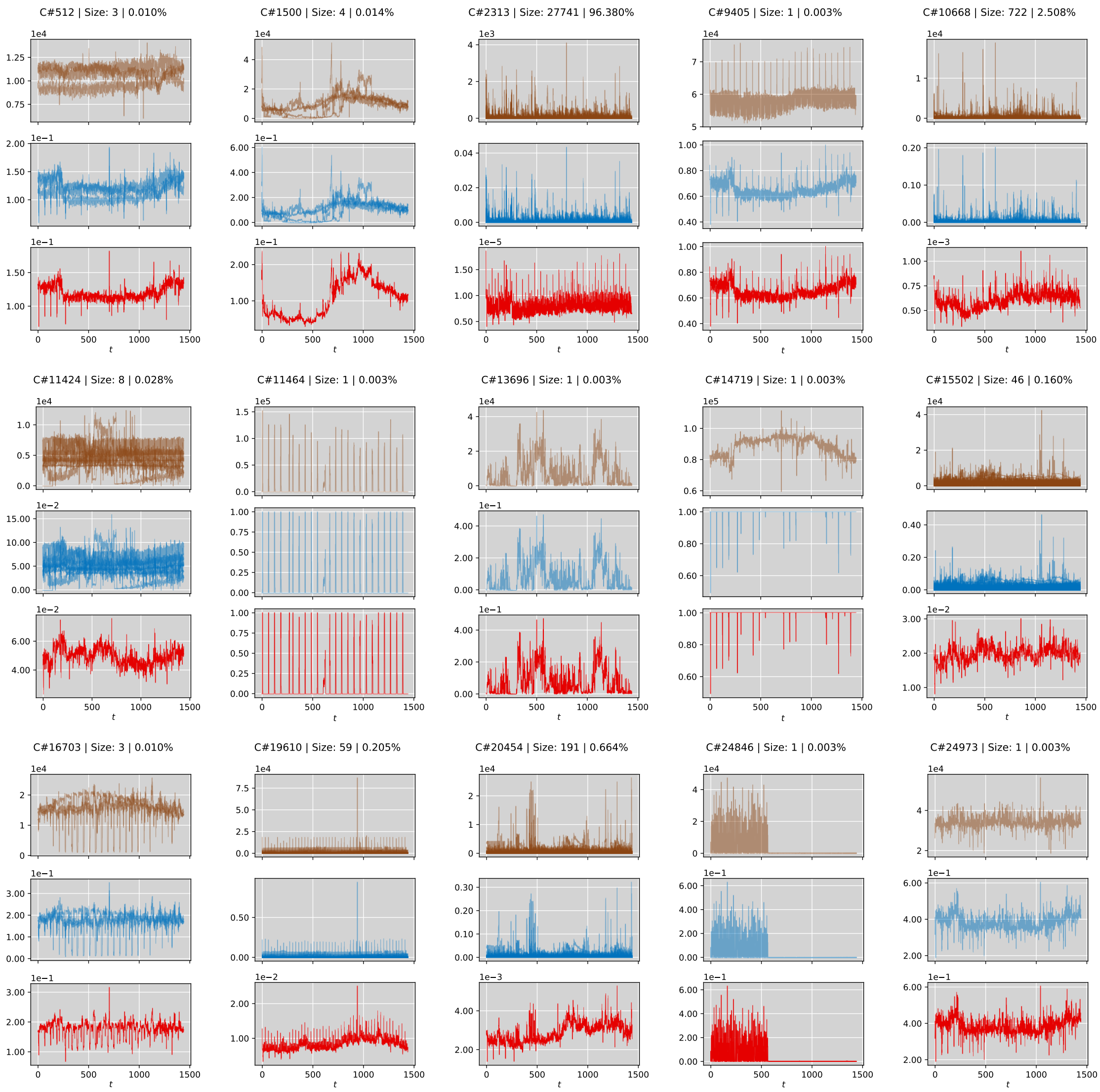


Figure A.7: Clusters produced by the $M4$ shape-based method via k-medoids algorithm using DTW-processed pairwise distance matrices derived from the normalized time series of **dataset I**. The graphs depicts invocation time-series assigned to the cluster in the following order: (1) original series (2) normalized series, (3) cluster's centroid. Additionally, each graph depicts the cluster's label (C), the total number of time-series assigned to the cluster ($Size$), and the percentage share of the assigned functions relative to the entire dataset. Clusters $C2313$, $C10668$, and $C20454$ contain the majority of dataset, containing 96.380%, 2.508%, and 0.664% of the functions respectively.

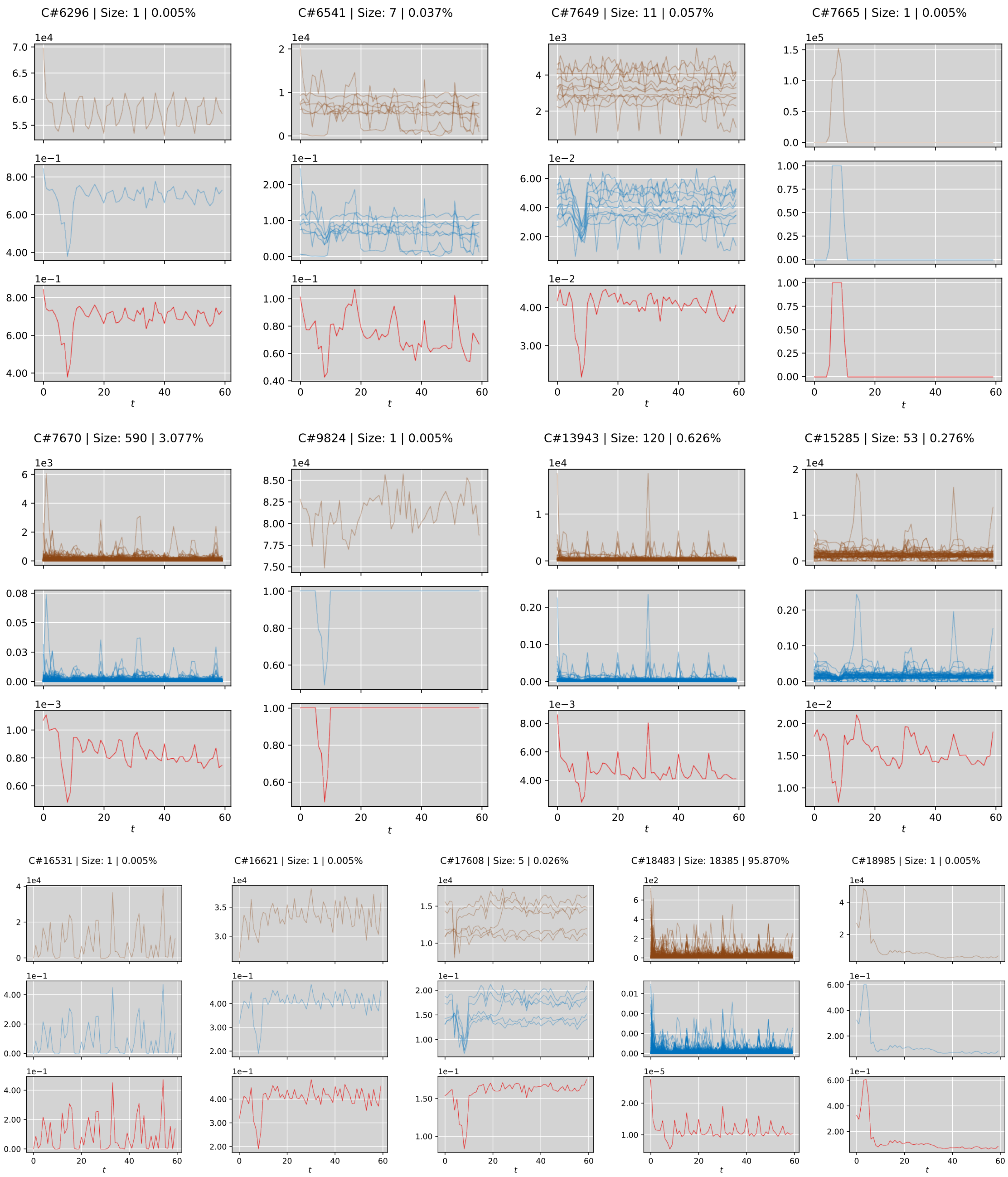


Figure A.8: Clusters produced by the $M4$ shape-based method via k-medoids algorithm using DTW-processed pairwise distance matrices derived from the normalized time series of **dataset II**. The graphs depicts invocation time-series assigned to the cluster in the following order: (1) original series (2) normalized series, (3) cluster's centroid. Additionally, each graph depicts the cluster's label (C), the total number of time-series assigned to the cluster ($Size$), and the percentage share of the assigned functions relative to the entire dataset. Clusters $C18483$, $C7670$, and $C13943$ contain the majority of dataset, containing 95.870%, 3.077%, and 0.626% of the functions respectively.

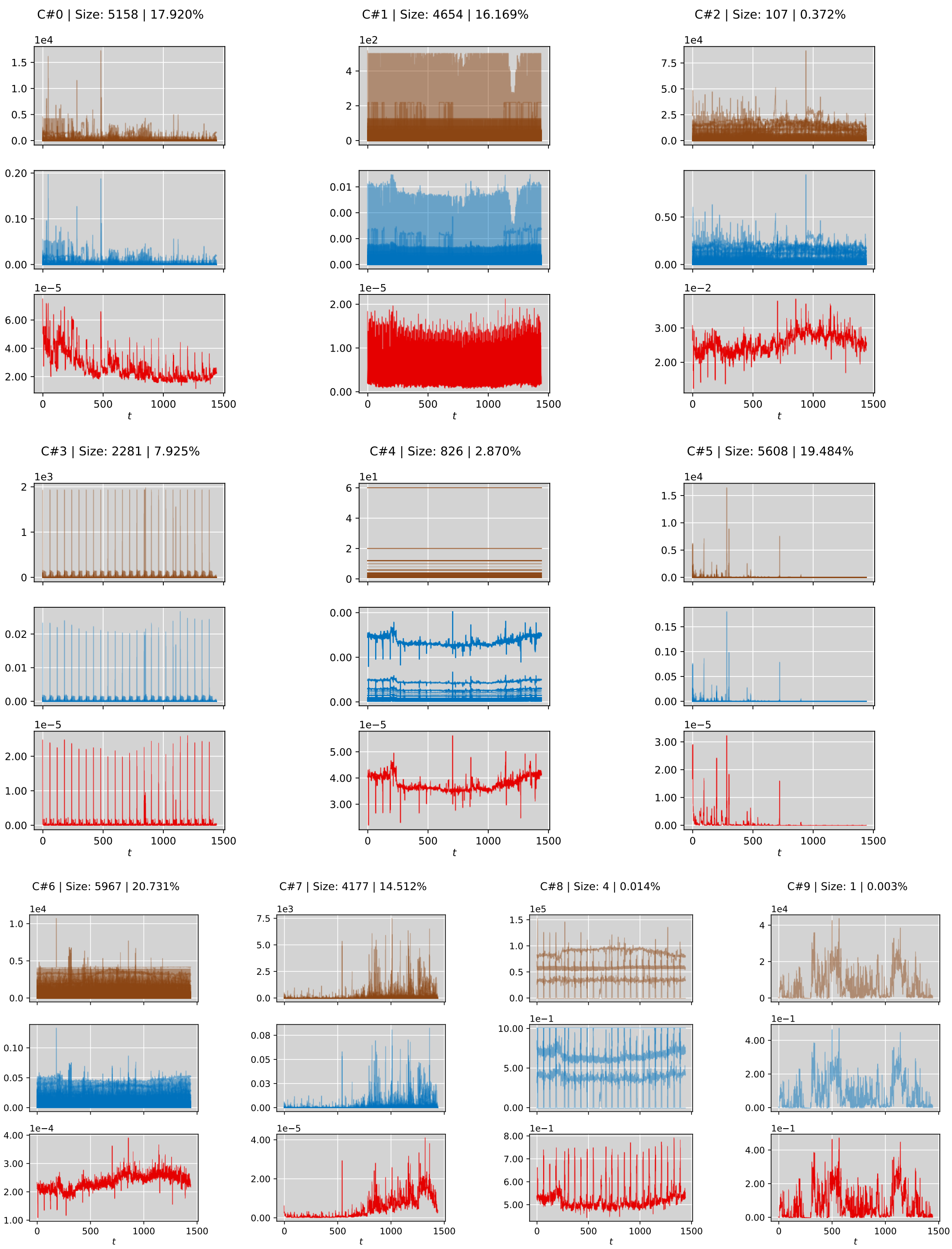


Figure A.9: Clusters produced by the $M5$ feature-based method via k-means algorithm using input vectors of principal components. Each vector contains 49 components derived from features of **dataset I**. The graphs depict invocation time-series assigned to the cluster in the following order: (1) original series (2) normalized series, (3) cluster's centroid based on the normalized series. Additionally, each graph depicts the cluster's label (C), the total number of time-series assigned to the cluster ($Size$), and the percentage share of the assigned functions relative to the entire dataset. Clusters $C6$, $C5$, and $C0$ contain the majority of dataset, containing 20.731%, 19.484%, and 17.920% of the functions respectively.

Figure A.10: Clusters produced by the $M5$ feature-based method via k-means algorithm using input vectors of principal components. Each vector contains 20 components derived from features of **dataset II**. The graphs depict invocation time-series assigned to the cluster in the following order: (1) original series (2) normalized series, (3) cluster's centroid based on the normalized series. Additionally, each graph depicts the cluster's label (C), the total number of time-series assigned to the cluster ($Size$), and the percentage share of the assigned functions relative to the entire dataset. Clusters $C4$, $C8$, and $C12$ contain the majority of dataset, containing 16.989%, 12.917%, and 6.586% of the functions respectively.

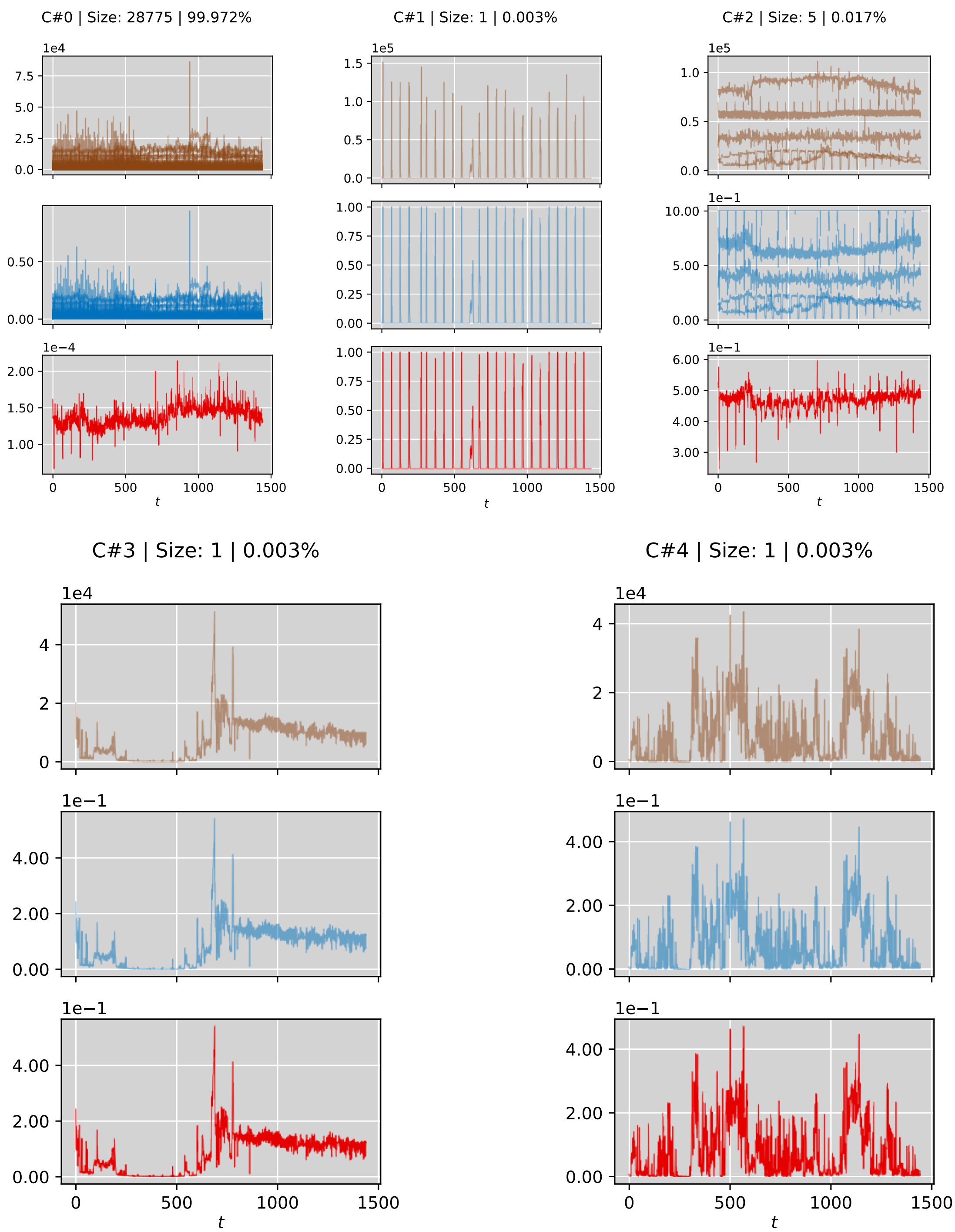


Figure A.11: Clusters produced by the $M6$ feature-based method via k-medoids algorithm using input vectors of principal components. Each vector contains 49 components derived from features of **dataset I**. The graphs depict invocation time-series assigned to the cluster in the following order: (1) original series (2) normalized series, (3) cluster's centroid based on the normalized series. Additionally, each graph depicts the cluster's label (C), the total number of time-series assigned to the cluster ($Size$), and the percentage share of the assigned functions relative to the entire dataset. Clusters $C0$ and $C2$ contain the majority of dataset, containing 99.972% and 0.017% of the functions respectively.

Figure A.12: Clusters produced by the $M6$ feature-based method via k-medoids algorithm using input vectors of principal components. Each vector contains 20 components derived from features of **dataset II**. The graphs depict invocation time-series assigned to the cluster in the following order: (1) original series (2) normalized series, (3) cluster's centroid based on the normalized series. Additionally, each graph depicts the cluster's label (C), the total number of time-series assigned to the cluster ($Size$), and the percentage share of the assigned functions relative to the entire dataset. Clusters $C0$ and $C2$ contain the majority of dataset, containing 99.972% and 0.017% of the functions respectively.

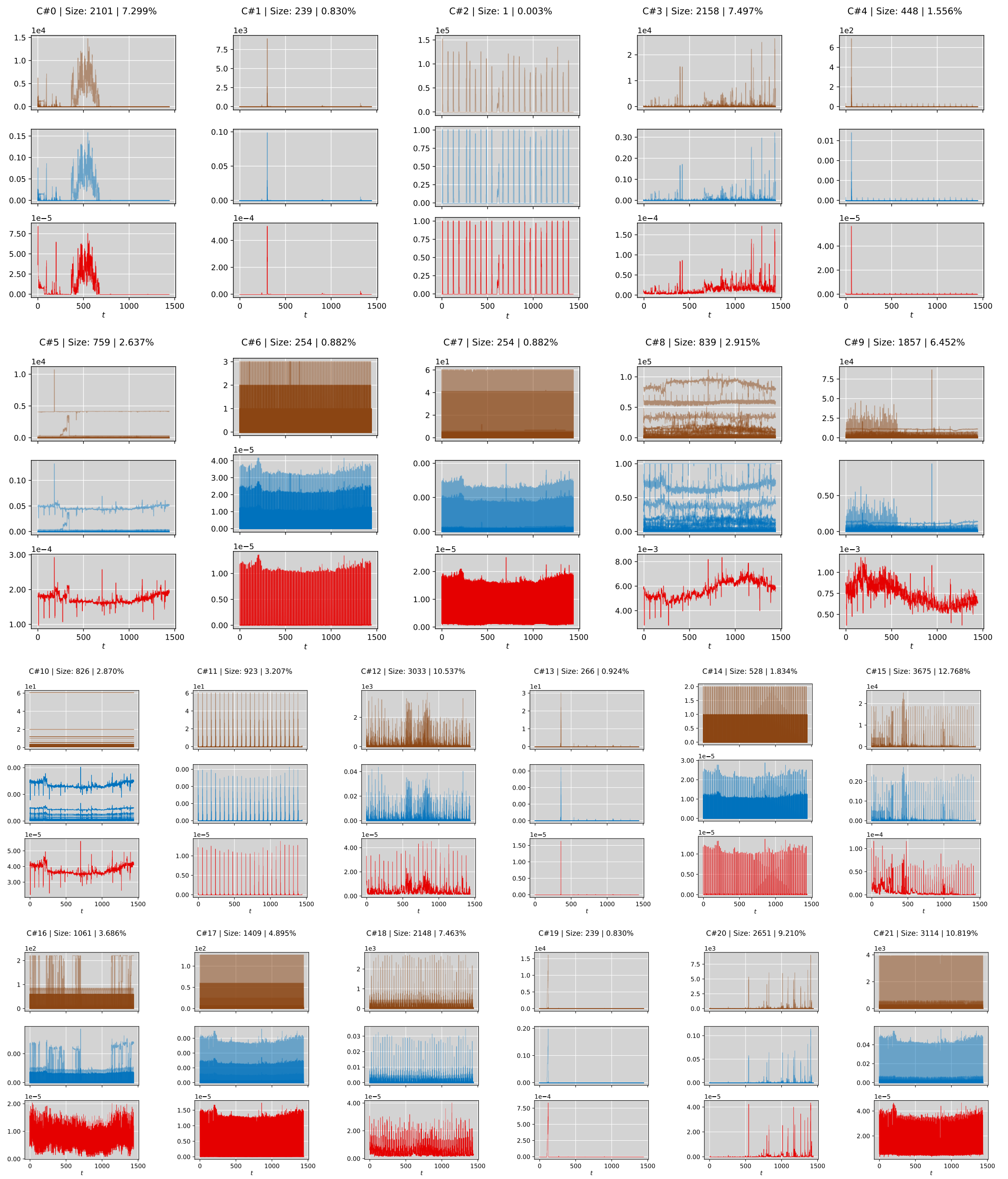


Figure A.13: Clusters produced by the $M7$ feature-based method via k-medoids (PAM) algorithm using input vectors of principal components. Each vector contains 49 components derived from features of **dataset I**. The graphs depict invocation time-series assigned to the cluster in the following order: (1) original series (2) normalized series, (3) cluster's centroid based on the normalized series. Additionally, each graph depicts the cluster's label (C), the total number of time-series assigned to the cluster ($Size$), and the percentage share of the assigned functions relative to the entire dataset. Clusters $C15$, $C21$, and $C12$ contain the majority of dataset, containing 12.768%, 10.819%, and 10.537% of the functions respectively.

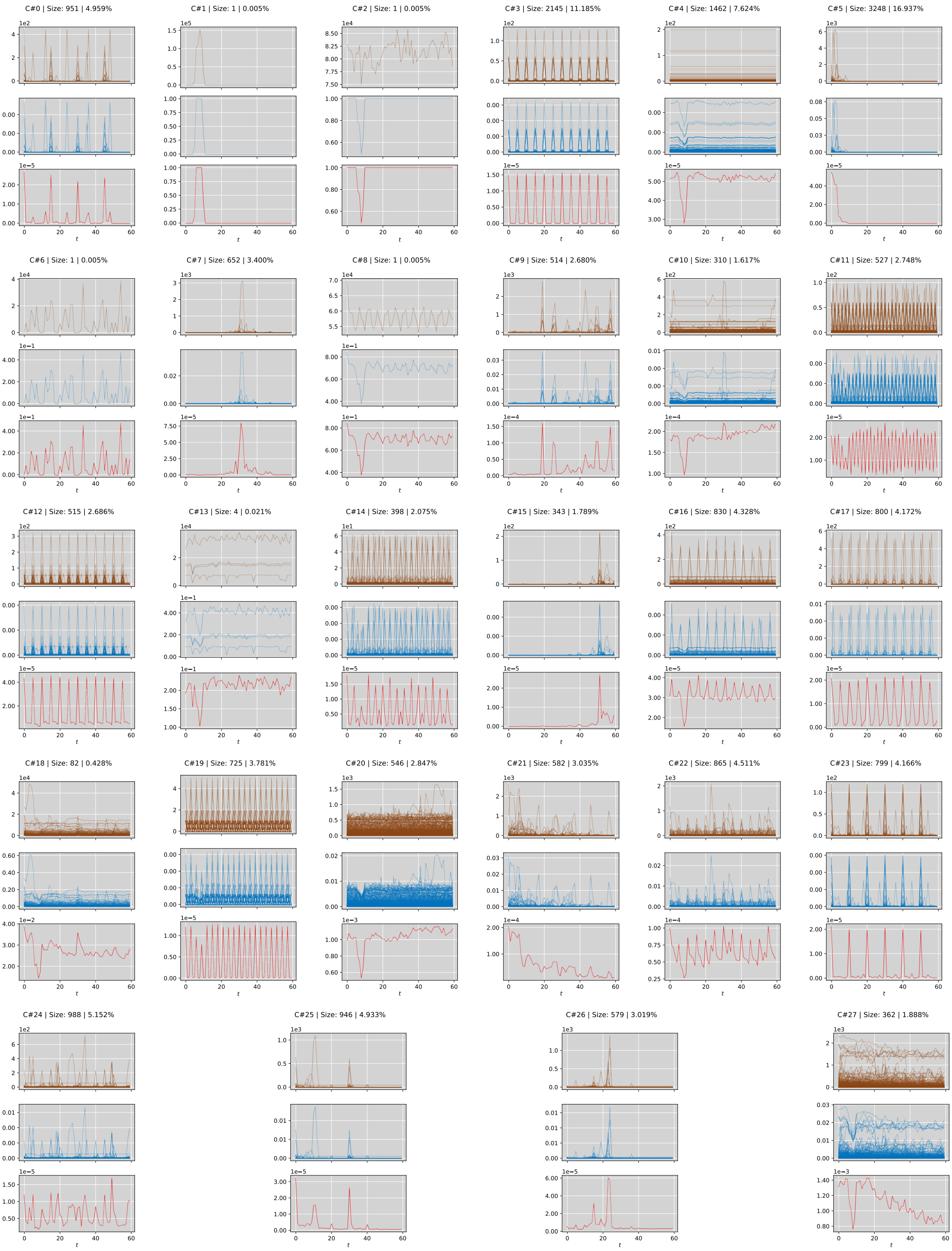


Figure A.14: Clusters produced by the $M7$ feature-based method via k-medoids (PAM) algorithm using input vectors of principal components. Each vector contains 20 components derived from features of **dataset II**. The graphs depict invocation time-series assigned to the cluster in the following order: (1) original series (2) normalized series, (3) cluster's centroid based on the normalized series. Additionally, each graph depicts the cluster's label (C), the total number of time-series assigned to the cluster ($Size$), and the percentage share of the assigned functions relative to the entire dataset. Clusters $C15$, $C21$, and $C12$ contain the majority of dataset, containing 12.768%, 10.819%, and 10.537% of the functions respectively.

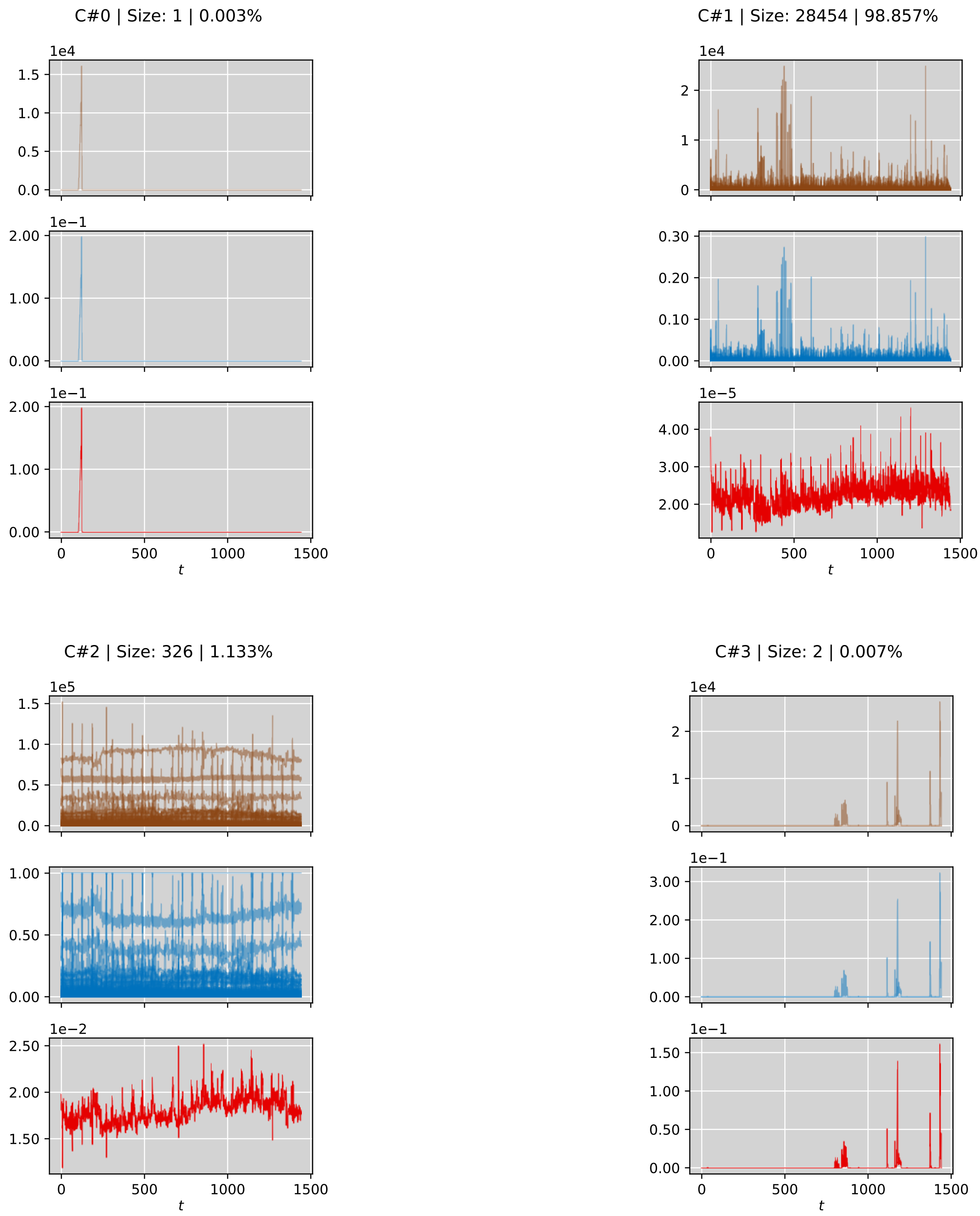


Figure A.15: Clusters produced by the $M8$ hybrid method using self organizing map (SOM) and k-means algorithm and normalized time series from **dataset I**. Clustering the map's U-matrix yielded 4 clusters with time series, and 3 clusters matching the empty regions without assigned data. The graphs depict invocation time-series assigned to the cluster in the following order: (1) original series (2) normalized series, (3) cluster's centroid based on the normalized series. Additionally, each graph depicts the cluster's label (C), the total number of time-series assigned to the cluster ($Size$), and the percentage share of the assigned functions relative to the entire dataset. Clusters $C1$ and $C2$ contain the majority of dataset, containing 98.857% and 1.133% of the functions respectively.

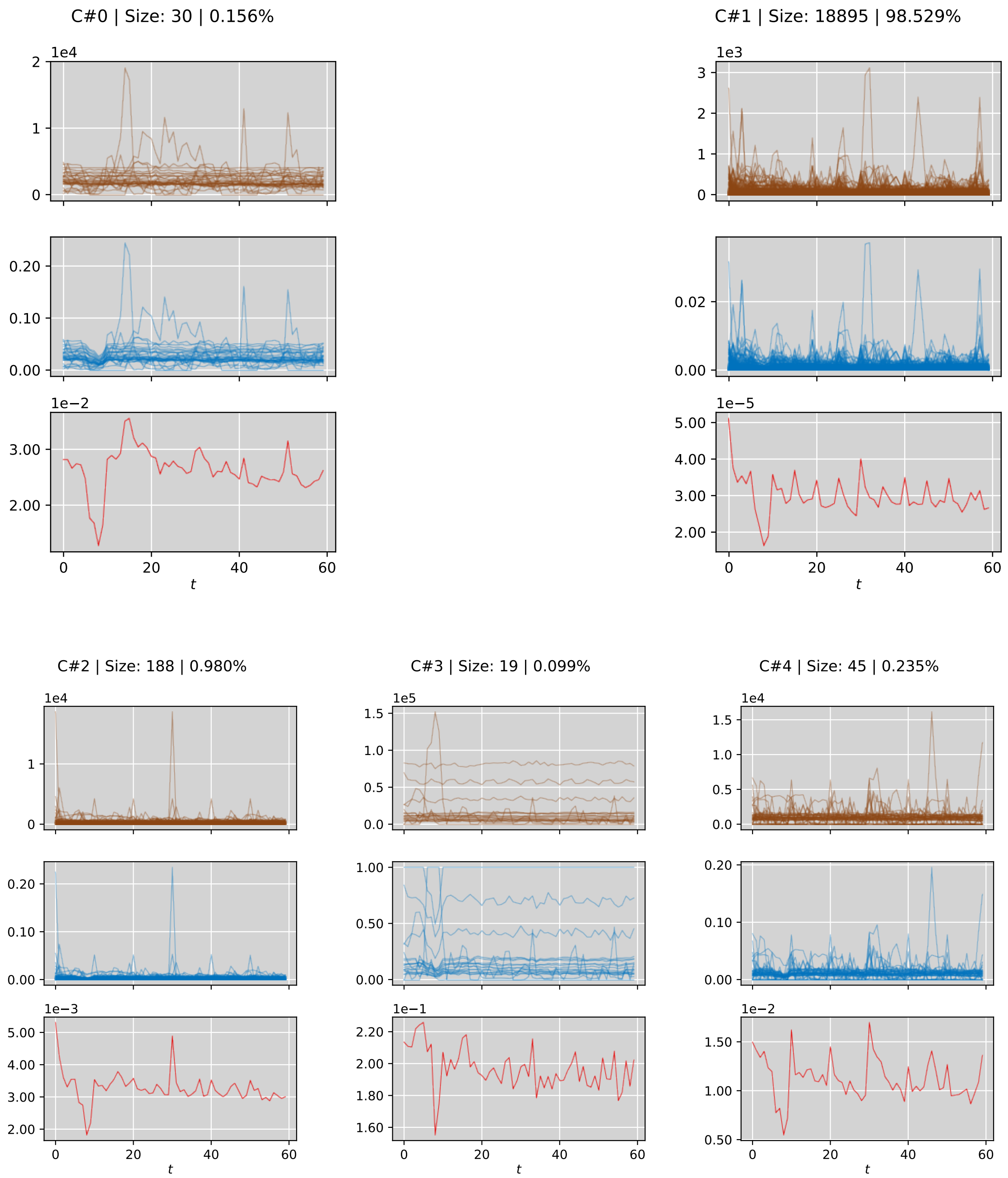


Figure A.16: Clusters produced by the $M8$ hybrid method using self organizing map (SOM) and k-means algorithm and normalized time series from **dataset II**. Clustering the map's U-matrix yielded 4 clusters with time series, and 3 clusters matching the empty regions without assigned data. The graphs depict invocation time-series assigned to the cluster in the following order: (1) original series (2) normalized series, (3) cluster's centroid based on the normalized series. Additionally, each graph depicts the cluster's label (C), the total number of time-series assigned to the cluster ($Size$), and the percentage share of the assigned functions relative to the entire dataset. Clusters $C1$ and $C2$ contain the majority of dataset, containing 98.529% and 0.980% of the functions respectively.

A.2 Cluster distribution for optimal-k results

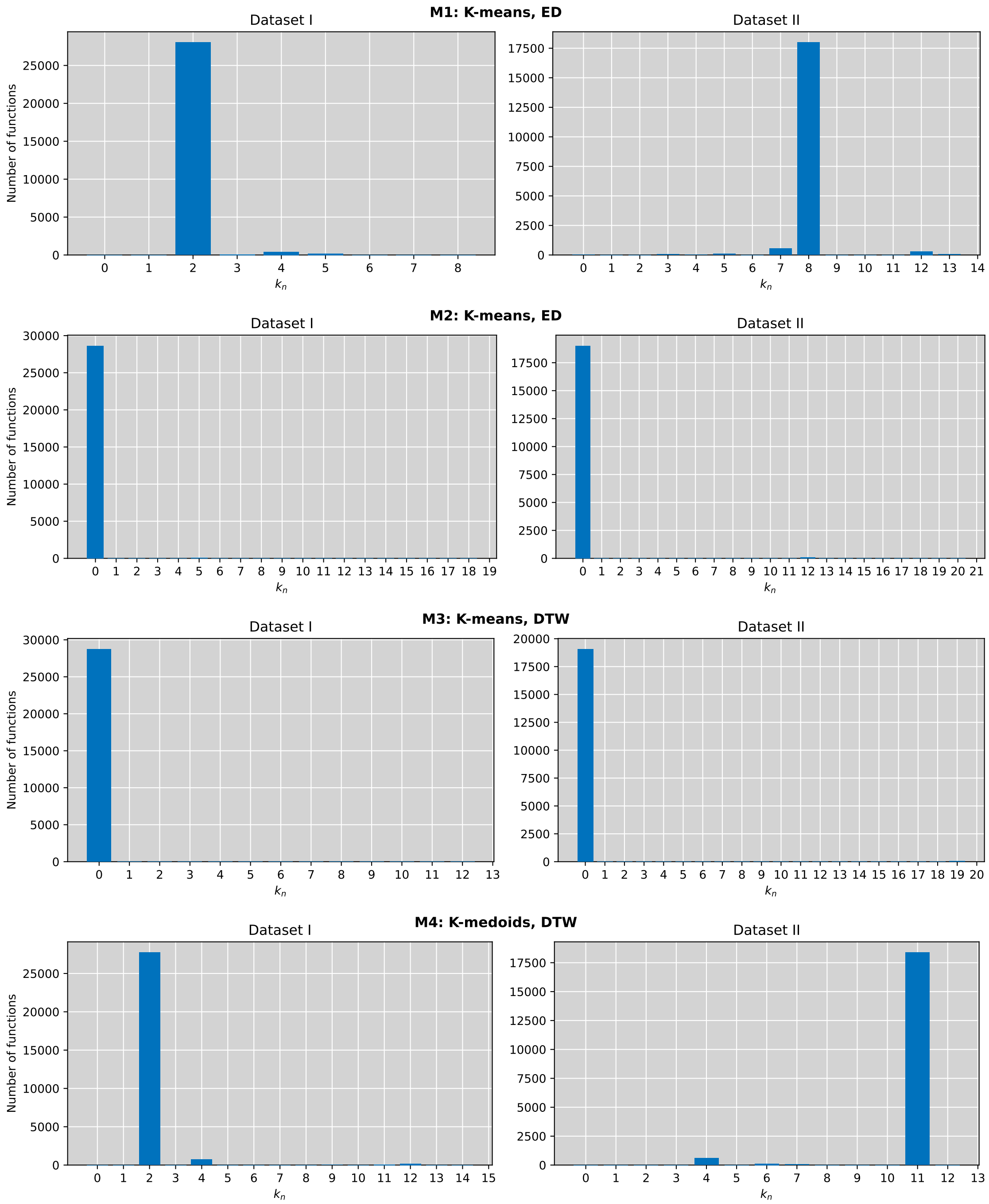


Figure A.17: Number of invocation functions assigned to clusters C_n for the tested shape-based techniques. Despite the given k chosen according to the best internal validation metrics' scores, the visual inspection indicates poor results, with 1 dominant cluster present in each shape-based result, containing more than 95% of serverless functions within respective datasets.

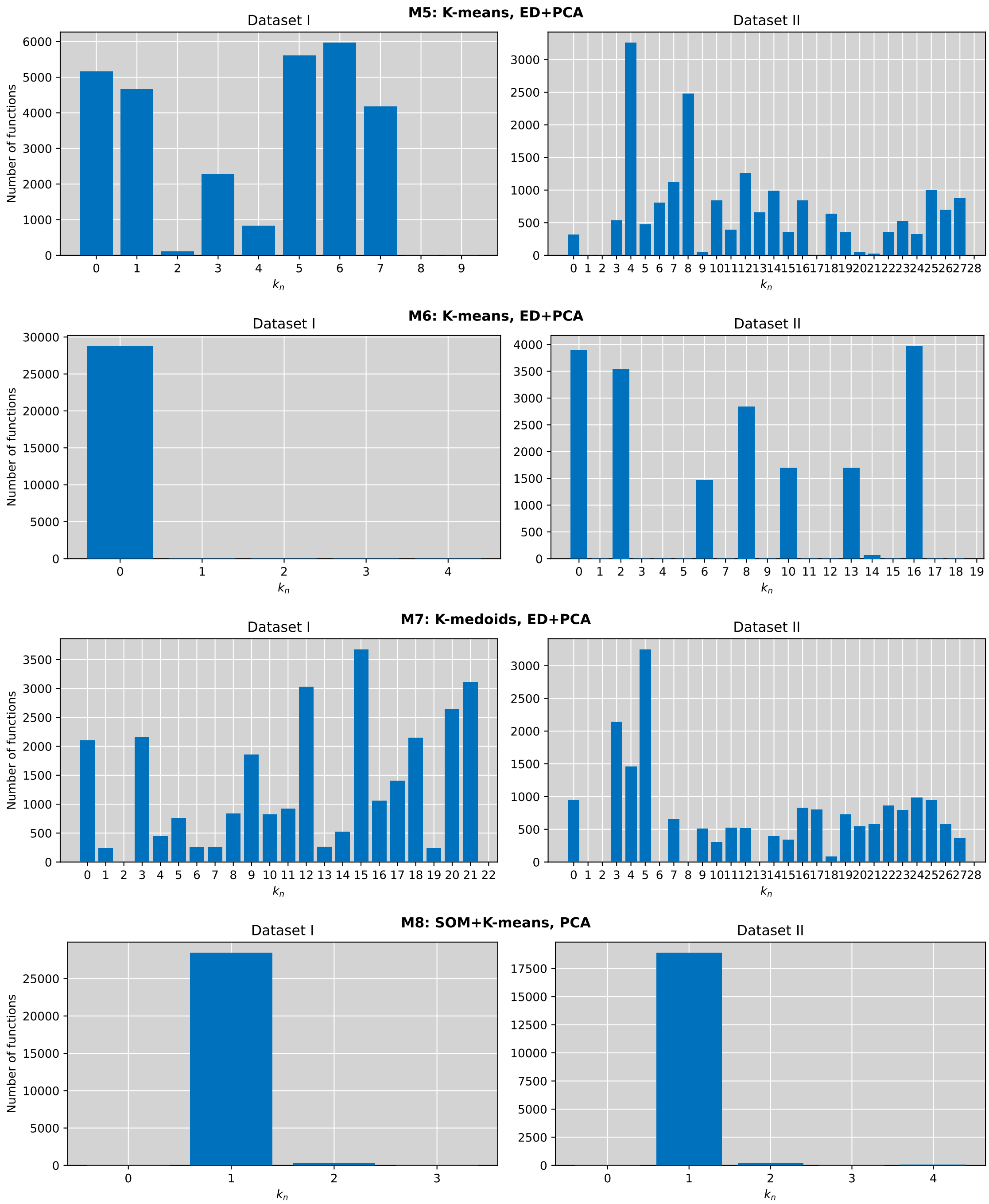


Figure A.18: Number of invocation functions assigned to clusters C_n for the tested feature-based and hybrid techniques. With the exception to the $M6$ technique applied on dataset I, the feature-based clustering produce more balanced distribution of serverless functions between clusters C_n. Meanwhile, the hybrid technique $M8$ which combines dimensionality-reduction via self organizing map (SOM), PCA-based initialization of SOM's initial weights, and applying the k-means algorithm to SOM's distance map (U-matrix) produces results comparable shape-based techniques showcased in Fig. A.17