Mobile Apps with Dynamic Bindings between the Fog and the Cloud

Mobile Apps with Dynamic Bindings between the Fog and the Cloud

Dionysis Athanasopoulos[1], Mitchell McEwen[2], and Austen Rainer[1]

[1] EEECS School, Queen's University Belfast, United Kingdom
{d.athanasopoulos, a.rainer}@qub.ac.uk
[2] DXC Technology, Wellington, New Zealand
mmcewen3@dxc.com

Abstract. The back-ends of mobile apps usually use services executed on remote (e.g., cloud) machines. The transmission latency may though make the usage of remote machines a less efficient solution for data that need short analysis time. Thus, apps should further use machines located near the network edge, i.e., on the Fog. However, the combination of the Fog and the Cloud introduces the research question of when and how the right binding of the front-end to an edge instance or a remote instance of the back-end can be decided. Such a decision should not be made at the development or the deployment time of apps, because the response time of the instances may not be known ahead of time or cannot be guaranteed. To make such decisions at run-time, we contribute the conceptual model and the algorithmic mechanisms of an autonomic wrapper of edge/remote instances. The wrapper predicts the response time of instances and dynamically decides the binding of the front-end to an instance. The evaluation results of our approach on a real-world app for a large number of datasets show that the wrapper makes efficient binding-decisions in the majority of the datasets, decreasing significantly the response time of the app.

Keywords: Fog · mobile back-end · autonomic control-loop · predictive model.

1 Introduction

Amelia is an avid eBay user, always ready to snap up a bargain. And generally enjoys the thrill of a bidding fight right up to the final moments. She wants to buy a nearly new Xbox but she finds it difficult to decide on the best bidding price. Thus, she downloaded on her phone an auction app that predicts bidding prices [1]. However, Amelia complains she lost some final-moment bidding fights due to delays in the app response.

What Amelia does not know is that data collected on her phone are moved to the Cloud and the output of the analysis is sent back to her. While the Cloud offers powerful machines for efficient data-analytics, the latency of the transmission may make the usage of the Cloud a less efficient solution for data that need short analysis time [2]. To make apps more efficient, service instances (i.e., replicas) of a back-end should be further deployed on the Fog. The Fog constitutes machines located near the network edge (e.g., laptops, small-scale data-centers) [3]. The combination of the Fog and the

Cloud though introduces the research question of when and how the right binding of the front-end of an app to an edge or a remote instance can be decided[3].

Concerning the first part of the question, bindings should not be decided at the development or the deployment time of apps (as the state-of-the-art does), because the response time of instances may not be known ahead of time or cannot be guaranteed. Thus, we face the challenge of deciding the binding of the front-end at run-time. Regarding the second part of the question, we consider that the response time of an instance depends on the execution time of the instance on a machine and on the network latency to reach out the machine[4]. Thus, we face the challenge to predict the response time of instances based on the input datasets and the machines used for deploying the instances. On top of that, the efficiency challenge of predicting response times from a large number of datasets is raised, since the number of the datasets increases over time.

To address the above challenges, we contribute the conceptual model and the algorithmic mechanisms of an *autonomic* wrapper of the edge and the remote service-instances of a back-end (one wrapper for each back-end). The wrapper is offered as a service, is deployed on the Fog, and acts as a proxy between the front-end and the back-end instances[5]. Each time the front-end interacts with the wrapper, the latter dynamically predicts the response time of the instances and decides the binding of the front-end to an instance. To do it in an autonomic manner, the wrapper follows the *autonomic control-loop* of self-adaptive software [4]. Specifically, the wrapper monitors the past invocations to the instances, analyses a few *representative input datasets* (for addressing the efficiency challenge) to (re-)build *predictive models* of the response time of the instances, and dynamically decides the binding of the front-end to an instance.

To evaluate our approach, we implement a research prototype of the autonomic wrapper of the auction app (Fig. 1). A large number of datasets, collected from the UC Irvine machine-learning repository [5], is given as input to the app. The experimental results show that the wrapper makes efficient binding-decisions in the majority of the datasets, decreasing significantly the response time of the app.

The rest of the paper is structured as follows. Section 2 describes the related approaches and compares them against ours. Sections 3 and 4 specify the conceptual model and the algorithmic mechanisms of autonomic wrapper. Section 5 presents the evaluation of our approach. Section 6 discusses the threats to the validity of our work. Section 7 summarizes our contribution and discusses future directions of our research.

2 Related Work

The approaches that use both the Fog and the Cloud for the execution of mobile apps have typically focused on the development or the deployment time of apps. These approaches aim at meeting the non-functional requirements of apps and/or machines.

[3] The front-end of an app includes the graphical user-interface of the app and the programming client that interacts with (feeding with data) the service back-end.

[4] The datasets that are locally stored by the instances are synchronized to the storage of the cloud instance. We do not consider the synchronization time in the current work.

[5] At least an edge and a remote instance have been pre-deployed. Their endpoints are registered to the wrapper during its deployment. The decision of the number of instances is future work.

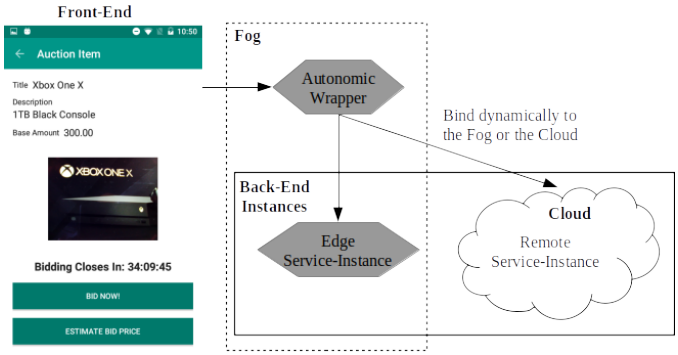

Fig. 1. Auction app extended with the autonomic wrapper and edge/remote back-end instances.

Concerning the non-automated approaches, [6] provides suggestions for deploying the back-ends of an app to machines. The suggestions mainly aim at reducing the network latency, the energy consumed by apps and the financial cost of renting machines. [7] proposes a methodology for assessing the security level of deployment plans.

Regarding the automated approaches, [8], [9] and [10] generate deployment plans that minimize the network latency, the delay of machines to serve apps and the renting cost, respectively. [11, 12] generate deployment plans that minimize the network usage. [13, 14] generate deployment plans for a set of apps to reduce the power consumption. [15] produces deployment plans based on the renting cost and the end-users' budgets. [16] selects machines at deployment time based on their ranking with respect to the delay of machines and the power consumption. [17] regenerates deployment plans via modeling the delay of machines as a function of the elapsed discrete-time. Finally, [18] regenerates deployment plans when the latency of back-ends exceeds a time threshold.

Overall, two approaches monitor the app execution to use the current values of non-functional properties for regenerating deployment plans [17, 18]. The two approaches are though reactive because they suspend the app execution to redeploy the apps. The approaches also lay between apps and operating systems (e.g., deployment engines). Our approach defines autonomic wrappers that run at the application layer and proactively (without suspending the app execution) decide the binding of the front-end.

3 Conceptual Model of Autonomic Wrapper

The conceptual model of an autonomic wrapper is depicted in Fig. 2. Based on the model, the autonomic wrapper offers an API (Section 3.1), implements a dynamic binding-mechanism (Section 3.2) and follows an autonomic control-loop (Section 4).

3.1 API of Autonomic Wrapper

The API offers all of the operations of the back-end of a mobile app. From the Web-service technology perspective, an API is exposed by using the REST [19] or the SOAP [20] protocol. We define the notion of the API in a generic manner as follows.

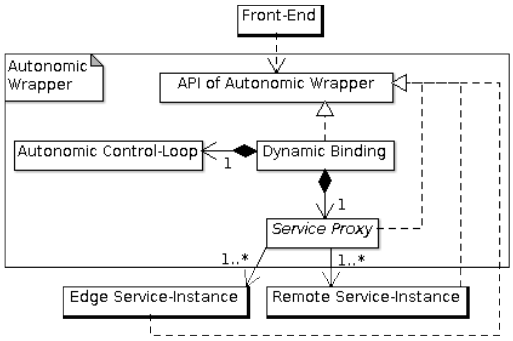

Fig. 2. The UML diagram of the conceptual model of an autonomic wrapper.

Definition 1. *Each operation of the API of an autonomic wrapper corresponds to a programming method*[6] *of the API of the back-end. Each operation accepts/returns a (possibly empty) set of input/output parameters.*

The schemas of input/output parameters are technically specified in the XML or the JSON format. Given that XML/JSON schemas can be transformed to programming objects (e.g., JAXB[7]), we define the notion of the parameter in a generic manner based on the object-oriented paradigm as follows.

Definition 2. *A parameter p is characterized by a name and a built-in or complex data-type. A complex data-type is a group of or a reference to another parameter.*

Taking the example of the auction app, we assume the back-end uses the k-means clustering algorithm [22]. The operation of the API accepts the following parameters: the number of clusters, the number of the algorithm iterations and a dataset. The first and the second parameters have a built-in data-type (e.g., `int`), while the third parameter has the complex data-type of a list of multi-dimensional data-points (i.e., `List<Double[] dataPoint> dataset`). In turn, a data-point is a parameter that has a complex data-type too. The latter is defined by a vector of coordinates, where each coordinate has a built-in (`Double`) data-type.

Based on Def. 2, a parameter has a hierarchical structure. However, only the leaves of the above structure carry actual data values. Returning to our example, while a two-layer structure is formed, only the second/lowest layer (`Double[] dataPoint`) contains actual data (a.k.a., the coordinates of data-points). Based on this observation, our approach pre-processes multi-layer parameters and converts them to a set of unstructured ones. We define the notion of the unstructured parameter as follows.

Definition 3. *The unstructured parameters of a multi-layer parameter p is the set of the leaf data-types $\{x\}$ of p. Each unstructured parameter x is defined by a tuple (n, c)*

[6] While the programming methods are explicitly defined in a SOAP-based API, the methods can be determined in a RESTful API by parsing the suffix of the URI of the API [21].

[7] https://docs.oracle.com/javase/tutorial/jaxb/intro/index.html

that consists of the name n and the cardinality c of the corresponding leaf data-type. The cardinality c equals to product of the cardinalities of the grouping structures that are met in the path followed to reach n from the root node of p.

The multi-layer parameter, $p = List < Double[]\ dataPoint > dataset$, in our example is converted to a singleton set, $\{x\} = \{(dataPoint, c)\}$, where c is the cardinality of the list structure (the list is the only node that precedes x in the path from the root)[8]. Hereafter, we use the term parameter to refer to an unstructured parameter.

3.2 Dynamic Binding-Mechanism of Autonomic Wrapper

The dynamic binding-mechanism comprises the components of the `Dynamic Binding`, the `Autonomic Control-Loop` and the `Service Proxy`, as depicted in Fig. 2.

The `Dynamic Binding` implements all of the operations of the API of the back-end. In particular, the implementation of the operations of the `Dynamic Binding` handles calls from the front-end to the operations of the back-end. To handle such calls, the `Dynamic Binding` first pre-processes the input data (to form the unstructured parameters), following uses the `Autonomic Control-Loop` to predict the response time of the edge/remote instances and then finds the instance that has the lowest predicted-value. Finally, the `Dynamic Binding` uses the `Service Proxy`[9] to forward the operation call to the selected instance. To forward operation calls, the `Dynamic Binding` instantiates the target parameters of the operation of the selected instance. To instantiate them, the `Dynamic Binding` considers that an one-to-one mapping exists between the parameters of the called operation of the autonomic wrapper and the parameters of the called operation of the selected instance. This mapping exists because the edge instance and the remote instance implement the same API and hence, their operations accept/return the same parameters with those of the autonomic wrapper.

4 Autonomic Control-Loop

The control loop of the autonomic wrapper extends the generic Monitor-Analyze-Plan-Execute and Knowledge loop (MAPE-K) of self-adaptive software [4]. The Monitoring mechanism records the response time of edge/remote service-instances, i.e., the elapsed time between when the invocation is made and when the response is returned back. Thus, the response time is the sum of the execution time of an instance on a machine and of the network latency.

The core tasks of the loop are implemented by the Analysis and the Planning mechanisms (Sections 4.1 and 4.2). The Analysis mechanism (re-)constructs predictive models of response times that are expressed as a function of the values of the input parameters. A different predictive-model is constructed for each edge/remote instance because instances are usually deployed on different machines.

[8] If the cardinality is not declared in parameter schemas, then our approach considers a large pre-defined value as an artificial cardinality.

[9] The relationship between the `Dynamic Binding` and the `Service Proxy` is UML composition (depicted by filled diamond) so as to hide the edge/remote instances from the front-end.

The Planning mechanism dynamically (re-)creates groups of parameter values and stores a representative value for each group, along with the corresponding monitoring response-times[10]. In this way, the mechanism does not store a high number of parameter values. The mechanism uses the constructed models and the stored parameter-values to predict response times and select the instance that has the lowest predicted-value.

4.1 Analysis Mechanism

We firstly define the notions of the predictive model and the prediction error.

Predictive Model and Prediction Error Since the mechanism constructs a different predictive-model for each service instance of an API, we define the above notions for each (edge or remote) service instance and especially, for each operation of an instance.

Definition 4 (Predictive model of an operation). *The predictive model, p_{op}, of an operation, op, of a service instance, si, is defined by the tuple:* $\Big(x[D,N],\ \widehat{y}[D],\ y(x)\Big)$

- $x[D,N]$*: D past values of each one of the N input parameters of op*
- $\widehat{y}[D]$*: D monitoring response-times of op*
- $y(x)$*: a polynomial function of x that predicts the response time of op.*

Definition 5 (Prediction error for an operation). *The prediction error, e, for the response time of an operation op, of a service instance si for the current values, $x[N]$, of the input parameters of op, equals to the relative distance of the predicted (for x) response-time, y, from the monitoring response-time, $\widehat{y}$, of op:* $e = \frac{|y(x)-\widehat{y}|}{\widehat{y}}$

Analysis Algorithm Considering that polynomials describe the performance of programs well [23], the algorithm builds polynomial functions to predict response times. One of the most widely used techniques to build polynomials is the regression technique that applies the Taylor's term-expansion [24]. However, it takes all of the possible variable combinations and consequently, forms long expressions with unneeded terms. To build compact expressions, greedy techniques have been proposed [24, 25]. We extend the sparse-term technique of [25] that examines a small number of terms. Our technique further selects the term that is dominant (i.e., it has the lowest prediction-error) and confident (i.e., its prediction error is higher than a threshold).

The algorithm steps are specified in Alg. 1. Alg. 1 accepts as input the current values of the input parameters, the past and the current monitoring response-times and the polynomial function of the predictive model of an operation of a service instance. The inputs of Alg. 1 further include a threshold ω of the lowest prediction-error. Alg. 1 initially calculates the prediction error (Alg. 1 (1-4)). If the error is higher than ω, Alg. 1 rebuilds the predictive model via fitting all of the possible single-variable terms to the past and the current response-times (Alg. 1 (8)). To fit a term to the response times, Alg. 1 applies the linear least-square regression-technique [24] (Alg. 1 (5-10)). Finally, Alg. 1 selects the term that is dominant and confident (Alg. 1 (11-20)). This term is the polynomial function of the updated predictive-model and is returned back.

[10] The constructed predictive-models, the groups of parameter values, and the monitoring response-times are stored as the Knowledge of the loop.

Algorithm 1 Analysis Mechanism

Input: $x[D, N]$, $\widehat{y}[D]$, $y(x)$, ω
Output: $y(x)$

1: $e \leftarrow \frac{|y(x[D,N]) - \widehat{y}[D]|}{\widehat{y}[D]}$;
2: **if** $e > \omega$ **then**
3: $T \leftarrow$ FIT$(x,\ \widehat{y})$;
4: $y(x) \leftarrow$ SELECT$(T,\ x,\ \widehat{y})$;
5: **function** FIT($x[D, N]$, $\widehat{y}[D]$): T
6: **for all** $1 \leq j \leq N$ **do**
7: $y(x) \leftarrow a * x^b$
8: FIND $a, b : \sum_{i=1}^{D} (y(x[i, j]) - \widehat{y}[i])^2$ is minimized
9: T.ADD$(y(x))$;
10: **end function**
11: **function** SELECT(T, $x[D, N]$, $\widehat{y}[D]$): $y(x)$
12: **for all** $y(x) \in T$ **do**
13: **for all** $1 \leq i \leq D$ **do**
14: e += $\frac{|y(x[i,N]) - \widehat{y}[i]|}{\widehat{y}[i]}$;
15: $\overline{e} \leftarrow \frac{e}{|D|}$;
16: **if** $\overline{e} < min$ and $\overline{e} \geq \omega$ **then**
17: $min \leftarrow \overline{e}$;
18: $min_y \leftarrow y(x)$;
19: $y(x) \leftarrow min_y$;
20: **end function**

4.2 Planning Mechanism

We firstly define the notion of the partition that is used by algorithm.

Partitions of the Domain Values of Parameter The Planning mechanism partitions the domain of the values of each parameter and stores a representative value for each partition. A partition is defined as an one-dimension interval. Overall, the partitions of a parameter are a set of intervals with consecutive integer-endpoints, as defined below.

Definition 6 (Partitions). *Let x_{min} and x_{max} be the min and the max domain values of a parameter, x. The set, r, of the Q partitions of the domain values of x is defined below:*

$$r = \Big\{r_1,\ \ldots,\ r_j,\ \ldots,\ r_Q\Big\},\ \textit{where}$$

$$r_1 = \Big[x_{min},\ x_{min} + len(x)\Big]$$

$$r_j = \Big[x_{min} + len(x) * (j-1) + 1, x_{min} + len(x) * j\Big],\ j \in [2, Q-1]$$

$$r_Q = \Big[x_{min} + len(x) * (Q-1),\ x_{max}\Big]$$

The partition length that is used in Def. 6 is defined as follows.

Algorithm 2 Planning Mechanism

Input: $x[N], r[N], v[N], \{si\}, \{y(x)\}$
Output: si

1: PARTITION(r, v, x);
2: $si \leftarrow$ SELECT($\{si\}, x, \{y(x)\}$);
3: **procedure** PARTITION($r[N], v[N], x[N]$)
4: **for all** $1 \leq i \leq N$ **do**
5: $adjustment \leftarrow$ `false`;
6: **if** $x[i] > x_{max}[i]$ **then**
7: $x_{max}[i] \leftarrow x[i]$;
8: $adjustment \leftarrow$ `true`;
9: **else if** $x[i] < x_{min}[i]$ **then**
10: $x_{min}[i] \leftarrow x[i]$;
11: $adjustment \leftarrow$ `true`;
12: **if** $adjustment$ = `true` **then**
13: $l \leftarrow \frac{x_{max}[i]-x_{min}[i]}{Q}$;
14: $r_1[i] \leftarrow \left[x_{min}[i], x_{min}[i] + l\right]$;
15: $v_1[i] \leftarrow$ UPDATE($v_1[i],\ r_1[i]$);
16: **for all** $2 \leq j \leq Q$ **do**
17: $r_j[i] \leftarrow \left[x_{min}[i] + l * (j-1) + 1, x_{min}[i] + l * j\right]$;
18: $v_j[i] \leftarrow$ UPDATE($v_j[i],\ r_j[i]$);
19: $r_Q[i] \leftarrow \left[\ x_{min}[i] + l * Q, x_{max}[i]\ \right]$;
20: $v_Q[i] \leftarrow$ UPDATE($v_Q[i],\ r_Q[i]$);
21: **for all** $1 \leq j \leq Q$ **do**
22: **if** $x_j[i].c \in r_j[i]$ **then**
23: **if** $|v_j[i]|$ = 0 **then**
24: $v_j[i] \leftarrow$ ADD($v_j[i],\ x_j[i].c$);
25: **end procedure**

Definition 7 (Partition length). *Let x_{min} and x_{max} be the min and the max domain values of a parameter, x, and Q be a number of partitions. The partition length is calculated by dividing the range of values of x in Q equally-sized intervals:* $len(x) = \lceil \frac{x_{max}-x_{min}}{Q} \rceil$

The representative values of a parameter that are stored in the partitions (one value in each partition) are defined as follows.

Definition 8 (Representative values of partitions). *Let r be the set of the partitions of the domain values of a parameter x. The Q representative values of the Q partitions are defined by the set,* $v = \left\{x_1.c,\ \ldots,\ x_j.c,\ \ldots,\ x_Q.c\right\}$, *where* $x_j.c \in r_j$.

However, the parameter values are not available beforehand. Thus, the Planning mechanism dynamically (re-)defines the partitions whenever a new parameter-value arrives and this value is not represented by an existing partition.

Planning Algorithm Alg. 2 accepts as input the current values of the input parameters of an operation of an API, along with the existing partitions of the parameters. The

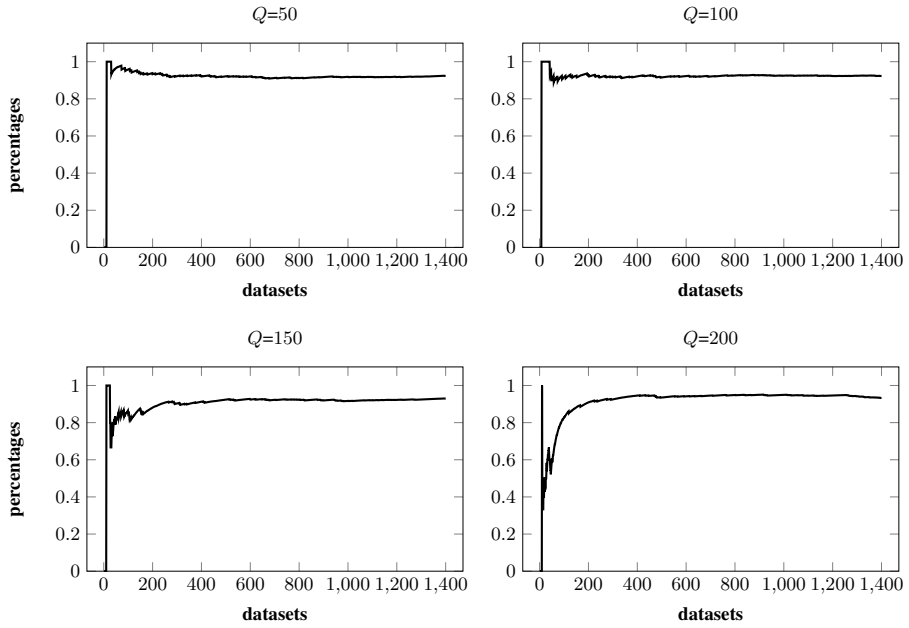

Fig. 3. The percentages of the correct binding-decisions.

inputs of Alg. 2 further include the available service-instances of the API and the set of the polynomial functions of the current predictive-models of the service instances. If the current values of the input parameters do not belong to the existing partitions, Alg. 2 redefines the partitions by adjusting their endpoints (Alg. 2 (5-11)) and accordingly redistributing their representative values[11] (Alg. 2 (13-20)). Moreover, Alg. 2 adds the current values of the input parameters to the proper partitions as their representative values only if these partitions do not currently have such values (Alg. 2 (21-24)). Following, Alg. 2 uses the predictive models to predict the response times of the service instances for the current values of the input parameters and returns back the instance that has the lowest predicted-value[11].

5 Experimental Evaluation

We evaluate our approach on five benchmarks, B1 – B5 (Section 5.2). Prior to presenting the results, we set up our experiments (Section 5.1).

5.1 Experimental Setup

We extended an existing auction app[12] with a data-analytics back-end that uses the k-means clustering algorithm. We implemented a research-prototype of the autonomic

[11] We do not specify the algorithmic details of the functions `UPDATE`, `ADD` and `SELECT`, since they are straightforward.

[12] https://github.com/jagmohansingh/auction-system

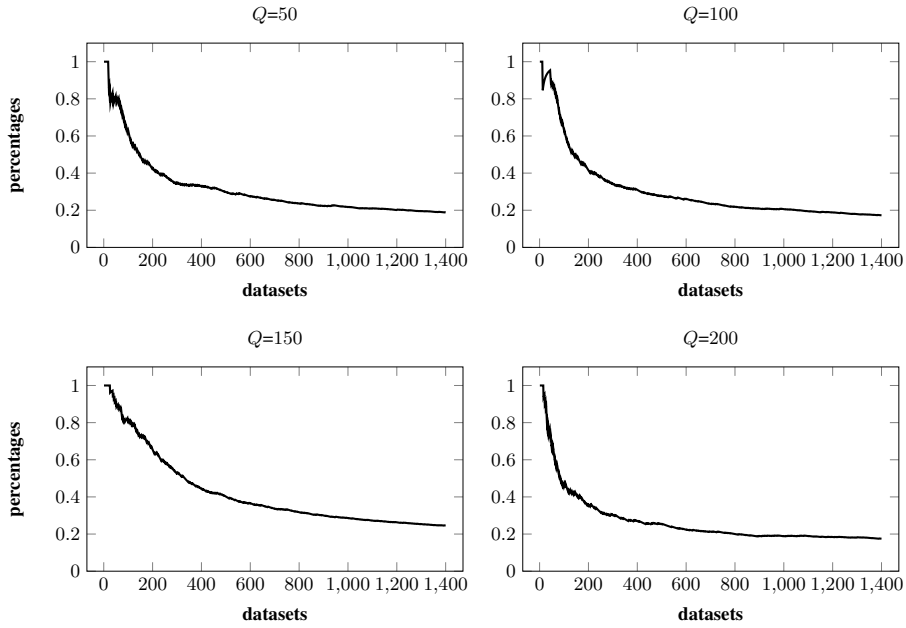

Fig. 4. The percentages of the datasets used for rebuilding predictive models.

wrapper of the back-end in Java. Datasets collected from UC Irvine machine-learning public repository[13] were given as input to the app. We used all of the datasets where the number (resp., dimensions) of the data points in the respective dataset was less than 12000 (resp., 12). We did it due to the computational constraints of the used machines. As also the evaluation results show in Section 5.2, the usage of extra datasets from the repository would have been redundant. In that way, we concluded to use 1456 datasets. We run the experiments 1456 times, each time adding an extra dataset, starting with one dataset and progressing with the remaining datasets in a random order.

The front-end of the auction app runs on a mobile phone[14], the edge instance and the wrapper on a laptop[15] (that was connected to the same LAN with the mobile phone), and the remote instance on a virtual machine deployed to the Google cloud[16].

5.2 Evaluation Results

B1. *How many binding decisions are correct*?
During this experiment, each dataset is given as input to the wrapper. For each dataset, the wrapper makes a binding decision between the edge instance and the

[13] http://archive.ics.uci.edu/ml/index.php
[14] 1.9GHz CPU, 4GB RAM, Android 8.0.
[15] 2.70GHz CPU, Intel Core i5-5257U, 64-bit Windows 10 Home, 8GB RAM.
[16] 2.2GHz 2 vCPU, Intel Xeon E5 v4 (Broadwell) platform, 7.5GB RAM, Windows server 2016 (the cost of renting a more powerful machine for our experiments was very high).

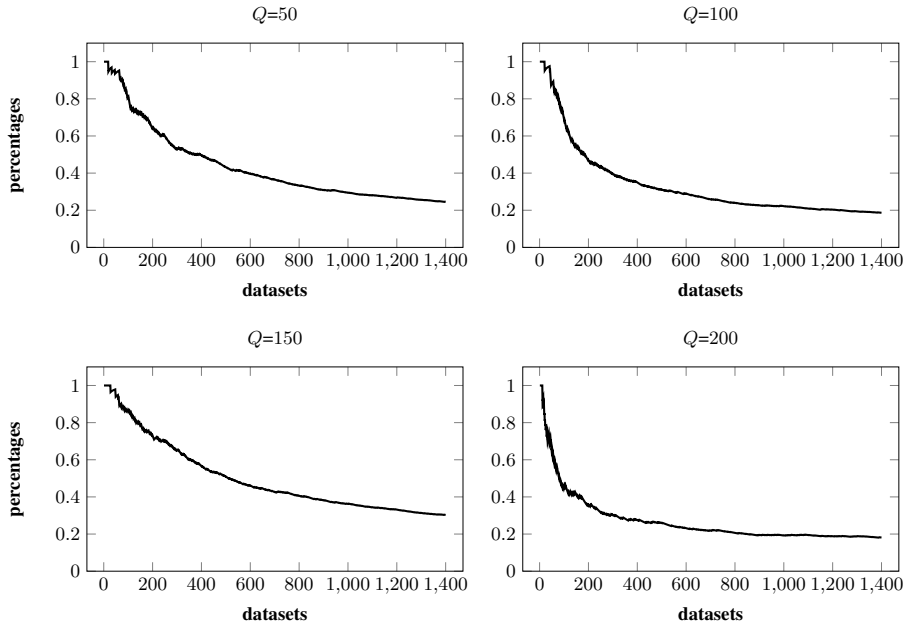

Fig. 5. The percentages of the monitoring response-times used for rebuilding predictive models.

remote instance. We repeated this experiment for different numbers of partitions Q of the domain values of parameters (the value of Q affects the times the predictive models are reconstructed). Fig. 3 depicts the percentages of the correct binding-decisions (i.e., the number of the correct decisions is divided by the number of all the decisions). We observe from the results that the percentages of the correct decisions are increasing with the increase of the number of the provided datasets and are finally stabilized at the 90% of the total number of the decisions.

B2. *How many datasets are used for rebuilding predictive models*?

We present in Fig. 4 the percentages of the datasets that are used for rebuilding the predictive models in the above experiments (i.e., the number of the used datasets is divided by the number of all of the datasets). We observe from the results that the percentages of the used datasets decrease with the increase of the number of the provided datasets. Especially, a small percentage (20%-30%) of the total number of the datasets is finally used. It further shows that the usage of extra datasets in the experiment would have been redundant.

B3. *How many monitoring response-times are used for rebuilding predictive models*?

Fig. 5 depicts the percentages of the monitoring response-times of the edge/remote instances used in the above experiments (i.e., the number of the used response-times is divided by all of the monitoring response-times). We observe from the results that the percentages of the used response-times decrease with the increase of the number of the provided datasets. Especially, a small percentage (20%-30%) of the total number of the monitoring response-times is finally used. Furthermore, as

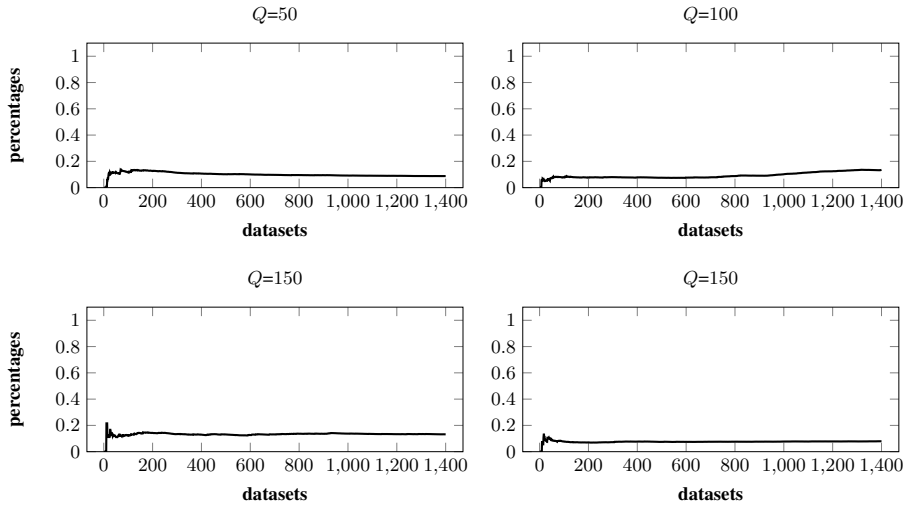

Fig. 6. The overhead added by the autonomic control-loop to the response time of the app.

expected, the curves of the percentages of the used datasets and the used response-times in Fig. 4 and Fig. 5, respectively, are analogous.

B4. *What is the overhead to the response-time of the app*?
We present in Fig. 6 the percentages of the overhead to the response time of the app from the usage of the autonomic mechanisms (i.e., the execution times of the mechanisms is divided by the total response-times of the app). We observe from the results that the overhead comes to the 10%-15% of the response time of the app. The overhead is higher in the firstly provided datasets because the reconstruction of the predictive models and the parameter partitions occur more frequent.

B5. *What is the improvement to the response time of the app*?
To examine the improvement, we divide the datasets into small-, medium-, and large-sized datasets. We consider that the size of a dataset equals to the number of the datapoints of the dataset[17]. The first two charts of Fig. 7 present the response times of the app when it uses for small- and medium-sized datasets the remote instance only or the edge instance only. The next two charts present the response times of the app for large datasets[18]. The last two charts present the response times of the app when it uses the wrapper. Comparing Fig. 7 (vi) against Fig. 7 (iii), we observe that the wrapper selects the remote instance for large-sized datasets. On the contrary, comparing Fig. 7 (v) against Fig. 7 (ii), we observe that the wrapper selects the edge instance for small- and medium-sized datasets.
The above observations verify our intuition that Cloud machines are much more efficient than Fog machines on large-sized datasets. Since the wrapper selects for small- and medium-sized datasets the edge instance (instead of the original op-

[17] We define the equally-sized intervals of the dataset sizes, (1, 4000], (4000, 8000] and (8000, 12000], which correspond to small-, medium-, and large-sized datasets, respectively.

[18] The scale of the y-axis in the first two charts is different from the scale in the next two charts.

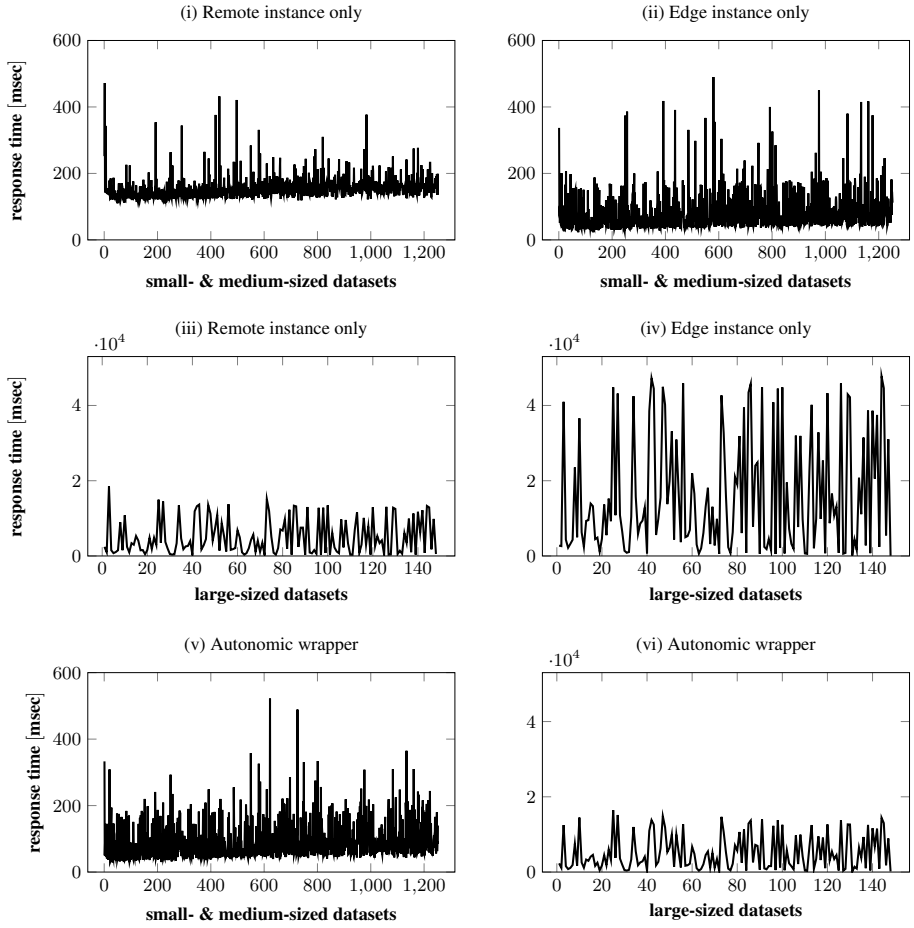

Fig. 7. The response time of the app when it uses the edge/remote instances or the wrapper.

tion of the remote instance), the response time of the app is improved. To quantify that improvement, we calculate for each small- and medium-sized dataset the percentage of the decrease of the response time of the app by using the formula, $\frac{y_{remote}-y_{edge}}{y_{remote}}$. Overall, the average improvement over all of the small- and medium-sized datasets comes to the 50% of the original response-time of the app.

6 Threads to Validity

A possible threat to the internal validity of the study is the exclusion from the experiments of the datasets that have more than 12000 datapoints and 12 dimensions. However, this threat may be mitigated based on the observation from the results that our approach needs only the firstly provided 20%-30% of the datasets to stabilize the number of the correct binding-decisions. Regarding the external validity, our study does not

explicitly associate the built predictive-models to the software/hardware properties of machines or the machine/network load. To reduce this threat, our approach builds separate predictive-models for each service instance. Additionally, each model is built as a function of the response time of an instance that is equal to sum of the execution time of the instance on a machine and of the network latency. In this way, the models are indicative of the network latency and the delay of machines to serve apps at the time periods and the network locations that the monitoring measurements were made.

7 Conclusions & Future Work

We contributed with the specification of the conceptual model and the algorithmic mechanisms of an autonomic wrapper of edge/remote instances of a mobile backend. The evaluation results on five benchmarks showed that the number of the correct binding-decisions is the 90% of the total number of decisions, the number of the monitoring data (datasets and response times) used for reconstructing predictive models is the 20%-30% of the total number of data, the overhead added by the autonomic mechanisms comes to the 10%-15% of the response time of an app, and the response time of an app is decreased to the 50% of its original response-time.

A future research-direction is to consider the synchronization time (spent by the edge/remote instances for storing datasets) in the construction of predictive models. Another direction is to explicitly associate the predictive models to the machine properties/load and the network properties/load. A final direction is to enhance the wrapper with the capability to dynamically (de-)register service instances that are (not) available on the Fog and the Cloud.

Acknowledgments

The work was partially funded from the Victoria University of Wellington in New Zealand. We further express many thanks to Prof. B. Pernici for her valuable reviews.

References

1. P. Kaur, M. Goyal, and J. Lu. Pricing analysis in online auctions using clustering and regression tree approach. In *Agents and Data Mining Interaction*, pages 248–257, 2012.
2. P. Plebani, D. García-Pérez, M. Anderson, D. Bermbach, C. Cappiello, R. I. Kat, F. Pallas, B. Pernici, S. Tai, and M. Vitali. Information logistics and fog computing: The ditas approach. In *International Conference on Advanced Information Systems Engineering*, pages 129–136, 2017.
3. B. Varghese, M. Villari, O. Rana, P. James, T. Shah, M. Fazio, and R. Ranjan. Realizing edge marketplaces: Challenges and opportunities. *IEEE Cloud Computing*, 5(6):9–20, 2018.
4. J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.
5. D. Dheeru and E. Karra Taniskidou. UCI machine learning repository, 2017.
6. M. Ashouri, P. Davidsson, and R. Spalazzese. Cloud, edge, or both? towards decision support for designing iot applications. In *International Conference on Internet of Things: Systems, Management and Security*, pages 155–162, 2018.

7. A. Brogi, G. L. Ferrari, and S. Forti. Secure cloud-edge deployments, with trust. *CoRR*, abs/1901.05347, 2019.
8. A. Brogi and S. Forti. Qos-aware deployment of iot applications through the fog. *IEEE Internet of Things Journal*, 4(5):1185–1192, 2017.
9. R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang. Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. *IEEE Internet of Things Journal*, 3(6):1171–1181, 2016.
10. A. Brogi, S. Forti, and A. Ibrahim. Deploying fog applications: How much does it cost, by the way? In *International Conference on Cloud Computing and Services Science*, pages 68–77, 2018.
11. N. Mohan and J. Kangasharju. Edge-fog cloud: A distributed cloud for internet of things computations. In *Cloudification of the Internet of Things*, pages 1–6, 2016.
12. N. Mohan and J. Kangasharju. Edge-fog cloud: A distributed cloud for internet of things computations. *CoRR*, abs/1702.06335, 2017.
13. A. Brogi, S. Forti, and A. Ibrahim. How to best deploy your fog applications, probably. In *International Conference on Fog and Edge Computing*, pages 105–114, 2017.
14. H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296, 2017.
15. D. H. Tran, N. H. Tran, C. Pham, S. M. A. Kazmi, E.-N. Huh, and C. S. Hong. Oaas: Offload as a service in fog networks. *Computing*, 99(11):1081–1104, 2017.
16. X. Guo, R. Singh, T. Zhao, and Z. Niu. An index based task assignment policy for achieving optimal power-delay tradeoff in edge cloud systems. In *IEEE International Conference on Communications*, pages 1–7, 2016.
17. A. Yousefpour, G. Ishigaki, and J. P. Jue. Fog computing: Towards minimizing delay in the internet of things. In *IEEE International Conference on Edge Computing*, pages 17–24, 2017.
18. E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwälder. Incremental deployment and migration of geo-distributed situation awareness applications in the fog. In *ACM International Conference on Distributed and Event-based Systems*, pages 258–269, 2016.
19. L. Richardson and S. Ruby. *Restful Web Services*. O'Reilly, first edition, 2007.
20. T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, 2005.
21. M. Fokaefs and E. Stroulia. Using WADL specifications to develop and maintain REST client applications. In *2015 IEEE International Conference on Web Services*, pages 81–88, 2015.
22. A. J. Smola and S.V.N. Vishwanathan. *Introduction to Machine Learning*. Cambridge University Press, 2008.
23. S. Goldsmith, A. Aiken, and D. S. Wilkerson. Measuring empirical computational complexity. In *International Symposium on Foundations of Software Engineering*, pages 395–404, 2007.
24. L. Huang, J. Jia, B. Yu, B.-G. Chun, P. Maniatis, and M. Naik. Predicting execution time of computer programs using sparse polynomial regression. In *International Conference on Neural Information Processing Systems*, pages 883–891. 2010.
25. D. Athanasopoulos and B. Pernici. Building models of computation of service-oriented software via monitoring performance indicators. In *International Conference on Service-Oriented Computing and Applications*, pages 173–179, 2015.