The Journal of Systems & Software 200 (2023) 111665

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Multi-objective empirical computational complexity of single-tenant service instances deployed at the Edge☆

Dionysis Athanasopoulos [a,*], Mitchell McEwen [b]

[a] School of Electronics, Electrical Engineering, and Computer Science, Queen's University Belfast, Belfast, BT7 1NN, UK
[b] DXC Technology NZ Limited, 100 Willis Street, Wellington, 6011, New Zealand

ARTICLE INFO

Article history:
Received 15 August 2022
Received in revised form 5 December 2022
Accepted 26 February 2023
Available online 1 March 2023

Keywords:
Web services
Empirical complexity
Predictive model
Edge computing

ABSTRACT

The Edge has been recently introduced to augment the back-end of apps with service instances deployed to machines that are located close to end-users' devices. However, given that machines at the Edge are usually resource constrained, a service instance that needs a big amount of hardware resources may be inefficiently executed at the Edge. We consider a service instance is Edge-efficient if the runtime consumption of hardware resources by a service instance is lower than the amount of the hardware resources provided by a machine at the Edge. The runtime consumption of hardware resources of a service instance depend on the computational complexity of the algorithms implemented by the service instance. The computational complexity should include multiple objectives like the time objective and the space objective. Each objective usually corresponds to a separate hardware resource. However, the automated calculation of the theoretical computational complexity is reduced to the insolvable halting problem. We propose an approach that learns the multi-objective empirical complexity of service instances deployed at the Edge as mathematical functions of the input parameters of the programming interface of services. We further contribute with an algorithm that decides on whether a service instance is Edge-efficient. We evaluate the accuracy of our approach via measuring the percentages of the correct decisions on Edge-efficient service instances. We use in our experiment a real-world mobile app that has been deployed to a resource-constrained machine at the Edge. We compare the accuracy of our approach against two baseline state-of-the-art approaches. The results show that the accuracy of our approach is higher than the accuracy of the baseline approaches.

1. Introduction

The use of mobile apps for performing data analytics is constantly being increased (Plebani et al., 2017). The back-end of apps is usually deployed as a service in remote (e.g. cloud) powerful servers. The interaction between the front-end and the back-end of apps is achieved by using the technology of web services (Erl, 2016). However, the use of remote servers for performing data analytics may lead the apps' end-users to experience delays on receiving the analysis results due to unexpected network latency. Shorter network latency can be achieved if data are analyzed at machines closer to end-users' devices.

The Edge has been recently introduced to augment the back-end of apps with service instances deployed to machines that are located near/at the network edge (close to end-users' devices) (Hong and Varghese, 2019). However, given that machines at the Edge are usually resource constrained, a service instance that needs a big amount of hardware resources (e.g. CPU, RAM) may be inefficiently executed at the Edge, leading to runtime deterioration of the *dynamic QoS* of service instances (Kritikos et al., 2013). The term of the dynamic QoS refers to the values of QoS attributes that are measured during the execution of service instances. For instance, the lower the frequency of the CPUs provided by a machine at the Edge is, the higher the execution time of a service instance in this machine is. We introduce in this work the term of an *Edge-efficient* service instance to refer to a service instance with runtime consumption of hardware resources that is lower than the amount of the hardware resources provided by at least one machine at the Edge.

1.1. Motivation and challenges

In practice, if a service instance is not Edge-efficient with respect to the machine where the service instance has been deployed, then the platform that performs *reactive* management of the machines at the Edge can reassign the service instance to another machine. However, if we want to enhance the platform

☆ Editor: Laurence Duchien.

* Corresponding author.
E-mail addresses: D.Athanasopoulos@qub.ac.uk (D. Athanasopoulos), mmcewen3@dxc.com (M. McEwen).

https://doi.org/10.1016/j.jss.2023.111665
0164-1212/

with a *proactive* management of machines (to avoid deploying service instances to machines where reactive management will be needed), then we need to train models that can predict the runtime consumption of hardware resources of machines by service instances.

To build predictive models of the runtime consumption of hardware resources, we need to have available historical values of the runtime consumption of hardware resources by service instances. In general, the consumption of hardware resources by software programs depend on the *computational complexity* of the algorithms implemented by the programs. According to the theory of computational complexity proposed by Papadimitriou (1994), the *theoretical* computational complexity of an algorithm captures the complexity of the algorithm as the size of the data provided as input to the algorithm increases. Especially, the theoretical *worst-case* complexity (that is usually expressed by using the Big-O notation) of an algorithm captures the *growth rate* (e.g. linear, quadratic) of the complexity of the algorithm as the size of input data becomes *asymptotically* big.

For instance, the theoretical worst-case complexity of an algorithm that includes a loop depends on/scales with the number of the iterations of the loop. The higher is the number of the iterations of the loop requested by an algorithm, the higher the worst-case time complexity of the algorithm is and consequently, the higher the execution time of the algorithm is. An algorithm with high worst-case time complexity consumes a big amount of CPU for asymptotically big input data.

The *manual calculation* of the theoretical complexity of service instances is challenging because the implementation of a service instance is usually long and complex (e.g. the implementation may include many interconnected software sub-systems). On the other hand, the *automated calculation* of the theoretical complexity of a large-sized algorithm is reduced to the *insolvable halting* problem (Papadimitriou, 1994). Moreover, the automated calculation of the computational complexity of service instances should capture the runtime consumption of hardware resources on a *particular utility* of the service instances.

The computational complexity of an algorithm can include multiple *objectives* (e.g. time complexity, space complexity) (Papadimitriou, 1994). Each objective should correspond to the consumption of a separate hardware resource (e.g. the time objective corresponds to CPU). However, the usage of one objective is not enough for capturing the computational complexity of an algorithm. For instance, an algorithm may use a big amount of RAM in order to decrease its execution time. To this end, a big amount of data should be stored in RAM for retrieving them in a very short amount of time during the execution of an algorithm. In this way, while an algorithm consumes a small amount of CPU, the algorithm consumes a big amount of RAM. In this case, the space objective that corresponds to RAM is needed for capturing the computational complexity of an algorithm.

The approaches that make decisions based on QoS predictions (Chen and Bahsoon, 2017) calculate a single QoS value by integrating the values of all the QoS attributes/objectives by using aggregation functions (Beliakov et al., 2007), machine-learning models (Tan et al., 2019), or collaborative-filtering techniques (Silic et al., 2015). In other words, these approaches convert a multi-objective decision to a single-objective decision that may negatively affect the effectiveness of a decision because a single value cannot always be clearly interpreted (i.e. it hides the values of its individual objectives) (Athanasopoulos and Zarras, 2015).

Implications. To train predictive models of the computational complexity of service instances, we need the historical consumption of hardware resources by the service instances in machines at the Edge. A separate model is trained for each machine because the total amount of provided hardware resources of machines can be different from one machine to another machine. We consider machines that provide the same hardware resources (e.g. CPU, RAM) even if the type of each hardware resource is differ from one machine to another machine (e.g. CPU with Intel Core i3 vs i5).

Assumptions. However, a service instance deployed to a machine may be accessed by multiple users that results in a dynamic change of the available amount of hardware resources in the machine (multi-tenancy models) (Ru et al., 2014). Moreover, the management activities (e.g. migration of virtual machines) performed by a platform can interfere with the execution of a service instance and cause deterioration to the dynamic QoS of the service instance (Amannejad et al., 2015b). In our current work, we have left as future work the consideration of multi-tenancy models and platform-management activities.

Overall, we face the following research challenge: *"How can we train predictive single-tenant models of the multi-objective management-agnostic computational complexity of a service instance (deployed to a machine at the Edge) as a function of the input data to the service instance?"*

1.2. Motivating example

To illustrate our research challenge and sketch how we address it, we present below a motivating example that is further used as a running example throughout the paper and as a case study in the evaluation of the proposed approach.

In our example, Stacey is an avid user of the eBay[1] marketplace, always ready to snap up a bargain in an online auction. But Stacey finds it difficult to decide on the best reserve price to maximize her profits from an auction. Stacey prefers using mobile apps for predicting a price in auctions (Kaur et al., 2011). Such auction apps use data-mining approaches (e.g. k-means clustering algorithm) for making groups and analyzing historical bidding data (Tan et al., 2019). However, data-mining approaches usually demands a big amount of hardware resources to be executed.

From the perspective of software architecture, the back-end of an auction app performs data analysis, while the front-end corresponds to the part of the app that interacts with end-users. The network latency in the communication between the front-end and the back-end is a crucial factor in online auctions. Potential delayed responses from the back-end may result in financial loses. This is the reason we assume that the back-end of an auction app is deployed to a machine at the Edge (in a close proximity with Stacey's device).

The back-end is exposed as a web service (Erl, 2016) following the architectural style of the mobile back-end as a service.[2] The programming interfaces/APIs of web services can be specified by using the REST technology (Richardson and Ruby, 2007) or the SOAP technology (Erl, 2016). Web services at the Edge usually follow the REST technology because the execution of RESTful services is lighter than the execution of SOAP-based services with respect to the amount of the consumed hardware resources.

However, the back-end of an auction app is not always efficiently executed in machines with limited amount of hardware resources. For instance, running the back-end in a machine with CPU frequency at 2.4 GHz and RAM at 2 GB on a small amount

[1] https://www.ebay.co.uk.

[2] https://azure.microsoft.com/en-us/solutions/mobile.

of data (less than 500 bidding data), Stacey was happy to see that the execution time was ranging from 0.15–34 s. Unfortunately, Stacey experienced the execution time of 3.5 min for a big amount of data (more than 5000 bidding data). This is the reason that Stacey used another auction app that promises smaller execution time by using a bigger amount of RAM to cache data. In this time, Stacey experienced a failure of getting a response from the app for a big amount of data because the machine where the service instance was deployed went out of memory.

To overcome this, we need a *proactive* approach that trains models of the computational complexity of a service instance to predict the runtime amount of hardware resources consumed by the service instance in the machines and decide whether the service instance is Edge-efficient at these machines. These models should be multi-objective models for covering various hardware resources of machines consumed by the service instances deployed to these machines.

1.3. Contribution

We are inspired by an approach that calculates the *empirical computational complexity* of programs to approximate the theoretical computational complexity of the programs (Goldsmith et al., 2007). This approach is a general-purpose approach applied to small-sized programs for calculating the time objective of (single-objective) empirical complexity as a mathematical function of a small number of input variables of programs. Additionally, this approach calculates the empirical complexity of small parts of an algorithm (i.e. it does not calculate the complexity of a whole algorithm). This approach is not directly applicable to the case of service instances because web services can accept complex JSON/XML data as input that contain tens/hundreds of variables. Moreover, the implementation of a service instance usually corresponds to large-scale software system.

To overcome the restrictions in the state-of-the-art, we propose an approach that learns the multi-objective empirical computational complexity of service instances deployed at the Edge as mathematical functions of the input parameters of the programmable interface/API of web services. Our approach uses the learnt predictive models of the multi-objective empirical complexity of a service instance for deciding on whether the service instance is Edge-efficient.

Overall, our contribution is threefold: (i) the formal specification of the multi-objective empirical computational complexity of service instances; (ii) an algorithm for learning predictive models of multi-objective empirical complexity; (iii) an algorithm for deciding on whether a service instance is Edge-efficient. Our approach is suitable for learning empirical computational complexity of atomic web services (i.e. services that do not internally invoke any other service) and belongs to the category of the single-tenant and management-agnostic approaches.

We developed a research prototype, `EmpCom`, of our approach. We evaluate the accuracy of `EmpCom` via measuring the percentages of the correct decisions on Edge-efficient service instances. We compare the accuracy of `EmpCom` against a baseline single-objective approach the uses the technique of the regression analysis to train predictive models (Athanasopoulos et al., 2019). We further compared the accuracy of `EmpCom` against a baseline single-objective approach that uses a machine-learning technique to train predictive models (Athanasopoulos and Liu, 2022). We conduct our experiment by using the auction app of our motivating example as a case-study. We extend the back-end of the auction app to use a clustering algorithm (Tan et al., 2019). The back-end is deployed to a resource-constrained machine at the Edge. We provide to the clustering algorithm real-word datasets collected from the UC Irvine machine-learning repository (Dheeru and Karra Taniskidou, 2017).

The rest of the paper is structured as follows. Section 2 surveys the related approaches to determine the research gap in the literature. Section 3 defines the conceptual model of web services at the Edge. Section 4 specifies the fundamentals on the single-objective empirical complexity of programs. Section 5 defines the concept of the multi-objective empirical complexity of service instances. Sections 6 and 7 describe the algorithms for learning predictive models and deciding on whether a service instance is Edge-efficient, respectively. Section 8 presents the evaluation of our approach. Finally, Section 9 summarizes our contribution and discusses the future directions of our research.

2. Related work

We organize the related approaches into two groups: (i) the approaches that build profiles of dynamic QoS of web services (Section 2.1); (ii) the approaches that build predictive models of dynamic QoS of web services (Section 2.2). Our work belongs to the category of the approaches that build predictive models of dynamic QoS because we consider that the efficiency of the execution of service instances in machines at the Edge (a.k.a. Edge-efficiency) is a QoS attribute. To measure/predict the multiple aspects of the efficiency of service instances (e.g. time efficiency, space efficiency), we build multi-objective models of the computational complexity of the service instances that can predict the multi-aspect efficiency of the service instances.

2.1. Profiling dynamic QoS of web services

Runtime profilers visualize measurements of dynamic QoS of software systems (Ammons et al., 2004). In the case of web services, an approach monitors dynamic QoS values with millisecond precision (Lai, 2018). Another approach monitors dynamic QoS values at all the software-engineering phases of the life-cycle of service-oriented systems (Oriol et al., 2015). However, software engineers should determine in what monitoring/profiled data are interested. This task is a time-consuming and error-prone task. To overcome this, an approach adds an extra feature to classical runtime profilers that checks the compliance of dynamic QoS to SLA (Service-Level Agreement) between service providers and clients (Ibrahim et al., 2018b). Another approach provides a rich language for specifying the monitoring of multiple QoS attributes (Dawes et al., 2020). Overall, runtime profilers do not make predictions of dynamic QoS that would be used for deciding on whether a service instance is Edge-efficient.

2.2. Predicting dynamic QoS of web services

The existing approaches build predictive models for *atomic* services (that do not internally invoke any other service) or *composite* services. Some approaches are *infrastructure-agnostic* approaches (Sections 2.2.1 and 2.2.3) that do not consider the machines at the Cloud, the Edge, or the Fog, where service instances have been deployed. Some other approaches are infrastructure-based approaches (Sections 2.2.2 and 2.2.4). To build a predictive model, AI techniques (Goodfellow et al., 2016) or collaborative-filtering techniques (Herlocker et al., 1999) have been adopted.

A summary and a comparison of the approaches that build predictive models of the QoS of service instances are presented in Table 1. We observe that all the existing approaches convert a multi-objective decision to a single-objective decision by producing a single prediction value that integrates the values of all the QoS objectives. Moreover, there are only two approaches that take into account the Edge infrastructure to build predictive

Table 1
Summary and comparison of the related approaches that build predictive models of dynamic QoS of service instances.

	Predictive Decision	QoS attribute(s)	Predictive model		Machine Infrastructure
			Technique	Input variable(s)	
Hasnain et al. (2019)		Response time throughput	Neural networks	Elapsed time	
Wang et al. (2018)		Reliability			
Athanasopoulos and Pernici (2015)		Any	Regression	API	
Guerriero et al. (2019)		Reliability performance	Markov chains	Response rate	Agnostic
Huang et al. (2018)		Hardware resources scheduled tasks	Queuing network	Request rate	
Zhu et al. (2017)		Reliability	Collaborative filtering	Service users elapsed time	
Zheng and Lyu (2010)					
Zheng et al. (2011)				Service users	
Yu et al. (2013)		Any			
Silic et al. (2015)		Reliability			
Ibrahim et al. (2018d)		Any	Machine learning	Service load	
Ibrahim (2018)					
Ibrahim et al. (2018a,c)					
Chen and Bahsoon (2017)		Response time throughput availability reliability		Request rate knobs	
Barna et al. (2017, 2018)	Single objective	Response time CPU		Request rate input data	Cloud
Amannejad et al. (2015a,b)		Response time	Collaborative filtering	Service load elapsed time	
Mukherjee et al. (2020)		Response time	Queuing network	Service load	
Singh et al. (2018)		Response time	Regression	CPU request rate num. of machines	
Bartalos et al. (2016)		Power consumption		Elapsed time memory cache	
Fanjiang et al. (2022)		Any	Genetic programming	Elapsed time	
Athanasopoulos et al. (2019) Athanasopoulos and Liu (2022)		Response time	Regression machine learning	API	Fog
Zhang and Zhang (2005)		Reliability	Petri-nets	Correctness fault tolerance	Agnostic
Zhao et al. (2011)			Structural patterns		
Ding et al. (2016)		Execution time	Workflow patterns		
Zheng et al. (2017)		Response time reliability	Semi-Markov chains	Discrete time series	
AbdelBaky and Parashar (2022)		Any	Workflow patterns	Elapsed time	Cloud
Liu et al. (2022)		Any	Machine learning	Service context	Edge
White and Clarke (2022)		Response time	Echo network	Time series	
Zhang et al. (2022)		Any	Collaborative filtering	User location	
Our approach	*Multi-objective*	*Efficiency (empirical complexity)*	*Regression*	*API*	

models of the QoS of service instances as a function of the input parameters of the operations of the programming interface/API of web services (Athanasopoulos et al., 2019; Athanasopoulos and Liu, 2022). These are two single-objective approaches that build predictive models by using regression analysis and machine learning, respectively. We compare our multi-objective approach against these two baseline approaches. Finally, our proposed approach goes one step further than these two approaches by using predictive models for deciding on whether service instances are Edge-efficient.

2.2.1. Infrastructure-agnostic atomic web services

AI techniques. An existing approach constructs neural networks (a variant of recurrent deep-learning neural network) to predict the response time and the throughput of service instances as a function of the elapsed time of service invocations (Hasnain et al., 2019). Another approach constructs multi-layer neural networks that combine convolution neural networks and long short-term memory to predict the reliability of service instances as a function of the elapsed time of service invocations (Wang et al., 2018). An approach constructs predictive models by using the regression analysis to predict the values of any QoS attribute of service instances as a function of the input parameters of the operations of the programming interface/API of services (Athanasopoulos and Pernici, 2015). Another approach builds predictive models of the reliability and the performance of service instances by using Markov chains of discrete time (Guerriero et al., 2019). The models are expressed as a function of the response rate of service

instances. Finally, an approach builds predictive models of the hardware resources consumed by service instances and of the tasks scheduled for service instances (Huang et al., 2018). These predictive models are built by defining queuing networks and are expressed as a function of the request rate.

Collaborative-filtering techniques. We meet two groups of collaborative-filtering techniques. The first group consists of approaches known as memory-based approaches that use historical data collected from invocations to service instances (Silic et al., 2015). These historical data are organized with respect to service users that have experienced similar values of QoS attributes. For instance, two approaches predict the reliability of service instances for groups of similar users (Zheng and Lyu, 2010; Zheng et al., 2011). Another approach builds a matrix to store historical context-aware reliability values of service instances (Zhu et al., 2017). The context of a service instance is defined with respect to user locations, network connections, hardware resources (e.g. CPU, memory), and time slices.

The second group consists of approaches known as model-based that train predictive models of the relationships that exist among historical service invocations (Silic et al., 2015). For instance, an approach builds predictive models of QoS attribute values experienced by similar users of service instances for making service recommendations (Yu et al., 2013). This approach makes groups of similar users that share common service environment (e.g. communication link, user-service distance) and are expected to experience similar QoS values. Another approach builds collaborative-filtering models that predict the reliability of service instances. The models are trained based on aggregated historical invocations to service instances (Silic et al., 2015).

2.2.2. Cloud/edge-based atomic web services

Cloud-based approaches. Some approaches orchestrate various workload patterns for service instances to build predictive machine-learning models of QoS attributes (Ibrahim et al., 2018d; Ibrahim, 2018; Ibrahim et al., 2018a,c). Another approach uses historical values on the response time, the throughput, the availability, and the reliability of service instances to build predictive machine-learning models as a function of (software or hardware) control knobs of a cloud infrastructure and of the number of the requests to service instances (Chen and Bahsoon, 2017). Other approaches use historical values of the CPU consumption and of the response time of service instances to build predictive models by using Kalman filters (Barna et al., 2017, 2018). These models are expressed as a function of the length of input data to services, the number of the service requests, and the elapsed time of service invocations.

Other approaches determine cloud-management activities (e.g. VM migration) that cause deterioration to the QoS of service instances (Amannejad et al., 2015a,b). These approaches adopt collaborative-filtering techniques applied on historical response-times of service instances to predict future invocations to service instances that suffer from interference with cloud-management activities. The predictions are made as a function of the load of services and the elapsed time of service invocations. Another approach determines internal anomalies (e.g. bugs in service code) and external anomalies (e.g. interference with cloud-management activities) in the execution of multi-tier cloud-based web services (Mukherjee et al., 2020). The approach monitors the response time of services to build queuing models that detect anomalies. Another approach determines the interference with cloud-management activities to multi-tier cloud-based web services for building predictive models of the response time of service instances (Singh et al., 2018).

An approach builds predictive models of the power consumed by service instances by using the technique of the regression analysis (Bartalos et al., 2016). These models are expressed as a function of the percentage of the elapsed time of service invocations and of the number memory cache misses. Finally, a very recent approach builds predictive models of QoS of service instances as a function of the elapsed time of service invocations by using genetic programming (Fanjiang et al., 2022).

Fog/edge-based approaches. There is an approach that builds predictive models of the response time of service instances by using regression analysis. The models are built as a function of the input parameters of the operations of the programming interface/API of services (Athanasopoulos et al., 2019). In a similar way, another approach builds predictive models of the response time of service instances by using machine-learning techniques (Athanasopoulos and Liu, 2022). Another approach builds predictive models of the response time of IoT service instances as a function of discrete time series by using a noisy echo state network-based technique (White and Clarke, 2022). Finally, an approach uses a privacy-preserving collaborative-filtering technique for predicting the QoS of service instances with respect to the geographical locations of service users (Zhang et al., 2022).

2.2.3. Infrastructure-agnostic composite services

An approach builds predictive Petri-net models of the reliability of service instances as a function of the correctness and the fault tolerance of services (Zhang and Zhang, 2005). In a similar way, an approach builds predictive models of reliability based on control structural patterns of composite services (Zhao et al., 2011). Another approach defines transactions for composite services and constructs predictive models of the execution time of service instances as a function of the correctness and the fault tolerance of services (Ding et al., 2016). The predictive models are built based on (sequential and parallel) workflow patterns of composite services. Another approach builds predictive models of the response time and the reliability of composite services (Zheng et al., 2017). The predictive models are built based on semi-Markov chains on control structural patterns for composite services. These models are expressed as a function of discrete time series.

2.2.4. Cloud/edge-based composite web services

Cloud-based approaches. An approach builds predictive models of the QoS of multi-cloud composite service instances (AbdelBaky and Parashar, 2022). The predictive models are built by using (sequential and parallel) workflow patterns of composite services and are expressed as a function of the elapsed time of service invocations.

Edge-based approaches. An approach builds predictive models of the QoS of composite service instances for mobile edge computing (Liu et al., 2022). These models are built by using machine-learning techniques and are expressed as a function of the context of services. The context is defined based on user-related and service-related contextual factors (e.g. service tasks, workload, underlying network).

3. RESTful web services at the edge

The programming interfaces/APIs of web services are specified via using the REST technology (Richardson and Ruby, 2007) or the SOAP technology (Erl, 2016). Lately, RESTful services have received wider adoption than SOAP-based services. For instance, `Amazon`[3] has discontinued/retired the SOAP-based APIs of the Amazon Web services.

[3] www.stoneedge.net/forum/pop_printer_friendly.asp?TOPIC_ID=12687.

RESTful services have received wider adoption because invocations to RESTful services are performed via the REST protocol that uses the standard HTTP methods, e.g. `GET`, `POST`, `PUT`, or `DELETE` (Fielding and Taylor, 2002). Using the HTTP methods, programming clients of a RESTful service send HTTP requests for retrieving, creating, modifying, or deleting resources that have been stored in the machine where a service instance has been deployed. On the other hand, SOAP-based services need the extra steps of creating, sending, and parsing SOAP messages. Given that RESTful services do not need extra middleware, the execution of RESTful services is lighter than the execution of SOAP-based services with respect to the amount of the hardware resources consumed in the machines where the service instances have been deployed. This is the reason that web services at the Edge usually follow the REST technology. The syntactic specification of the programming interface/API of RESTful services is described in Section 3.1.

3.1. Syntactic specification of RESTful APIs

A key principle of the service orientation is the standardized service contract that consists of a set of service descriptions (Erl, 2016). Each description specifies a web service from a different aspect, e.g. a syntactic aspect, a semantic aspect, a behavioral aspect, or a policy-induced aspect (Andrikopoulos et al., 2012).

The syntactic aspect specifies the format and the structure of the messages that can be exchanged with web services for getting programmatic access to the operations that the services provide. In other words, the syntactic specification of web services provide a machine processable description of the messages exchanged with the programming operations of web services. The semantic aspect specifies the concepts implemented by a service like ontologies (e.g. web ontology language (OWL)[4]). The behavioral aspect specifies the conversations/protocols that a service participates to achieve business goals (e.g. BPEL[5]). Finally, the policy-induced aspect mainly specifies the QoS attributes of a service instance.

In this work, we focus on the syntactic aspect of the specification of service descriptions because our approach uses the syntactic specification of the programming operations of a service for learning the empirical complexity of the service. The syntactic description of RESTful APIs can be formally specified in a single document by using various formats/languages such as the web application description language (WADL),[6] the open data protocol (OData),[7] the OpenAPI specification, [8] the RESTful Service Description Language (RSDL),[9] and the RESTful API Modeling Language (RAML).[10] Only the OpenAPI specification has been very recently evolved releasing a new stable version (version 3.1, February 2021).[11] This is the reason that we focus on the OpenAPI specification of RESTful APIs.

An OpenAPI specification (formerly known as Swagger Specification) is an open-source format for describing and documenting RESTful APIs. An OpenAPI specification comes with version 2 or 3 and can be written in JSON[12] or YAML.[13] We use version 3 that is described in Section 3.2.

4 https://www.w3.org/OWL.

5 www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.

6 https://www.w3.org/Submission/wadl.

7 https://www.odata.org/documentation.

8 https://swagger.io/resources/open-api.

9 https://www.balisage.net/Proceedings/vol10/html/Robie01/BalisageVol10-Robie01.html.

10 https://apievangelist.com/2014/03/01/api-design-tooling-from-raml.

11 https://www.openapis.org/blog/2021/02/18/openapi-specification-3-1-released.

12 www.w3schools.com/json.

13 https://yaml.org.

3.2. OpenAPI specification of RESTful APIs

The main sections of an OpenAPI specification in version 3 that our approach uses for learning empirical complexity of a service are the following: (i) information about the underlying service; (ii) a set of paths that correspond to the programming operations of the service; (iii) components of the schemas of the request message and the response message of the operations. Components correspond to parts of schemas that are reusable by the request/response messages of multiple operations of the same OpenAPI specification.

To illustrate the main sections of an OpenAPI specification used by our approach, we revisit our running example of an auction app. We assume the service of the back-end of an auction app makes groups and analyzes bidding data by using a k-means clustering algorithm (Tan et al., 2019). A part of the OpenAPI specification in JSON of the clustering service is depicted in Fig. 1. In particular, lines 2–5 of Fig. 1 correspond to the information section. We assume the clustering service provides a single path that corresponds to the clustering operation (lines 6–28). Lines 29–36 corresponds to the components used by the request message of the operation of the service.

Regarding the information section, it mainly consists of a textual description/title of the service, the categories of the service, the service version, the service provider, and the service name. The information section of the clustering service (lines 2–5 of Fig. 1) includes the title of the service, the service version, and the service name.

In general, the definition of a path of an operation can include: (i) one of the standard HTTP methods; (ii) a textual description of the operation; (iii) the name/ID of the corresponding operation; (iv) an optional set of parameters; (v) a request message; (vi) an optional set of references to components; (vii) a response message; (viii) an optional security scheme of the service. Each parameter is characterized by a name, a description, a primitive data-type (e.g. `int`) or (a reference to) a schema. A request/response message is defined by: (i) the name of the message; (ii) a textual description of the message; (iii) a (possibly empty) set of parameters; (iv) a (possibly empty) set of references to other messages; (v) a (possibly empty) set of references to schemas of messages; (vi) an optional aggregation (e.g. choice) of references to other messages or schemas. A schema of messages is defined by: (i) the name of the schema; (ii) a textual description of the schema; (iii) a set of parameters; (iv) a (possibly empty) set of references to other schemas.

In our example, the clustering operation accepts an input a set of multi-dimensional data points (that will get clustered). The definition of the path of the clustering operation (line 7 of Fig. 1) includes: (i) the HTTP method, `POST` (line 8 of Fig. 1); (ii) a textual description/summary of the operation (line 8 of Fig. 1); (iii) the name/ID of the clustering operation (line 9 of Fig. 1); (iv) two parameters (lines 10–21 of Fig. 1) that correspond to the maximum number of clusters and the maximum number of the iterations/epochs of the main loop of the clustering algorithm (the primitive data-type of the two parameters is defined by two schemas that contain the corresponding data-types); (v) the body of the request message (lines 22–25 of Fig. 1) is defined by a reference to a reusable component (this component is specified below in the section of the components of the OpenAPI specification); (vi) the response message (line 26 of Fig. 1).

In general, the components of an OpenAPI specification contains the definitions of the reusable parts of the request/response messages of the paths and the security schemes of a service. The definition of the components of our clustering service includes a schema (line 29 of Fig. 1) that contains two components (lines 29 and 32 of Fig. 1). The first component, `DataPoints`, corresponds

```
1  {"openapi": "3.0.3",
2      "info": { "title": "Clustering multi-dimension data points using the k-means algorithm",
3                "version": "2022-08-14T00:00:00.000Z",
4                "x-serviceName": "Clustering Service"
5              },
6      "paths": {
7          "/cluster/{clusterNum}/{iterations}/dataPoints": {
8              "post": { "summary": "This operation takes a set of data points as input.",
9                        "operationId": "clusterDataPoints",
10             "parameters": [{ "name": "clusterNum",
11                              "in": "path",
12                              "description": "",
13                              "required": true,
14                              "schema": { "type": "integer" } },
15                            { "name": "iterations",
16                              "in": "path",
17                              "description": "",
18                              "required": true,
19                              "schema": { "type": "integer" }
20                            }
21                           ],
22             "requestBody": { "description": "",
23                              "content": { "application/json":
24                              { "schema": { "$ref": "#/components/schemas/DataPoints" } } }
25                            },
26             "responses": { "default": { "description": "successful operation" } } }
27         }
28     },
29     "components": { "schemas": { "DataPoints": { "type": "array",
30                                                  "items": { "$ref": "#/components/schemas/
                                                      DataPoint" }
31                                                },
32                                  "DataPoint": { "type": "array",
33                                                 "items": { "type": "number",
34                                                            "format": "double64",
35                                                            "example": 10.567 } } } }
36 }
```

Fig. 1. The part of the OpenAPI specification in JSON of the clustering service used by the empirical complexity of the service.

to a set of multi-dimensional data-points provided as input to the clustering operation. The data-type of this component is specified as an one-dimension array (line 29 of Fig. 1). Each item of the array is defined as a reference to the second component (line 30 of Fig. 1). The second component, `DataPoint`, corresponds to a multi-dimensional data-point. The data-type of this component is specified as an one-dimension array (line 32 of Fig. 1). Each item of the array is defined as a number (line 33 of Fig. 1). The data-type/format of each item corresponds to a number of a double precision (line 34 of Fig. 1).

We define in Section 3.3 a service model of OpenAPI specifications for empirical complexity that is independent of the syntax of OpenAPI specifications.

3.3. Service model for empirical complexity

The part of an OpenAPI specification that our approach uses for learning empirical complexity of a service instance is the set of the paths that correspond to the programming operations of the service, as defined below.

Definition 1 (*Service*)**.** We define a service by using the following tuple that comprises the name of a service and the set of the operations of the service:

$$s = (n,\ \{op_i\}) \tag{1}$$

Our example clustering service is modeled by the tuple, $s =$ ('Clustering Service', $\{op\}$) that comprises the name of the service and the single operation of the service.

As described in Section 3.2, the input of each operation can include parameters, a request message, and references to messages/components. Due to the fact that references to messages/schemas can be used, the input of an operation forms a hierarchical structure of parameters, messages, schemas, and components rooted at the operation. The leaves of this hierarchical structure correspond to the nodes that have primitive data-types. To learn the empirical complexity of an operation, our approach uses the leaves of the hierarchical structure because leaf nodes carry actual data provided as input to an operation. In other words, leaf nodes capture the empirical utility of an operation. Our approach further uses the multiplicity of the (internal or leaf) nodes of the hierarchical structure. The multiplicity of input data further determines the empirical utility of an operation.

To illustrate such a hierarchical structure, we revisit the clustering operation of our running example. The nodes of the hierarchical structure that specify the names and the data-types of the parameters, the request message, the schemas, and the components are depicted in Fig. 2. We following post-process this hierarchical structure to filter out the leaf nodes and the nodes with multiplicity higher than one.

Definition 2 (*Service operation*)**.** We define a service operation by using the following tuple that consists of the name of the operation and the set of the features of the operation that correspond to the leaf nodes of the hierarchical structure of the input of the operation and to the (internal/leaf) nodes with multiplicity higher than one.

$$op = (n,\ \{f_i\}) \tag{2}$$

A feature of the input of an operation is defined below.

Definition 3 (*Input feature*)**.** We define a feature of the input of an operation by using the following tuple that comprises the name, the primitive data-type of the feature, and the multiplicity of the feature. If a feature corresponds to an internal node of

```
"clusterDataPoints": {
    "parameters": [{ "name": "clusterNum",
                     "schema": { "type": "integer" } },
                   { "name": "iterations",
                     "schema": { "type": "integer" } } ],
    "requestBody": { { "schema": { "$ref": "#/components/schemas/DataPoints" } } }
    "components": { "schemas": { "DataPoints": { "type": "array",
                                                 "items": { "$ref": "#/components/schemas/
                                                     DataPoint" }},
                                 "DataPoint": { "type": "array",
                                                "items": { "type": "number",
                                                           "format": "double64",
                                                           "example": 10.567 } } } }
}
```

Fig. 2. The hierarchical structure of the name and data-type of the input elements of the clustering operation in JSON format.

```
{"clusterDataPoints": [{
    "f1": { "name": "clusterNum",
            "data-type": "integer",
            "multiplicity": "integer"
          },
    "f2": { "name": "iterations",
            "data-type": "integer",
            "multiplicity": "integer"
          },
    "f3": { "name": "DataPoints",
            "data-type": "array",
            "multiplicity": "integer"
          },
    "f4": { "name": "DataPoint",
            "data-type": "number",
            "multiplicity": "integer"
          } } ]
}
```

Fig. 3. The input features of the clustering operation.

the hierarchical structure of the input of the operation, then the data-type of the feature is empty.

$$f = (n,\ t,\ u) \tag{3}$$

To illustrate the input features of an operation, Fig. 3 depicts the four features of the clustering operation that includes the three leaves (`clusterNum`, `iterations`, `items` of `DataPoint`) of the hierarchical structure of the input. The node, `DataPoint`, is included in the features of the operation not only because it is a leaf node in the hierarchical structure, but also because its multiplicity is higher than one. The data-type of the `items` of the leaf node, `DataPoint`, is a `number`. The multiplicity of the `items` in this node is determined by the length of the corresponding `array`. The fourth feature corresponds to the internal node, `DataPoints`, that has multiplicity higher than one. The multiplicity of the `items` in this node is determined by the length of the corresponding `array`.

Overall, the clustering operation is modeled by the tuple, $op =$ (‘`clusterDataPoints`’, $\{f_1, f_2, f_3, f_4\}$), that consists of the name of the operation and the set of the following input features: $f_1 =$ (‘`clusterNum`’, `integer`, 1), $f_2 =$ (‘`iterations`’, `integer`, 1), $f_3 =$ (‘`DataPoints`’, `integer`, u_1), and $f_4 =$ (‘`DataPoint`’, `number`, u_2), where u_1 and u_2 equals to the length of the corresponding arrays of data points and dimensions of data points, respectively.

3.4. Edge-efficient service instance

In general, an infrastructure of machines at the Edge includes two groups of devices (Hong and Varghese, 2019). The first group consists of devices where the front-end of mobile apps is installed (e.g. end-users’ devices). The second group consists of devices used for deploying service instances of the back-end of mobile apps. These devices provide physical/virtual machines for deployment purposes and are usually more powerful than the machines of the first group. The communication between the devices of the first group and the second group is usually performed via using one-hope (or few-hope) wireless connections that have negligible network latency. In this work, we focus on machines that belong to the second group.

A machine at the Edge is characterized by a set of hardware resources. We assume that each hardware resource is characterized by a *critical value* of the empirical complexity of a service instance. A critical value is the value for which a service instance fully consumes the hardware resource, as defined below.

Definition 4 (*Machine at the Edge*)**.** A machine at the Edge is characterized by an array of the critical values of the empirical complexity of a service instance that lead to the full consumption of the hardware resources of the machine by the service instance.

$$m = \text{HR}[p], \quad \text{where p is a hardware resource} \tag{4}$$

To determine the critical consumption of a hardware resource of a machine by a service instance, the empirical complexity of the service instance for the hardware resource should be first learnt. Then, the critical consumption of the hardware resource by the service instance can be determined by monitoring the consumption of the hardware resource by the service instance, along with the values of the expression of the empirical complexity of a service instance for the hardware resource.

We further define below the concept of a service instance that is deployed to a machine at the Edge.

Definition 5 (*Service instance at the Edge*)**.** We define a service instance that has been deployed to a machine at the Edge by the following tuple that consists of the URI of the endpoint of the service instance, the service model (Definition 1), the machine of the service instance (Definition 4), and the set of the predictive models, $\{\widehat{C}_{op,p,m}\}$, of the empirical complexity learnt for each operation of the service instance (Definition 17).

$$si = (uri,\ s,\ m,\ \{\widehat{C}_{op,p,m}\}) \tag{5}$$

The empirical complexity of a service instance is defined not only in terms of the machine to which the service instance has been deployed, but also it is defined in terms of a metric, p, of the values of a hardware resource of the machine. This metric is specified in Definition 12. The concept of the empirical complexity is formally defined in Section 5. We illustrate the concept of the empirical complexity by revisiting the clustering service in Section 5.

Given that an empirical complexity is defined with respect to multiple hardware resources, the empirical complexity is captured by a multi-objective expression. We consider an operation

of a service instance is Edge-efficient if the value of each objective of the empirical complexity of the operation is lower than the critical value of the corresponding objective, as defined below.

Definition 6 (*Edge-efficient operation*)**.** An operation of a service instance is predicted to efficiently run in a machine at the Edge for an invocation to the operation with respect to each hardware resource of the machine, if the value of the objective of the empirical complexity of the invocation to the operation that corresponds to the hardware resource is lower than the critical value of the hardware resource.

$$\forall p: \ \widehat{C}_{op,p,m} < m.HR[p] \tag{6}$$

Prior to defining the concept of the multi-objective empirical complexity of service instances (Section 5), we describe the fundamentals on the single-objective empirical complexity of programs (Section 4).

4. Fundamentals on empirical complexity

4.1. Single-objective empirical complexity

The literature uses the term of the *location* to refer to a set of lines of a program that play a major role in the empirical complexity of the program. For instance, a loop increases the execution time of a program as opposed to an `if` statement. Moreover, the literature uses the term of the *actual cost* to refer to the measurement of the cost of a location. The cost of a location is measured by counting the number of the times that the location is executed in a single run of a program. For instance, the cost of a loop equals to the number of the iterations of the loop (Goldsmith et al., 2007).

The literature uses the term of the *workload* to refer to the data that are provided as input to a program at a single run of the program (Goldsmith et al., 2007). The workload is specified in terms of a set of features (Definition 7). The term of the *feature* refers to the input parameters of a program (Goldsmith et al., 2007). For example, a feature can correspond the number of the iterations of a loop. To measure the empirical complexity, the cost of a location should be measured not only on a single run but also on multiple runs. The literature uses the term of the *bag* of workloads to refer to the values of the input features of a program at multiple runs of the program (Definition 8).

Definition 7 (*Workload of a program*)**.** We define the workload of a single run of a program as the vector, $w[M]$. The elements of the vector contain values for the M features provided as input to the program in the current run.

f_1	f_2	...	f_M
$w[1]$	$w[2]$	...	$w[M]$

Definition 8 (*Bag of workloads*)**.** We define the workloads of the N runs of a program as a two-dimension array, $W[N, M]$. The first dimension of the array corresponds to the N workloads that are provided as input at N runs of the program. The second dimension corresponds to the M features of the program.

	f_1	f_2	...	f_M
Run 1	*W[1, 1]*	*W[1, 2]*	...	*W[1, M]*
Run 2	*W[2, 1]*	*W[2, 2]*	...	*W[2, M]*
...	...	...	...	...
Run N	*W[N, 1]*	*W[N, 2]*	...	*W[N, M]*

Definition 9 (*Actual costs of a program*)**.** We define the actual costs of the locations of a program as a two-dimension array, $C[N, K]$. The first dimension of the array corresponds to the N workloads provided as input at N runs of the program for the M features of the program. The second dimension corresponds to the K locations of the program.

	l_1	l_2	...	l_K
$W[1, 1] \dots W[1, M]$:	*C[1, 1]*	*C[1, 2]*	...	*C[1, K]*
$W[2, 1] \dots W[2, M]$:	*C[2, 1]*	*C[2, 2]*	...	*C[2, K]*
...	...	...	...	...
$W[N, 1] \dots W[N, M]$:	*C[N, 1]*	*C[N, 2]*	...	*C[N, K]*

4.2. Single-objective predictive models

An approach that constructs a predictive model of the empirical complexity of a program takes as input the actual costs of the program on historical workloads and learns how to predict the costs of the program on unseen workloads.

Definition 10 (*Predicted costs of a program*)**.** We define the predicted costs of the locations of a program as a two-dimension array, $\widehat{C}[M, K]$ that is learnt by using an array, $C[N, K]$, of the actual costs of the locations of the program (Definition 9). The first dimension of the array corresponds an unseen workload, $\widehat{w}[M]$, that contains values for the M features of the program. The second dimension corresponds to the K locations of the program.

Learning costs	l_1	l_2	...	l_K
$W[1, 1] \dots W[1, M]$:	*C[1, 1]*	*C[1, 2]*	...	*C[1, K]*
$W[2, 1] \dots W[2, M]$:	*C[2, 1]*	*C[2, 2]*	...	*C[2, K]*
...	...	...	...	...
$W[N, 1] \dots W[N, M]$:	*C[N, 1]*	*C[N, 2]*	...	*C[N, K]*
Predicting costs	l_1	l_2	...	l_K
$\widehat{w}[1]$:	$\widehat{C}[1, 1]$	$\widehat{C}[1, 2]$	...	$\widehat{C}[1, K]$
$\widehat{w}[2]$:	$\widehat{C}[2, 1]$	$\widehat{C}[2, 2]$	...	$\widehat{C}[2, K]$
...	...	...	...	...
$\widehat{w}[M]$:	$\widehat{C}[M, 1]$	$\widehat{C}[M, 2]$	...	$\widehat{C}[M, K]$

In general, an approach that learns to predict costs can be defined to make predictions for a combination of features or for a dominant feature exclusively. However, Definition 10 is applied to make predictions for one feature at a time. We use this definition because the number of the combinations of features can be prohibitively high (e.g. exponential). Thus, before starting making predictions, the dominant feature should first selected for each location of a program. Then, predictions can be made for each location of a program with respect to the dominant feature exclusively. The concept of the dominant feature is defined below.

Definition 11 (*Dominant feature of a program*)**.** We define the dominant feature of a location of a program as the feature, f_d, that gives the lowest prediction errors over all the historical workloads, $W[N, M]$, among all the features. We calculate the prediction error of a feature, f, on a historical workload, $W[i, 1] \dots W[i, M]$, for a location, l, as the supplement of the following fraction: the minimum cost between the actual cost and the predicted cost for this location divided by the maximum cost.

$$e_{i,l,f} = 1 - \frac{min(C[i, l], \widehat{C}[f, l])}{max(C[i, l], \widehat{C}[f, l])} \tag{7}$$

Finally, the dominant feature for a location, l, is the feature that has the minimum sum of prediction errors over all the historical workloads, as follows.

$$\text{Find } f_d \in [1, M]: \ \sum_{i}^{N} e_{i,l,f_d} \text{ is minimum.} \tag{8}$$

Table 2
Families of growth functions in computational complexity.

Family name	Mathematical expression
Constant function	$\widehat{C}_1[i,j] = a$
Poly-logarithmic function	$\widehat{C}_2[i,j] = \log(\widehat{W}[i,j])^a$
Polynomial function	$\widehat{C}_3[i,j] = a_1 * (\widehat{W}[i,j])^{a_2}$
Exponential function	$\widehat{C}_4[i,j] = a^{\widehat{W}[i,j]}$

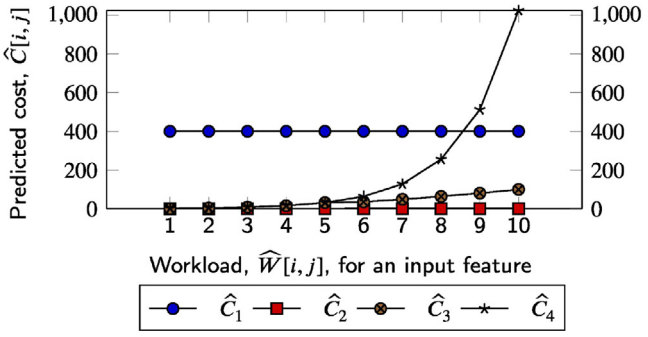

Fig. 4. The growth rates of the main families of functions.

Learning a predictive model. According to Definition 10, a predictive model of empirical complexity should be learnt from the historical workloads and actual costs of a program. The idea behind constructing a predictive model is that the actual costs of the locations of a program should get correlated with the historical workloads for each input feature of the program. In other words, the dependent variables of a predictive model correspond to the costs of the locations of a program. The independent variables correspond to the workloads, the locations, and the input features of a program.

To correlate independent variables with dependent variables, a mathematical function for each feature should be learnt to calculate the cost of a location in terms of the workloads of each feature. In other words, a function should fit the costs of a location with the workloads for a feature. A function can be following used to predict costs of a location on unseen workloads for a feature.

To learn a mathematical function, we consider as the values of the workloads for a feature grow, the costs of the locations can grow too. For example, as the length of an input vector grows, the cost (e.g. the number of the iterations) of a loop that iterates over the vector grows too. Thus, we can learn a mathematical expression by learning the *growth function* of the cost of a location with respect to a feature as the values of the feature used in the location grows.

According to the fundamental theory of the computational complexity (Papadimitriou, 1994), a growth function, $\widehat{C}$, can generally belong to various families of mathematical functions. The families of functions that cover the main cases of growth rates are presented in Table 2. The variable, $\widehat{W}[i,j]$, in Table 2 refers to the growth function for the workload of the jth location and for the ith feature. The coefficients, a, a_1, and a_2, correspond to constant numbers.

Regarding the families of functions of Table 2, a constant function captures the case that there is no growth in terms of the variable, $\widehat{W}$. A (poly-)logarithmic function captures a (poly-)logarithmic growth with respect to (a power of) $\widehat{W}$. A polynomial function captures a linear growth, or a quadratic growth, or a cubic growth or growth of any other degree. Finally, exponential functions have a higher growth rate than polynomial functions. Exponential functions are practically traceable only on very small values of $\widehat{W}$. An example of a function for each family is presented in Fig. 4. The functions of the examples are the following: $\widehat{C}_1[i,j] = 400$, $\widehat{C}_2[i,j] = \log_{10}\widehat{W}[i,j]$, $\widehat{C}_3[i,j] = \widehat{W}[i,j]^2$, and $\widehat{C}_4[i,j] = 2^{\widehat{W}[i,j]}$, where the values of $\widehat{W}[i,j]$ belong to the interval, $[1, 10]$, of integers for all functions. Having learnt a function from each family, a function that best fits to historical workflows and actual costs can be selected for each feature of a program.

We adapt in Section 5 the fundamentals on empirical complexity of programs in the case of service instances.

5. Empirical complexity of service instances

We first define the concept of the multi-objective empirical complexity of service instances (Section 5.1). Following, we specify the concept of the multi-objective predictive model of empirical complexity (Section 5.2).

5.1. Multi-objective empirical complexity

We adapt below the terms used in empirical complexity from the case of programs to the case of service instances.

Locations of source code of service instances. We use the term of the location to refer to the whole source code of the implementation of an operation of a service instance because the number of the locations of an operation can be very high (a single operation usually implements a part of a software system). In general, if the number of locations is very high, then it is known that the calculation of empirical complexity is induced to the insolvable Halting problem (Papadimitriou, 1994).

Workload and features of service operations. The term of the workload in the case of a service operation refers to the data provided as input for the input features of the operation. The term of the feature in the case of a service operation refers to an input parameter of the operation (Definition 3). Regarding the term of the workload, we reuse Definition 7 in the case of service operations. In the same vein, we reuse Definition 8 in the case of service operations.

Cost of source code of service operations. We extend Definition 9 of the term of the actual cost of a program because the term of the location in a service operation refers to the whole source code of the operation. Consequently, the workload of an operation is not necessarily the same with the workload of another operation. In other words, the term of the cost should be defined for each operation independently of other operations. In particular, we use the term of the actual cost to refer to the runtime measurement of a hardware resource of a machine consumed by an operation invocation. Given that multiple hardware resources can be consumed by the same operation invocation, we define below the metric of a multi-objective hardware resource.

Definition 12 (*Multi-objective hardware resource*)**.** The multi-objective hardware resource of a service operation consists of the measurements of the actual consumption of multiple hardware resources by operation invocations.

$$C_p = \begin{cases} ex & \text{if } p = \text{execution time in CPU} \\ en & \text{if } p = \text{energy consumption in RAM} \\ \dots & \text{if } p = \text{another metric} \end{cases} \quad (9)$$

The multi-objective metric in Definition 12 is extensible because it is parameterized to use multiple hardware resources. Definition 12 uses two indicative metrics of hardware resources: (i) the execution time in CPU (Definition 13) and (ii) the energy consumption in RAM (Definition 14) of an invocation to an operation. The execution time of an operation in CPU is indicative of the time that the source code of an operation keeps occupied the CPU of a machine, as defined below.

Definition 13 (*Execution time in CPU*)**.** The execution time of an operation in CPU equals to the elapsed time from the moment the operation starts being executed in a machine until the moment the operation terminates its execution.

However, the metric of the execution time cannot capture the amount of the RAM consumption. Moreover, an operation may use a big amount of RAM in order to decrease its execution time. For instance, a big amount of data can be stored in RAM for retrieving them in a very short period of time during the execution of an operation. In this case, while an operation consumes a small amount of CPU, it consumes a big amount of RAM. To further capture the amount of the RAM consumption, we use the metric of the energy consumed by an invocation to an operation. The bigger the amount of the RAM consumption is, the higher the energy consumption is.

Definition 14 (*Energy consumption in RAM*)**.** The energy in RAM consumed by the execution of an operation equals to the total amount of the energy consumed in RAM by the source code of the operation from the moment the operation starts being executed in a machine until the moment the operation terminates its execution.

A technical restriction in using the metric of the energy consumption is that the source code of an operation should be accessible for setting up the monitoring of the energy consumption. To this end, a monitoring agent executed in parallel with the execution of service operations can be used. However, an agent should monitor the energy consumption at the level of the operations of a service implementation. To achieve this, we use the Java agent, `Jalen` (Noureddine et al., 2015) that provides fine-grained monitoring capabilities at the level of the functions of programs.

Having specified the metrics we use for collecting the actual consumption of hardware resources, we define below the term of the actual cost of an operation. Given that the actual cost of an operation is related to a specific instance of the corresponding service deployed to a machine at the Edge (Definition 5), we define the actual cost in terms of the machine to which the service instance has been deployed.

Definition 15 (*Actual cost of service operation*)**.** We define the actual costs of an operation of a service instance in terms of a metric, p, of a hardware resource of a machine, m, where the service instance is deployed, as an one-dimension array, $C_{op,p,m}[N]$. The single dimension of the array corresponds to the N workloads provided as input for the M features of the operation at N invocations to the operation. The elements of the array contain values of the metric at each one of the N invocations to the operation.

$W[1,1]\dots W[1,M]$:	$C_{op,p,m}[1]$
$W[2,1]\dots W[2,M]$:	$C_{op,p,m}[2]$
$\dots$	$\dots$
$W[N,1]\dots W[N,M]$:	$C_{op,p,m}[N]$

While the actual cost of a program is defined as a two-dimension array (Definition 9), the actual cost of an operation is defined as an one-dimension array (Definition 15). The reason of this difference is because the actual cost of a program is defined not only in terms of workloads (first dimension), but also in terms of the locations of the program (second dimension). However, given that each operation is characterized by one location, the actual cost of an operation is defined by using a single dimension that corresponds to the workloads in the invocations to the operation. We finally define below how we can decide on whether an operation is Edge-efficient.

Definition 16 (*Edge-efficient OpenAPI operation*)**.** A service operation efficiently runs in a machine at the Edge if for each invocation, i, to the operation, the actual cost of the operation for each hardware resource of the machine is lower than the critical value of the hardware resource.

$$\forall p,\ \forall i:\ C_{op,p,m}[i] < m.\mathrm{HR}[p] \tag{10}$$

5.2. Multi-objective predictive models

To construct a multi-objective predictive model of the empirical complexity of an operation, our approach takes historical costs of the operation as input and learns to predict costs (Definition 17) on unseen workloads, as follows.

Definition 17 (*Predicted cost of an operation*)**.** We define the predicted cost of an operation in terms of a metric of hardware resource of a machine at the Edge, as an one-dimension array, $\widehat{C}_{op,p,m}[M]$ that is learnt by using an array, $C_{op,p,m}[N]$, of the actual historical costs of the operation (Definition 15). The single dimension of the array corresponds to unseen workloads, $\widehat{w}$, that are provided as input for each one of the M features of the operation.

Learning costs	$W[1,1]\dots W[1,M]$:	$C_{op,p,m}[1]$
	$W[2,1]\dots W[2,M]$:	$C_{op,p,m}[2]$
	$\dots$	$\dots$
	$W[N,1]\dots W[N,M]$:	$C_{op,p,m}[N]$
Predicting costs	$\widehat{w}[1]$:	$\widehat{C}_{op,p,m}[1]$
	$\widehat{w}[2]$:	$\widehat{C}_{op,p,m}[2]$
	$\dots$	$\dots$
	$\widehat{w}[M]$:	$\widehat{C}_{op,p,m}[M]$

To illustrate a multi-objective predictive model of the empirical complexity of an operation, we revisit our running example of the clustering operation. As described in Section 3.3, the input features of the clustering operation are the following: f_1 = (‘`clusterNum`’, `integer`, 1), f_2 = (‘`iterations`’, `integer`, 1), f_3 = (‘`DataPoints`’, `integer`, u_1), and f_4 = (‘`DataPoint`’, `number`, u_2), where u_1 and u_2 equals to the length of the corresponding arrays of data points and dimensions of data points, respectively. We indicatively present in Fig. 5 three historical workloads for the four features of the input features. We have also included the actual execution time of the operation with respect to each input feature. Finally, we present an unseen workload for the features and the predicted cost for each feature. The mathematical function that is learnt for each feature based on the three historical workloads and is used for calculating a predicted cost is presented in Section 6.

As mentioned in Section 4.2, our approach first learns to predict a cost for each feature independently of the other features. Then, our approach uses the predicted costs of the features to select a dominant feature that is defined below.

Definition 18 (*Dominant feature of an operation*)**.** We define the dominant feature of an operation in terms of a metric of a hardware resource of a machine at the Edge, as the feature, f_d, for which the prediction errors are the lowest over all the historical workloads, $W[N,M]$, among all the features. We calculate the prediction error of a feature, f, on a historical workload, $W[i,1]\dots W[i,M]$, as the minimum cost between the actual cost and the predicted cost divided by the maximum cost.

$$e_{i,f,op,p,m} = \frac{min(C_{op,p,m}[i], \widehat{C}_{op,p,m}[f])}{max(C_{op,p,m}[i], \widehat{C}_{op,p,m}[f])} \tag{11}$$

Actual costs	100 1000 948 3:	337.3
	100 1000 5739 7:	8011.2
	100 1000 10000 8:	40327.1
Predicted costs	100:	40327.1
	1000:	40327.1
	10000:	32112.8
	8:	25146.1

Fig. 5. Historical workloads and actual/predicted costs for the features of the clustering operation of our running example.

The dominant feature is the feature that has the minimum sum of prediction errors over all the historical workloads compared to the other features, as follows.

$$\text{Find } f_d \in [1, M] : \sum_{i}^{N} e_{i,f,op,p,m} \text{ is minimum.} \quad (12)$$

We apply Definition 18 to the previous example of the workloads and the actual/predicted costs of the clustering operation. The feature, f_1 = (‘`clusterNum`’, `integer`, `1`), f_2 = (‘`iterations`’, `integer`, `1`), f_3 = (‘`DataPoints`’, `integer`, u_1), and f_4 = (‘`DataPoint`’, `number`, u_2), where u_1 and u_2 equals to the length of the corresponding arrays of data points and dimensions of data points, respectively., is the dominant feature because the predicted cost for this feature is the minimum among all the features.

The algorithms for learning the growth functions of a multi-objective predictive model and for selecting dominant features are described in Section 7.

6. Learning & selecting growth functions

Our approach first learns a candidate growth function for each family of functions (Table 2). To learn the mathematical expression of a growth function, we use *regression analysis* that learns the correlation between a dependent variable and one or more independent variables (Hazewinkel, 2001). The most commonly used form of regression analysis is the linear regression that can determine closed-form solutions. It means that the linear regression builds mathematical expressions in a finite number of steps (Hazewinkel, 2001). A widely used technique of linear regression is the method of the least squares (Least-Square Technique (LST)) that minimizes the sum of the squared differences between actual measurements and predicted values.

We specify in Section 6.1 the way in which we employ LST for learning the mathematical expression of a growth function for an operation. We further specify in Section 6.2 the algorithmic steps that our approach follows for learning and deciding a growth function for an operation.

6.1. Employing linear least-square technique

Our approach employs LST to each family of functions (Table 2). The way that our approach employs LST is specified in Definitions 19–22. Apart from the constant function, Definitions 20–22 first convert a non-linear function (e.g. a poly-logarithmic function) to a linear function, as expected by the linear regression. Moreover, each definition applies LST to learn a separate growth function for each feature of an operation. As mentioned in Section 5.2, our approach constructs a separate growth function for each feature.

Definition 19 (*LST on a Constant Function*)**.** LST is applied on a constant function, $\widehat{C}_{op,p,m}[f] = a$, by determining the coefficient, a, that minimizes the sum of the squared errors between the predicted costs, $\widehat{C}_{op,p,m}[f]$, for a feature, f, of an operation and the actual costs, $C_{op,p,m}[1] \ldots C_{op,p,m}[N]$, of N invocations to the operation for the feature.

$$\text{Find a: } \sum_{i=1}^{N}\Big(C_{op,p,m}[i](W[i,f]) - \widehat{C}_{op,p,m}[f](W[i,f])\Big)^2 = \sum_{i=1}^{N}\Big(C_{op,p,m}[i](W[i,f]) - a\Big)^2 \text{ is min.}$$

Definition 20 (*LST on a Poly-Logarithmic Function*)**.** We employ the log operator on a poly-logarithmic function, $\widehat{C}_{op,p,m}[f] = \log(W[i,f])^a$, for converting the function to a linear function. We then apply LST on N pairs ($i \in [1, N]$) of values, ($\log C_{op,p,m}[i]$, $\log(\log W[i,f])$), for a feature, f, of an operation (N invocations to the operation) to determine the coefficient, a, that minimizes the sum of the squared errors between the predicted costs, $\widehat{C}_{op,p,m}[f]$, for the feature, f, and the actual costs, $C_{op,p,m}[1] \ldots C_{op,p,m}[N]$, of the N invocations for the feature.

$$\text{Find a: } \sum_{i=1}^{N}\Big(\log(C_{op,p,m}[i](W[i,f])) - \log(\widehat{C}_{op,p,m}[f](W[i,f])))^2 = \sum_{i=1}^{N}\Big(\log(C_{op,p,m}[i](W[i,f])) - a * \log(\log(W[i,f]))\Big)^2 \text{ is min.}$$

Definition 21 (*LST on a Polynomial Function*)**.** We employ the log operator on a polynomial function, $\widehat{C}_{op,p,m}[f] = a_1 * (W[i,f])^{a_2}$, for converting the function to a linear function. We then apply LST on N pairs ($i \in [1, N]$) of values, ($\log C_{op,p,m}[i]$, $\log W[i,f]$), for a feature, f, of an operation (N invocations to the operation) to determine the coefficients, a_1 and a_2, that minimize the sum of the squared errors between the predicted costs, $\widehat{C}_{op,p,m}[f]$, for the feature, f, and the actual costs, $C_{op,p,m}[1] \ldots C_{op,p,m}[N]$, of the N invocations for the feature.

$$\text{Find } a_1, a_2\text{: } \sum_{i=1}^{N}\Big(\log(C_{op,p,m}[i](W[i,f])) - \log(\widehat{C}_{op,p,m}[f](W[i,f]))\Big)^2 = \sum_{i=1}^{N}\Big(\log(C_{op,p,m}[i](W[i,f])) - (\log a_1 + a_2 * \log(W[i,f]))\Big)^2 \text{ is min.}$$

Definition 22 (*LST on a Exponential Function*)**.** We employ the log operator on an exponential function, $\widehat{C}_{op,p,m}[f] = a^{W[i,f]}$, for converting the function to a linear function. We apply LST on N pairs of values, ($\log C_{op,p,m}[i]$, $W[i,f]$), for a feature, f, of an operation (N invocations to the operation) to determine the coefficient, a, that minimizes the sum of the squared errors between the predicted costs $\widehat{C}_{op,p,m}[f]$, for the feature, f, and the actual costs,

Actual costs	100 1000 948 3:	337.3
	100 1000 5739 7:	8011.2
	100 1000 10000 8:	40327.1
Functions	$\widehat{w}[1]$:	$40327.1 * \widehat{w}[1]^{0.0}$
	$\widehat{w}[2]$:	$40327.1 * \widehat{w}[2]^{0.0}$
	$\widehat{w}[3]$:	$0.00045 * \widehat{w}[3]^{1.96}$
	$\widehat{w}[4]$:	$2.3 * \widehat{w}[4]^{4.47}$

Fig. 6. Mathematical functions learnt for the features of the clustering operation of our running example.

Prediction errors	$\widehat{w}[1]$:	0.30
	$\widehat{w}[2]$:	0.30
	$\widehat{w}[3]$:	0.12
	$\widehat{w}[4]$:	0.10

Fig. 7. Prediction errors of the predictive functions for the features of the clustering operation of our running example.

$C_{op,p,m}[1] \ldots C_{op,p,m}[N]$, of the N invocations for the feature.

$$\text{Find a: } \sum_{i=1}^{N}\Big(\log(C_{op,p,m}[i](W[i,f])) - \log(\widehat{C}_{op,p,m}[f](W[i,f]))\Big)^2 = \sum_{i=1}^{N}\Big(\log(\widehat{C}_{op,p,m}[f](W[i,f])) - \log a * W[i,f]\Big)^2 \text{ is min.}$$

6.2. Algorithm for learning growth functions

We move on to the algorithmic steps (Alg. 1) that our approach follows for first learning a separate growth function for each family of functions and then deciding on a growth function for an operation. Alg. 1 accepts as input an array, $W[N, M]$, of N historical workloads for the M features of an operation of a service instance, an array, $C_{op,p,m}[N]$, of historical costs of the operation for the N workloads in terms of a metric, p, of a hardware resource of a machine, m, where to which the service instance is deployed. Alg. 1 returns as output one of the element of the array, $\widehat{C}_{op,p,m}[M]$, of the growth functions learnt for the operation. This element corresponds to the dominant feature, f_d, of the operation. To this end, Alg. 1 passes through the following sequentially executed steps: `Learning Growth Functions` and `Selecting Dominant Feature`.

Learning growth functions. Alg. 1 iterates over the features of an operation (Alg. 1 (line 1)). For each feature, Alg. 1 applies LST to learn a separate growth function (Alg. 1 (lines 2–9)). In other words, Alg. 1 applies Definitions 19–22. Then, Alg. 1 decides on the growth function that fits best to the historical workloads for each feature. To this end, Alg. 1 calculates the sum of the predictions errors of the growth functions on the historical workloads for each feature and selects the growth function that exhibits the lowest sum of prediction errors (Alg. 1 (lines 11–17)). To calculate a prediction error, Alg. 1 applies Eq. (11).

To illustrate the learning of growth functions, we revisit our running example of the clustering operation. As described in Section 3.3, the input features of the clustering operation are the following: $f_1 =$ (‘`clusterNum`’, `integer`, 1), $f_2 =$ (‘`iterations`’, `integer`, 1), $f_3 =$ (‘`DataPoints`’, `integer`, u_1), and $f_4 =$ (‘`DataPoint`’, `number`, u_2), where u_1 and u_2 equals to the length of the corresponding arrays of data points and dimensions of data points, respectively. We indicatively present in Fig. 6 three historical workloads for the four features of the input features. We have also included the actual execution time of the operation with respect to each input feature. Finally, we present the mathematical function that is learnt for each feature based on the three historical workloads and is used for calculating a predicted cost on an unseen workload for each feature.

Selecting dominant feature. Alg. 1 iterates over the growth functions of all the features and decides on the growth function of a feature that fits best to the historical workloads. In particular, Alg. 1 calculates the sum of the prediction errors of the growth functions on the historical workloads for each feature and selects the growth function that exhibits the lowest sum of predictions errors (Alg. 1 (lines 21–27)). In other words, Alg. 1 applies Definition 18.

To illustrate the selection of a dominant feature, we revisit our running example of the clustering operation. We indicatively present in Fig. 7 the prediction error for each feature. The errors have been calculated on the unseen workload by using the mathematical functions learnt for each feature and presented above. Based on these errors, the dominant feature is the fourth feature (‘`DataPoint`’) that has the minimum error.

7. Deciding on edge-efficient operation

We specify the algorithmic steps (Alg. 2) of deciding whether a service operation can efficiently run in a machine at the Edge. Alg. 2 accepts as input a set of the growth functions, $\{\widehat{C}_{op,p_i,m}[f_d]\}$, of a dominant feature, f_d, for each hardware resource of a machine, an unseen workload, $\widehat{w}[f_d]$, for the dominant feature (the unseen workload corresponds to one invocation to the operation), and an array, HR, of the critical values of the hardware resources of the machine. Alg. 2 returns as output the decision of whether the operation is Edge-efficient (`Boolean` decision). To this end, Alg. 2 passes through the following steps.

Alg. 2 iterates over all the hardware resources (Alg. 2 (line 1)). For each hardware resource, Alg. 2 compares the value of the growth function for the unseen workload of the dominant feature against the critical value of the hardware resource. If the value of each growth function is lower than the critical value of each hardware resource, then Alg. 2 makes a positive decision on the Edge-efficiency of the operation (Alg. 2 (line 6)). If at least one of the values of the growth functions is higher than or equal to the corresponding critical value, then Alg. 2 makes a negative decision (Alg. 2 (lines 2–4)).

8. Evaluation of experiment

We developed a research prototype, `EmpCom`, of our approach in Java. We evaluate the accuracy of `EmpCom` via measuring the percentages of the correct decisions on Edge-efficient operations (Section 8.2). We further evaluate the speed of `EmpCom` via measuring the time needed for learning growth functions of empirical complexity (Section 8.2). We use the abbreviations of EE and EI to refer to the decisions on Edge-efficient operations and Edge-inefficient operations, respectively. Prior to presenting the results of the experiment, we set up our experiment (Section 8.1).

Algorithm 1 Learning & Selecting Growth Function

Input: $W[N, M]$, $C_{op,p,m}[N]$
Output: f_d, $\widehat{C}_{op,p,m}[f_d]$

▷ **Step 1: Learning growth functions.**

for all $1 \leq f \leq M$ **do**
  $a \leftarrow MIN(\sum_{i=1}^{N}(C_{op,p,m}[i](W[i,f]) - a)^2)$;
  $\widehat{C}^1_{op,p,m}[f] \leftarrow a$;
  $a \leftarrow MIN(\sum_{i=1}^{N}(\log(C_{op,p,m}[i](W[i,f])) - a * \log(\log(W[i,f])))^2)$;
  $\widehat{C}^2_{op,p,m}[f] \leftarrow \log(W[i,f])^a$;
  $(a_1, a_2) \leftarrow MIN(\sum_{i=1}^{N}(\log(C_{op,p,m}[i](W[i,f])) - (\log a_1 + a_2 * \log(W[i,f])))^2)$;
  $\widehat{C}^3_{op,p,m}[f] \leftarrow a_1 * (W[i,f])^{a_2}$;
  $a \leftarrow MIN(\sum_{i=1}^{N}(\log(\widehat{C}_{op,p,m}[f](W[i,f])) - \log a * W[i,f])^2)$;
  $\widehat{C}^4_{op,p,m}[f] \leftarrow a^{W[i,f]}$;
  $min \leftarrow \infty$;
  for all $1 \leq j \leq 4$ **do**
    $sum \leftarrow \sum_i^N \frac{min(C_{op,p,m}[i], \widehat{C}^j_{op,p,m}[f])}{max(C_{op,p,m}[i], \widehat{C}^j_{op,p,m}[f])}$;
    if $sum < min$ **then**
      $min \leftarrow sum$;
      $min_g \leftarrow j$;
    end if
  end for
  $\widehat{C}_{op,p,m}[f] \leftarrow \widehat{C}^{min_g}_{op,p,m}[f]$;
end for

▷ **Step 2: Selecting dominant feature.**

$min \leftarrow \infty$;
for all $1 \leq f \leq M$ **do**
  $sum \leftarrow \sum_i^N \frac{min(C_{op,p,m}[i], \widehat{C}_{op,p,m}[f])}{max(C_{op,p,m}[i], \widehat{C}_{op,p,m}[f])}$;
  if $sum < min$ **then**
    $min \leftarrow sum$;
    $f_d \leftarrow f$;
  end if
end for

Algorithm 2 Deciding on Edge-Efficient Operation

Input: $\{\widehat{C}_{op,p_i,m}[f_d]\}$, $\widehat{w}[f_d]$, HR
Output: `Boolean`

for all $1 \leq p_i \leq |\text{HR}|$ **do**
  if $\widehat{C}_{op,p_i,m}[f_d](\widehat{w}[f_d]) \geq \text{HR}[p_i]$ **then**
    RETURN(*false*);
  end if
end for
RETURN(*true*);

8.1. Setup of experiments

Datasets. We extend an open-source auction app[14] with a back-end (exposed as a RESTful API). The back-end analyzes data by using a k-means clustering algorithm (Tan et al., 2019). The OpenAPI specification of the service has already been presented in Section 3.2. According to this specification, the service provides only one operation. In other words, we learn in our experiment growth functions for the single operation of the clustering service.

We provide as input to the operation datasets that have been collected from the publicly available UC Irvine machine-learning repository.[15] These datasets are suitable for evaluating data-mining algorithms (Dheeru and Karra Taniskidou, 2017). Each dataset corresponds to a text file that consists of a set of multi-dimension data-points. We select datasets that contain at most 12 000 data-points of at most 14 dimensions. We exclude datasets with higher number and/or dimension of data-points because the execution of the clustering operation for such datasets in machines at the Edge is very slow (i.e. in the order of minutes). Overall, we select 1448 datasets that are available at this web link.[16] The overall statistics of the used datasets are presented in Table 3.

Table 3
The overall statistics of the datasets we used in the experiment.

Data-Points (DPs) in a single file		Number of files
Number of DPs	Dimensions of DPs	
[1, 999]	[3, 14]	1210
[1000, 1999]	[3, 12]	45
[2000, 2999]	[3, 10]	22
[3000, 3999]	[3, 7]	8
[4000, 4999]	[3, 12]	17
[5000, 5999]	[3, 12]	19
[6000, 6999]	[4, 7]	13
[7000, 7999]	[3, 7]	8
[8000, 8999]	[7, 7]	7
[9000, 9999]	[3, 8]	41
[10 000, 10 999]	[7, 8]	46
[11 000, 11 999]	[7, 7]	1
Total number of the files:		1448

Machine at the edge. The front-end of the auction app runs on a mobile phone with Android 8.0, CPU at 1.9 GHz, and RAM at 4 GB. One service instance of the back-end is deployed to a resource-constrained laptop. The laptop and the mobile phone are connected at the same local network. The hardware resources of the laptop are the following: Intel Core i3 with CPU at 2.4 GHz and RAM at 2 GB.

Regarding to the critical values of the laptop, we first used `EmpCom` to learn the growth functions of the empirical complexity of the clustering operation for each hardware resource of the laptop. Then, we provided as input each dataset to the clustering operation and we invoked the operation. We following monitored the consumption of each hardware resource and the values of the growth functions for this hardware resource. We finally determined for each hardware resource the minimum value of the corresponding growth function that leads to a full consumption of the hardware resource. In particular, the critical values for the laptop are the following: 11 618 ms for execution time and 19 J for energy consumption.

Accuracy metrics. We calculate the precision and the recall of each Edge-efficiency decision (Baeza-Yates and Ribeiro-Neto, 2011) by using the confusion matrix of Table 4. We then calculate the precision of the EI (resp. EE) decisions as the percentage of the correct decisions over all the EI (resp. EE) decisions. The recall of the EI (resp. EE) decisions equals to the percentage of the correct decisions over all the correct EI (resp. EE) decisions and the wrong EE (resp. EI) decisions. We finally calculated the F-measure of the

14 https://github.com/jagmohansingh/auction-system.

15 http://archive.ics.uci.edu/ml/index.php.

16 https://drive.google.com/file/d/1C-01edWDHwmWpkOmeQCdKHxSVGFQ4TBo/view?usp=sharing.

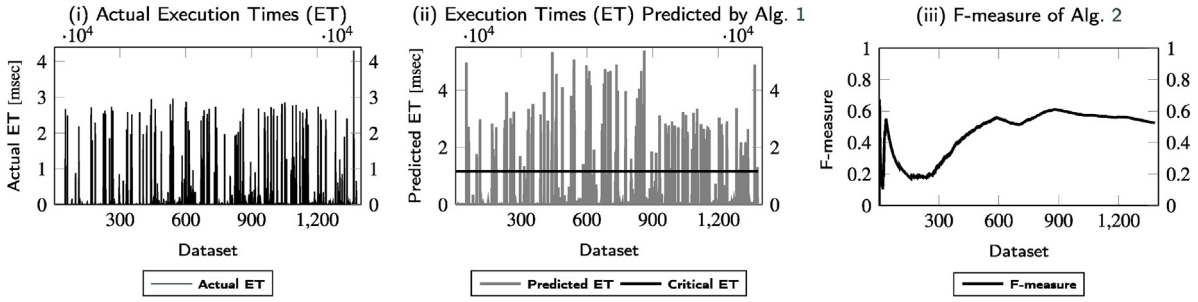

Fig. 8. The actual execution times, the predicted execution times, and the F-measure of the predictions made by A_1.

Table 4
The confusion matrix for calculating precision and recall.

		Prediction	
		EI	*EE*
Correct Decision	*EI*	True EI	False EE
	EE	False EI	True EE

EI (resp. EE) decisions that aggregates the EI (resp. EE) precision and EI (resp. EE) recall, as follows.

$$\text{EI F-measure} = \frac{2 * \text{EI precision} * \text{EI recall}}{\text{EI precision} + \text{EI recall}} \quad (13)$$

$$\text{EE F-measure} = \frac{2 * \text{EE precision} * \text{EE recall}}{\text{EE precision} + \text{EE recall}} \quad (14)$$

To qualitatively characterize the accuracy of an approach, we consider in this work that if the F-measure value belongs to the interval, [0.00, 0.25), then accuracy of an approach is low. We further consider that if the F-measure value belongs to the interval, [0.25, 0.75), then the accuracy of an approach is moderate. We finally consider that if the F-measure value belongs to the interval, [0.75, 1.00], then the accuracy of an approach is high.

Opposing/baseline state-of-the-art approaches. Our opposing approaches construct single-objective predictive models. In particular, our first opposing approach, A_1, uses only the objective of the execution time for making EE/EI decisions (Athanasopoulos et al., 2019). Our second opposing approach, A_2, is an adaptation of A_1 to use only the objective of the energy consumption for making EE/EI decisions. To evaluate a decision as a *true EE* in A_1 (resp. A_2), we consider whether the decision made based on predicted execution-time (resp. energy-consumption) is a correct EE decision. A decision is a *false EE* in A_1 (resp. A_2) if decision made based on predicted execution-time (resp. energy-consumption) is not a correct EE decision. The *true EI* and *false EI* decisions are evaluated in an analogous way. A_1 and A_2 correspond to approaches that learn predictive models by using the technique of the regression analysis.

We use the abbreviation, A_3, to refer to our approach, `EmpCom`. Our third opposing approach, A_4, corresponds to the single-objective state-of-the-art approach that learns predictive models by using a machine-learning technique (Athanasopoulos and Liu, 2022). In particular, A_4 builds four machine-learning models (neural network, k-nearest neighbors, support-vector machines, and decision trees). According to the authors (Athanasopoulos and Liu, 2022), the neural-network models achieve higher accuracy than the other three machine-learning models. This is the reason we compare the neural-network models built by A_4 against the regression models built by our approach, A_3.

Methodology of collecting the results. Given we do not know beforehand how many datasets `EmpCom` needs to use for learning growth functions, we iteratively repeat the experiment as many times as the number of all our datasets is (1448 times). In the first iteration, we execute the clustering operation for only one dataset and measure the execution time and the energy consumption. We then provide to the four approaches (A_1, A_2, A_3, and A_4) the measurements of the execution time and the energy consumption and we calculate the values of the EE/EI precision, the EE/EI recall, and the EE/EI F-measure of the approaches. In the second iteration, we use two datasets and we repeat the steps of measuring execution time/energy consumption and calculating EE/EI precision, EE/EI recall, and EE/EI F-measure. In the third iteration, we repeat the same steps for three datasets. At each iteration, we add a new dataset. The datasets are added in a random order in each iteration.

8.2. Results of the experiment

As mentioned in our methodology, we iteratively provide measurements of execution time and energy consumption the four approaches (A_1, A_2, A_3, and A_4). In particular, we provide these measurements to Alg. 1 of the four approaches for learning predictive models. These predictive models are then used by Alg. 2 to make decisions on whether the clustering operation is Edge-efficient.

Results of the experiment for A_1. We present the actual execution times of the clustering operation for all the datasets in Fig. 8(i). We also present the predicted execution times for the operation in all the datasets in Fig. 8 (ii). Fig. 8 (ii) further presents the critical execution time of the growth functions for the datasets. We observe that the predicted execution times are at most 2% higher than the actual execution times. This difference between the predicted execution times and the actual execution times exists because the prediction error of the growth functions of the execution time is lower than 2%, as shown in Fig. 11(i).

We further observe in Fig. 8 (ii) that there are predicted execution times that are higher than the critical execution time. This observation makes us to think that the clustering operation is not Edge-efficient for all the datasets. To check whether the decisions of Alg. 2 on whether the operation is Edge-efficient is correct, we present the F-measure of the decisions made for A_1 in Fig. 8 (iii). We observe that the F-measure of the decisions is increasing with the increase of the number of the used datasets and is finally stabilized at the interval, [0.5, 0.6]. In other words, the accuracy of the decisions made by A_1 using execution time is moderate (it is moderate because it belongs to the interval, [0.25, 0.75), that is specified in Section 8.1). There are two reasons for the moderate accuracy of A_1. One reason is because the predicted execution times are not always lower than the critical execution time. The other reason is because the energy consumption for the same datasets is not always lower than the critical energy consumption. In other words, the clustering operation is not Edge-efficient for some datasets with respect to both the execution time and the energy consumption. A_1 incorrectly decides that the operation is Edge-efficient for these datasets.

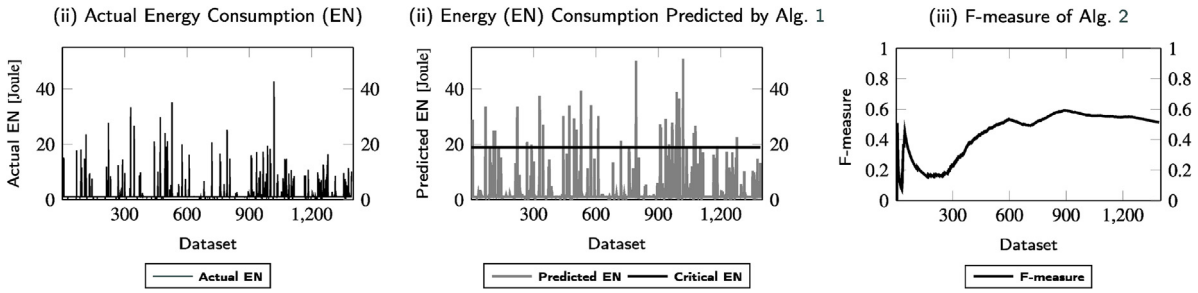

Fig. 9. The actual and predicted energy consumption, along with the F-measure of the predictions made by A_2.

Results of the experiment for A_2. We present the actual energy consumed by the clustering operation in Fig. 9(i). We also present the predicted energy consumption for the clustering operation in Fig. 9 (ii). Fig. 9 (ii) further presents the critical energy consumption of the growth function for all datasets. We observe that the predicted energy consumption is at most 3% higher than the actual energy consumption. This difference between the predicted and actual energy-consumption exists because the prediction error of the growth function of the energy consumption is lower than 3%, as shown in Fig. 11 (ii).

We further observe the predicted energy consumption in Fig. 9 (ii) is higher than the critical energy consumption for some datasets. This observation means that the clustering operation is not Edge-efficient for all the datasets. To check whether the decisions of Alg. 2 on whether the operation is Edge-efficient are correct, we present the F-measure of the decisions made for A_2 in Fig. 9 (iii). We observe that the F-measure of the decisions is increasing with the increase of the number of the used datasets and is finally stabilized at the interval, [0.5, 0.6]. In other words, the accuracy of the decisions made by A_2 using energy consumption is moderate (it is moderate because it belongs to the interval, [0.25, 0.75), that is specified in Section 8.1). There are two reasons for the moderate accuracy of A_2. One reason is because the predicted energy consumption is not always lower than the critical energy consumption. The other reason is because the execution time for the same datasets is not always lower than the critical execution time. In other words, the clustering operation is not Edge-efficient for some datasets with respect to both the execution time and the energy consumption. A_2 incorrectly decides that the operation is Edge-efficient for these datasets.

Results of the experiment for A_3. We present the actual execution times and energy consumption in Fig. 10(i) and (iii), respectively. We also present the predicted execution times and energy consumption in Fig. 10 (ii) and (iv), respectively. Fig. 10 (ii) and (iv) further present the critical execution time and energy consumption of the growth functions for the clustering operation. We observe that the predicted execution times and energy consumption are at most 2% and 3%, respectively, higher than the actual execution times and energy consumption. This difference between the predicted and the actual execution times (resp. energy consumption) exists because the prediction error of the growth function of the execution time (resp. energy consumption) is lower than 2% (resp. 3%), as shown in Fig. 11 (iv) (resp. (v)).

Fig. 10 (ii) and (iv) present the predicted execution times and energy consumption. We observe these predictions individually to each other are quite similar to the predictions made by A_1 and A_2 (Figs. 8 (ii) and 9 (ii)), respectively, because A_3 first learns growth functions in each one of the objectives of the execution time and the energy consumption, independently of the other objectives.

We further observe the predicted energy consumption is not always higher than the critical energy consumption for all the datasets. This observation makes us to think that the clustering operation is not Edge-efficient for all the datasets. To check whether the decisions of Alg. 2 on whether the operation is Edge-efficient are correct, we present the F-measure of the decisions made for A_3 in Fig. 9(v). We observe that the F-measure of the decisions is increasing with the increase of the number of the used datasets and is finally stabilized at the interval, [0.8, 0.85]. In other words, the accuracy of the decisions made by A_3 using execution time and energy consumption is high (it is high because it belongs to the interval, [0.75, 1.00], that is specified in Section 8.1). In contrast to A_1 and A_2, the reason for the high accuracy of A_3 is because A_3 considers the operation is Edge-efficient if both the predicted execution time and energy consumption are lower than the critical execution time and energy consumption, respectively.

Results of the experiment for A_4. We present the actual execution times of the clustering operation for all the datasets in Fig. 12(i). We also present the predicted execution times for the operation in all the datasets in Fig. 12 (ii). Fig. 12 (ii) further presents the critical execution time of the growth functions for the datasets. We observe in Fig. 12 (ii) that there are many predicted execution times that are lower than the critical execution time for the datasets with IDs [1, 900). This observation makes us to think that A_4 decides that the clustering operation is Edge-efficient on these datasets while the clustering operation is not Edge-efficient for these datasets (false Edge-efficient decisions).

One of the reasons that A_4 makes these incorrect decisions is because the machine-learning models have not been properly trained for these datasets. Especially, the prediction errors for these datasets is on average 50%, as we observe in Fig. 11 (iii). As the prediction errors are reduced with the increase of the number of datasets, the F-measure of the decisions is constantly increasing because the machine-learning models are trained better and better. The average F-measure of the decisions made by A_4 over all the datasets is moderate (equals to 32%). The other reason that A_4 makes these incorrect decisions is because the energy consumption for some of the datasets is not always lower than the critical energy consumption. In other words, the clustering operation is not Edge-efficient for some datasets with respect to both the execution time and the energy consumption.

Interestingly, the average F-measure of the decisions made by A_4 over the datasets with IDs [900, 1437] equals 0.85. Thus, the accuracy of the decisions by using execution time is high for the datasets that the machine-learning models have been properly trained. In other words, the polynomial regression gives us the flexibility of making predictions over all the datasets. On the other hand, machine-learning models need a bigger amount of datasets to get first properly trained before using them for making predictions.

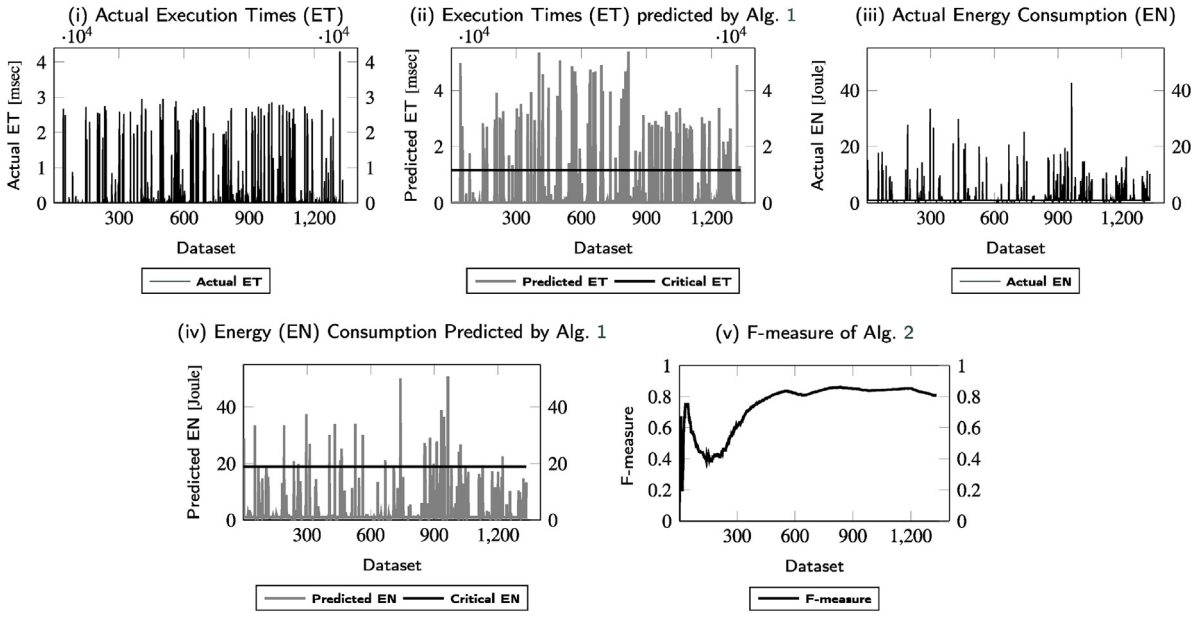

Fig. 10. The actual/predicted execution time/energy consumption and the F-measure of the predictions made by A_3.

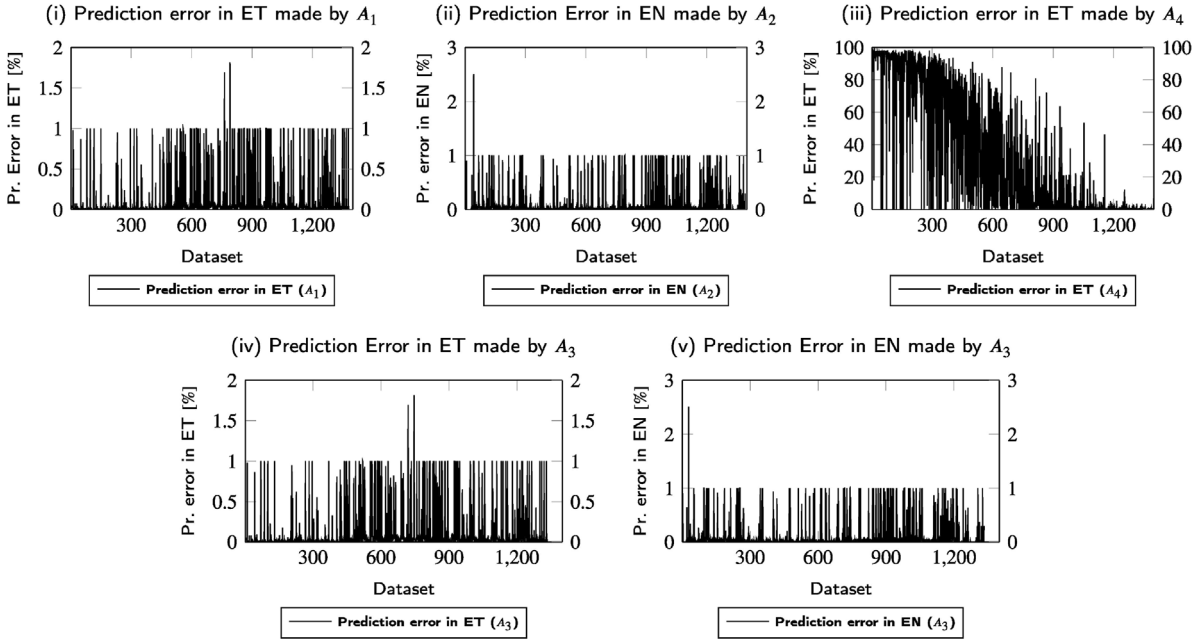

Fig. 11. The prediction errors in execution time and energy consumption made by A_1, A_2, A_3, and A_4.

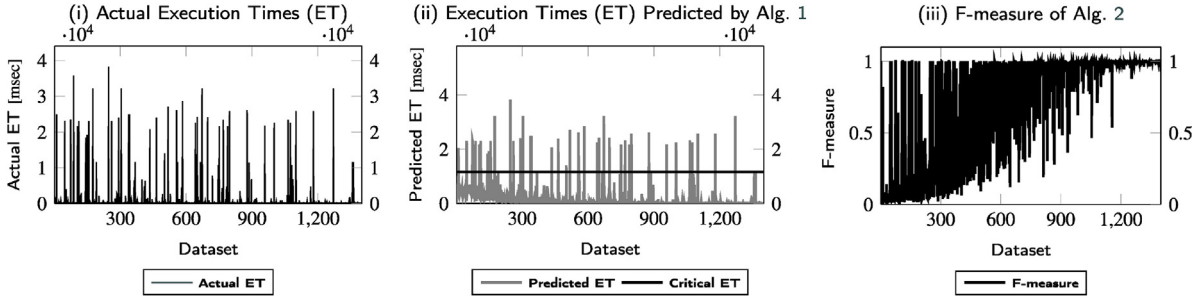

Fig. 12. The actual execution times, the predicted execution times, and the F-measure of the predictions made by A_4.

9. Conclusion and future work

We contributed with the formal specification of the multi-objective empirical complexity of service instances. We further contributed with an algorithm for learning predictive models of multi-objective empirical complexity and an algorithm for deciding on whether a service instance is Edge-efficient. We evaluated the accuracy of our approach via measuring the percentages of the correct decisions on Edge-efficient service instances. We compared the accuracy of our approach against two baseline/opposing state-of-the-art approaches. The results show that the accuracy of our approach is higher than that of the baseline approaches because our approach uses multi-objective predictive models of empirical complexity.

We are currently working on extending the current work by constructing predictive models that consider relations among input features (instead of selecting a dominant feature). Another future direction is the consideration of the interference of management activities at the Edge to the execution time and energy consumption of services. The consideration of multi-tenant models is one of our future research priorities. Another possible future work is the customization of predictive models based on the end-users of services. Finally, our approach could be extended for composite services.

CRediT authorship contribution statement

Dionysis Athanasopoulos: Conceptualization, Methodology, Design of the approach, Development of a part of the research prototype, Evaluation of a part of the approach, Writing of all the versions of the manuscript. **Mitchell McEwen:** Development of the research prototype, Evaluation of a part of our approach.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

AbdelBaky, M., Parashar, M., 2022. A general performance and QoS model for distributed software-defined environments. IEEE Trans. Serv. Comput. 15 (1), 228–240. http://dx.doi.org/10.1109/TSC.2019.2928300.

Amannejad, Y., Krishnamurthy, D., Far, B.H., 2015a. Detecting performance interference in cloud-based web services. In: International Symposium on Integrated Network Management. pp. 423–431. http://dx.doi.org/10.1109/INM.2015.7140319.

Amannejad, Y., Krishnamurthy, D., Far, B.H., 2015b. Managing performance interference in cloud-based web services. IEEE Trans. Netw. Serv. Manag. 12 (3), 320–333. http://dx.doi.org/10.1109/TNSM.2015.2456172.

Ammons, G., Choi, J., Gupta, M., Swamy, N., 2004. Finding and removing performance bottlenecks in large systems. In: European Conference on Object-Oriented Programming. pp. 170–194. http://dx.doi.org/10.1007/978-3-540-24851-4_8.

Andrikopoulos, V., Benbernou, S., Papazoglou, M.P., 2012. On the evolution of services. IEEE Trans. Softw. Eng. 38 (3), 609–628. http://dx.doi.org/10.1109/TSE.2011.22.

Athanasopoulos, D., Liu, D., 2022. AI back-end as a service for learning switching of mobile apps between the fog and the cloud. IEEE Trans. Serv. Comput. 15 (2), 656–668. http://dx.doi.org/10.1109/TSC.2021.3117927.

Athanasopoulos, D., McEwen, M., Rainer, A., 2019. Mobile apps with dynamic bindings between the fog and the cloud. In: International Conference on Service-Oriented Computing. pp. 539–554. http://dx.doi.org/10.1007/978-3-030-33702-5_41.

Athanasopoulos, D., Pernici, B., 2015. Building models of computation of service-oriented software via monitoring performance indicators (short paper). In: International Conference on Service-Oriented Computing and Applications. pp. 173–179. http://dx.doi.org/10.1109/SOCA.2015.21.

Athanasopoulos, D., Zarras, A.V., 2015. Multi-objective service similarity metrics for more effective service engineering methods (short paper). In: International Conference on Service-Oriented Computing and Applications. IEEE Computer Society, pp. 208–212. http://dx.doi.org/10.1109/SOCA.2015.36.

Baeza-Yates, R., Ribeiro-Neto, B.A., 2011. Modern Information Retrieval - the Concepts and Technology behind Search, second ed. Pearson Education Ltd., Harlow, England.

Barna, C., Khazaei, H., Fokaefs, M., Litoiu, M., 2017. Delivering elastic containerized cloud applications to enable DevOps. In: International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 65–75. http://dx.doi.org/10.1109/SEAMS.2017.12.

Barna, C., Litoiu, M., Fokaefs, M., Shtern, M., Wigglesworth, J., 2018. Runtime performance management for cloud applications with adaptive controllers. In: International Conference on Performance Engineering. pp. 176–183. http://dx.doi.org/10.1145/3184407.3184438.

Bartalos, P., Wei, Y., Blake, M.B., Damgacioglu, H., Saleh, I., Celik, N., 2016. Modeling energy-aware web services and application. J. Netw. Comput. Appl. 67, 86–98. http://dx.doi.org/10.1016/j.jnca.2016.01.017.

Beliakov, G., Pradera, A., Calvo, T., 2007. Aggregation Functions: A Guide for Practitioners. In: Studies in Fuzziness and Soft Computing, vol. 221, Springer, http://dx.doi.org/10.1007/978-3-540-73721-6.

Chen, T., Bahsoon, R., 2017. Self-adaptive and online QoS modeling for cloud-based software services. IEEE Trans. Softw. Eng. 43 (5), 453–475. http://dx.doi.org/10.1109/TSE.2016.2608826.

Dawes, J.H., Han, M., Javed, O., Reger, G., Franzoni, G., Pfeiffer, A., 2020. Analysing the performance of python-based web services with the VyPR framework. In: International Conference on Runtime Verification. In: Lecture Notes in Computer Science, vol. 12399, pp. 67–86. http://dx.doi.org/10.1007/978-3-030-60508-7_4.

Dheeru, D., Karra Taniskidou, E., 2017. UCI machine learning repository. URL: http://archive.ics.uci.edu/ml.

Ding, Z., Sun, Y., Jiang, C., Zhou, M., Liu, J., Song, W., 2016. Performance evaluation of transactional composite web services. IEEE Trans. Syst. Man Cybern.: Syst. 46 (8), 1061–1074. http://dx.doi.org/10.1109/TSMC.2015.2478886.

Erl, T., 2016. Service-Oriented Architecture: Analysis and Design for Services and Microservices, second ed. Prentice Hall.

Fanjiang, Y.-Y., Syu, Y., Huang, W.-L., 2022. Time series QoS forecasting for web services using multi-predictor-based genetic programming. IEEE Trans. Serv. Comput. 15 (3), 1423–1435. http://dx.doi.org/10.1109/TSC.2020.2994136.

Fielding, R.T., Taylor, R.N., 2002. Principled design of the modern Web architecture. ACM Trans. Internet Technol. 2 (2), 115–150. http://dx.doi.org/10.1145/514183.514185.

Goldsmith, S., Aiken, A., Wilkerson, D.S., 2007. Measuring empirical computational complexity. In: Joint Meeting of the European Software Engineering Conference and the International Symposium on Foundations of Software Engineering. pp. 395–404. http://dx.doi.org/10.1145/1287624.1287681.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press, http://www.deeplearningbook.org.

Guerriero, A., Mirandola, R., Pietrantuono, R., Russo, S., 2019. A hybrid framework for web services reliability and performance assessment. In: International Symposium on Software Reliability Engineering Workshops. pp. 185–192. http://dx.doi.org/10.1109/ISSREW.2019.00070.

Hasnain, M., Pasha, M.F., Lim, C.H., Ghani, I., 2019. Recurrent neural network for web services performance forecasting, ranking and regression testing. In: Asia-Pacific Signal and Information Processing Association Annual Summit and Conference. pp. 96–105. http://dx.doi.org/10.1109/APSIPAASC47483.2019.9023052.

Hazewinkel, M., 2001. Regression Analysis. In: Encyclopedia of Mathematics, Springer.

Herlocker, J.L., Konstan, J.A., Borchers, A., Riedl, J., 1999. An algorithmic framework for performing collaborative filtering. In: Annual International Conference on Research and Development in Information Retrieval. pp. 230–237. http://dx.doi.org/10.1145/312624.312682.

Hong, C., Varghese, B., 2019. Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms. ACM Comput. Surv. 52 (5), 97:1–97:37. http://dx.doi.org/10.1145/3326066.

Huang, J., Li, S., Chen, Y., Chen, J., 2018. Performance modelling and analysis for IoT services. Int. J. Web Grid Serv. 14 (2), 146–169. http://dx.doi.org/10.1504/IJWGS.2018.10006681.

Ibrahim, A.A.Z.A., 2018. PRESEnCE: A framework for monitoring, modelling and evaluating the performance of cloud saas web services. In: International Conference on Dependable Systems and Networks Workshops. pp. 83–86. http://dx.doi.org/10.1109/DSN-W.2018.00041.

Ibrahim, A.A.Z.A., Varrette, S., Bouvry, P., 2018a. On verifying and assuring the cloud SLA by evaluating the performance of SaaS web services across multi-cloud providers. In: International Conference on Dependable Systems and Networks Workshops. pp. 69–70. http://dx.doi.org/10.1109/DSN-W.2018.00034.

Ibrahim, A.A.Z.A., Varrette, S., Bouvry, P., 2018b. PRESENCE: toward a novel approach for performance evaluation of mobile cloud SaaS Web Services. In: International Conference on Information Networking. pp. 50–55. http://dx.doi.org/10.1109/ICOIN.2018.8343082.

Ibrahim, A.A.Z.A., Wasim, M.U., Varrette, S., Bouvry, P., 2018c. Monitoring and modelling the performance metrics of mobile cloud SaaS web services. Mob. Inf. Syst. 2018, 1351386:1–1351386:14. http://dx.doi.org/10.1109/DSN-W.2018.00034.

Ibrahim, A.A.Z.A., Wasim, M.U., Varrette, S., Bouvry, P., 2018d. Presence: Performance metrics models for cloud SaaS web services. In: International Conference on Cloud Computing. pp. 936–940. http://dx.doi.org/10.1109/CLOUD.2018.00140.

Kaur, P., Goyal, M.L., Lu, J., 2011. Pricing Analysis in Online Auctions Using Clustering and Regression Tree Approach. In: Lecture Notes in Computer Science, vol. 7103, Springer, pp. 248–257. http://dx.doi.org/10.1007/978-3-642-27609-5_16.

Kritikos, K., Pernici, B., Plebani, P., Cappiello, C., Comuzzi, M., Benbernou, S., Brandic, I., Kertész, A., Parkin, M., Carro, M., 2013. A survey on service quality description. ACM Comput. Surv. 46 (1), 1:1–1:58. http://dx.doi.org/10.1145/2522968.2522969.

Lai, C., 2018. MilliScope: A Fine-Grained Monitoring Framework for Performance Debugging of N-Tier Web Services (Ph.D. thesis). Georgia Institute of Technology, Atlanta, GA, USA, URL: http://hdl.handle.net/1853/59260.

Liu, Z., Sheng, Q.Z., Xu, X., Chu, D., Zhang, W.E., 2022. Context-aware and adaptive QoS prediction for mobile edge computing services. IEEE Trans. Serv. Comput. 15 (1), 400–413. http://dx.doi.org/10.1109/TSC.2019.2944596.

Mukherjee, J., Baluta, A., Litoiu, M., Krishnamurthy, D., 2020. RAD: detecting performance anomalies in cloud-based web services. In: International Conference on Cloud Computing. pp. 493–501. http://dx.doi.org/10.1109/CLOUD49709.2020.00073.

Noureddine, A., Rouvoy, R., Seinturier, L., 2015. Monitoring energy hotspots in software - Energy profiling of software code. Autom. Softw. Eng. 22 (3), 291–332. http://dx.doi.org/10.1007/s10515-014-0171-1.

Oriol, M., Franch, X., Marco, J., 2015. Monitoring the service-based system lifecycle with SALMon. Expert Syst. Appl. 42 (19), 6507–6521. http://dx.doi.org/10.1016/j.eswa.2015.03.027.

Papadimitriou, C.H., 1994. Computational Complexity. Addison-Wesley.

Plebani, P., García-Pérez, D., Anderson, M., Bermbach, D., Cappiello, C., Kat, R.I., Pallas, F., Pernici, B., Tai, S., Vitali, M., 2017. Information logistics and fog computing: The DITAS approach. In: International Conference on Advanced Information Systems Engineering. pp. 129–136, URL: http://ceur-ws.org/Vol-1848/CAiSE2017_Forum_Paper17.pdf.

Richardson, L., Ruby, S., 2007. Restful Web Services, first ed. O'Reilly.

Ru, J., Grundy, J.C., Keung, J.W., 2014. Software engineering for multi-tenancy computing challenges and implications. In: International Workshop on Innovative Software Development Methodologies and Practices. ACM, pp. 1–10. http://dx.doi.org/10.1145/2666581.2666585.

Silic, M., Delac, G., Srbljic, S., 2015. Prediction of atomic web services reliability for QoS-aware recommendation. IEEE Trans. Serv. Comput. 8 (3), 425–438. http://dx.doi.org/10.1109/TSC.2014.2346492.

Singh, D., Mukherjee, J., Saikrishna, P.S., Pasumarthy, R., Krishnamurthy, D., 2018. Performance management via MPC for web services in cloud. In: American Control Conference. pp. 5665–5670. http://dx.doi.org/10.23919/ACC.2018.8430989.

Tan, P.-N., Steinbach, M.S., Karpatne, A., Kumar, V., 2019. Introduction to Data Mining, second ed. Pearson, URL: https://www-users.cse.umn.edu/%7Ekumar001/dmbook/index.php.

Wang, H., Yang, Z., Yu, Q., Hong, T., Lin, X., 2018. Online reliability time series prediction via convolutional neural network and long short term memory for service-oriented systems. Knowl. Based Syst. 159, 132–147. http://dx.doi.org/10.1016/j.knosys.2018.07.006.

White, G., Clarke, S., 2022. Short-term QoS forecasting at the edge for reliable service applications. IEEE Trans. Serv. Comput. 15 (2), 1089–1102. http://dx.doi.org/10.1109/TSC.2020.2975799.

Yu, Q., Zheng, Z., Wang, H., 2013. Trace norm regularized matrix factorization for service recommendation. In: International Conference on Web Services. pp. 34–41. http://dx.doi.org/10.1109/ICWS.2013.15.

Zhang, P., Jin, H., Dong, H., Song, W., Bouguettaya, A., 2022. Privacy-preserving QoS forecasting in mobile edge environments. IEEE Trans. Serv. Comput. 15 (2), 1103–1117. http://dx.doi.org/10.1109/TSC.2020.2977018.

Zhang, J., Zhang, L., 2005. Criteria analysis and validation of the reliability of web services-oriented systems. In: International Conference on Web Services. pp. 621–628. http://dx.doi.org/10.1109/ICWS.2005.44.

Zhao, S., Lu, X., Zhou, X., Zhang, T., Xue, J., 2011. A reliability model for web services from the consumers' perspective. In: International Conference on Computer Science and Service System. pp. 91–94.

Zheng, Z., Lyu, M.R., 2010. Collaborative reliability prediction of service-oriented systems. In: International Conference on Software Engineering. pp. 35–44. http://dx.doi.org/10.1145/1806799.1806809.

Zheng, Z., Ma, H., Lyu, M.R., King, I., 2011. Qos-aware web service recommendation by collaborative filtering. IEEE Trans. Serv. Comput. 4 (2), 140–152. http://dx.doi.org/10.1109/TSC.2010.52.

Zheng, Z., Trivedi, K.S., Qiu, K., Xia, R., 2017. Semi-Markov models of composite web services for their performance, reliability and bottlenecks. IEEE Trans. Serv. Comput. 10 (3), 448–460. http://dx.doi.org/10.1109/TSC.2015.2475957.

Zhu, J., He, P., Xie, Q., Zheng, Z., Lyu, M.R., 2017. CARP: context-aware reliability prediction of black-box web services. In: International Conference on Web Services. pp. 17–24. http://dx.doi.org/10.1109/ICWS.2017.10.