Mobile Apps with Dynamic Bindings Between the Fog and the Cloud

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

[1] School of EEECS, Queen's University Belfast, Belfast, UK
{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 controller as a service. The autonomic controller predicts the response time of edge/remote instances of the back-end 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 autonomic controller 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[1].

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[2]. 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 controller-as-a-service* (one for each back-end) that is deployed on the Fog and further acts as a proxy between the front-end and the back-end instances[3]. Each time the front-end interacts with the autonomic controller, 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 controller follows the *control loop* of self-adaptive software [4]. Specifically, the controller monitors the past invocations to the instances, analyses a few *representative* (addressing the efficiency challenge) input datasets, (re-)builds *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 controller-as-a-service 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 autonomic controller 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 the controller. 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 the Fog and the Cloud for the execution of mobile apps have focused on the development or the deployment time of apps. Concerning the non-automated approaches, [6] provides suggestions for mapping back-ends to machines

[1] The front-end includes the programming clients that interact with the service back-end.

[2] The locally stored datasets are synchronized to the remote storage. We do not consider the synchronization time in the current work.

[3] We assume at least an edge and a remote instance have been pre-deployed (whose endpoints are registered to the controller). We leave as future work the decision of the number of instances.

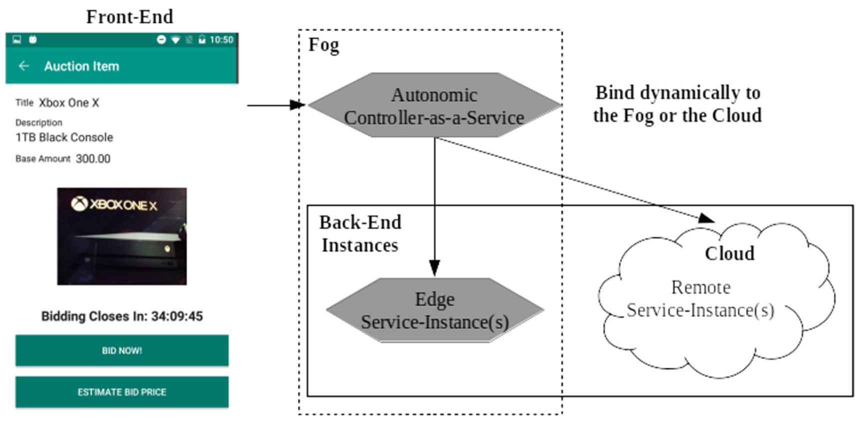

Fig. 1. Auction app extended with the autonomic controller and multiple back-end instances.

at deployment time. The suggestions mainly aim at reducing the network latency, the energy consumption and the financial cost of renting machines. [7] proposes a methodology for assessing the security level of deployment plans.

Regarding the automated approaches, [8–10] 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] produce deployment plans that reduce the power consumption. [15] generates 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, only two approaches monitor the app execution to regenerate deployment plans [17, 18]. However, [17, 18] are reactive (i.e., they suspend the app execution) and lay between apps and operating systems (e.g., redeployment engines). Contrarily, our approach runs at the application layer and pro-actively (without suspending the app execution) self-decides the binding of the front-end.

3 Conceptual Model of Autonomic Controller-as-a-Service

The autonomic controller-as-a-service (Fig. 2) mainly consists of its API (Sect. 3.1), its dynamic binding-mechanism (Sect. 3.2) and its control loop (Sect. 4).

3.1 API of Autonomic Controller

The API offers all of the operations of the back-end. 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.

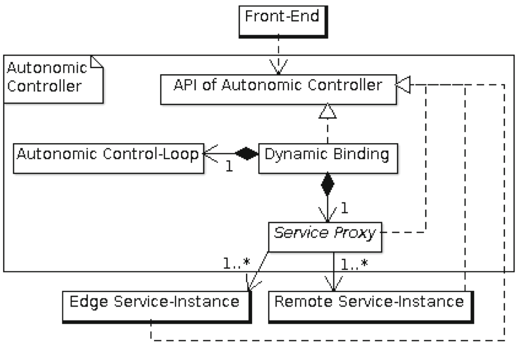

Fig. 2. The UML diagram of the conceptual model of autonomic controller.

Definition 1. *Each operation of the API of an autonomic controller corresponds to a programming method*[4] *of the API of the back-end. Each operation accepts/returns a (possibly empty) set of input/output parameters.*

Given that the XML/JSON schemas of input/output parameters can be easily transformed to programming objects (e.g., JAXB[5]), we define the notion of the parameter 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 parameters or a reference to another parameter.*

Returning to the auction app, we assume the back-end uses the k-means clustering algorithm [22]. The single operation of the API accepts the following parameters: a dataset and the numbers of clusters and algorithm iterations. The last two parameters have a built-in data-type (`int`), while the first parameter has the complex data-type of a list of multi-dimensional data-points (`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 data-type (`Double`).

Based on Definition 2, a parameter has a hierarchical structure. However, only the leaves of the 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). 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) 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.*

[4] The programming methods are explicitly defined in a SOAP-based API or they can be determined by parsing the suffix of the URI of the API in a RESTful API [21].

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

The multi-layer parameter, $p = List < Double[]\ dataPoint > dataset$, in our example is converted to the 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)[6]. Hereafter, we use the term parameter to refer to an unstructured parameter.

3.2 Dynamic Binding-Mechanism of Autonomic Controller

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. 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 response-time. Finally, the `Dynamic Binding` uses the `Service Proxy`[7] to forward an operation call to the selected instance. To this end, the `Dynamic Binding` instantiates the target parameters of the operation of the selected instance via considering the one-to-one mapping between the parameters of the called operation of the controller API and the parameters of the mapped operation of the selected instance. The mapping exists because the controller API is the same with that of the edge and remote instances.

4 Autonomic Control-Loop

The autonomic control-loop of the controller extends the generic Monitor-Analyze-Plan-Execute and Knowledge loop (MAPE-K) of self-adaptive software as follows [4]. Our Monitoring mechanism records the response time of the invocation of edge/remote instances, i.e., the elapsed time between when the invocation is made and when the response is returned back. In other words, the response time is the sum of (i) the execution time of an instance on a machine and (ii) the network latency. The Analysis mechanism (re-)constructs predictive models of response times and expresses them as a function of the input parameters (Sect. 4.1). A separate model is constructed for each instance because instances are usually deployed on different machines. 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 (Sect. 4.2). The mechanism uses the constructed models and the stored parameter-values to predict response times and select the instance that has the lowest predicted response-time. The Execution mechanism invokes the selected instance via implementing a proxy. The latter firstly instantiates the target parameters of the invoked operation

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

[7] 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.

of the instance via considering the one-to-one mapping that exists between the parameters of the called operation of the controller API and the parameters of the mapped operation of the instance (Sect. 3.2). Finally, the constructed predictive-models, the groups of parameter values and the monitoring response-times are stored as the Knowledge of the loop.

4.1 Analysis Mechanism

We firstly define the notions of predictive model and prediction error used by the mechanism. We also specify the algorithmic steps and the time complexity of the mechanism.

The mechanism constructs a separate predictive-model for each instance and especially, for each operation of an instance, as defined below.

Definition 4 (Predictive model of an operation). *The predictive model, p_{op}, of an operation, op, of a service instance 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 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. A widely used technique to build polynomials is the regression technique [24]. However, it takes all possible variable combinations forming long expressions with possibly unneeded terms. To build compact expressions, greedy techniques have been proposed [24,25]. We extend the sparse-term technique of [25]. 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 Algorithm 1. Algorithm 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 Algorithm 1 further include a threshold ω of the lowest prediction-error. Algorithm 1 initially calculates the prediction error (Algorithm 1 (1–4)). If the error is higher than ω, Algorithm 1 rebuilds the predictive model via fitting all of the possible single-variable terms to the past and the current response-times (Algorithm 1 (8)). To fit a term to the response times, Algorithm 1 applies the linear least-square regression-technique [24] (Algorithm 1 (5–10)). Finally, Algorithm 1 selects and returns the term that is dominant and confident (Algorithm 1 (11–20)).

Algorithm 1. Analysis Mechanism

```
Input: $x[D, N], \widehat{y}[D], y(x), \omega$
Output: $y(x)$
1: $e \leftarrow \frac{|y(x[D,N]) - \widehat{y}[D]|}{\widehat{y}[D]}$;
2: if $e > \omega$ then
3:     $T \leftarrow \text{FIT}(x,\ \widehat{y})$;
4:     $y(x) \leftarrow \text{SELECT}(T,\ x,\ \widehat{y})$;
5: function FIT( $x[D, N], \widehat{y}[D]$ ): $T$
6:     for all $1 \le j \le N$ 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:         $T.\text{ADD}(y(x))$;
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 $1 \le i \le D$ 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 $\overline{e} \ge \omega$ then
17:             $min \leftarrow \overline{e}$;
18:             $min_y \leftarrow y(x)$;
19:     $y(x) \leftarrow min_y$;
20: end function
```

Time Complexity. The complexity scales with the numbers D (parameter values) and N (fitted terms), $\mathcal{O}(N * D)$. Since N is much lower than D (we use a sparse-term technique, which is time efficient), the complexity is captured by the expression, $\mathcal{O}(N * D) \approx \mathcal{O}(D)$. Moreover, the complexity does not scale with the number of the back-end instances (even if Algorithm 1 is repeated for each instance), because this number is expected to be (in the order of tens or lower) much lower than D (Sect. 4.2).

4.2 Planning Mechanism

We firstly define the notion of the partition used by the mechanism. We also specify the algorithmic steps of the mechanism, along with its time complexity.

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

Definition 6 (Partitions of the values domain of a parameter). *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:*

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

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

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

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

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

Algorithm 2. Planning Mechanism

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

```
1:  PARTITION( $r$, $v$, $x$ );
2:  $instance \leftarrow$ SELECT($\{instances\}$, $x$, $\{y(x)\}$);
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:             $r_1[i] \leftarrow \Big[x_{min}[i], x_{min}[i] + l\Big]$;
15:             $v_1[i] \leftarrow$ UPDATE($v_1[i],\ r_1[i]$);
16:             for all $2 \leq j \leq Q$ do
17:                 $r_j[i] \leftarrow \Big[x_{min}[i] + l * (j-1) + 1, x_{min}[i] + l * j\Big]$;
18:                 $v_j[i] \leftarrow$ UPDATE($v_j[i],\ r_j[i]$);
19:             $r_Q[i] \leftarrow \Big[\ x_{min}[i] + l * Q, x_{max}[i]\ \Big]$;
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 $len(x)$ is calculated by dividing the range of values of x in Q equally-sized intervals:* $\left\lceil \frac{x_{max} - x_{min}}{Q} \right\rceil$

The representative values (one for each partition) of parameter are defined below.

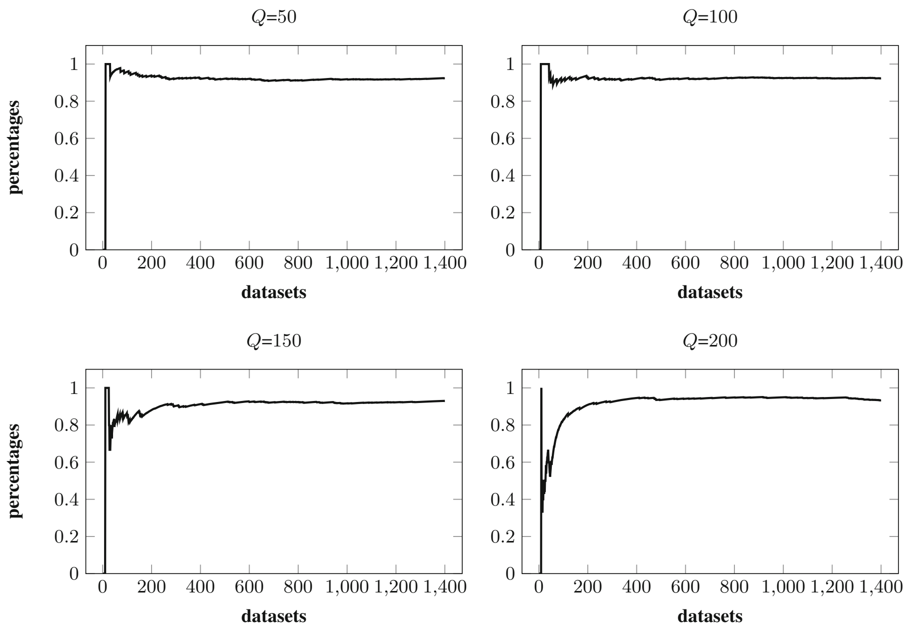

Fig. 3. The percentages of the correct binding-decisions for the auction app.

Definition 8 (Representative values of parameter). *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* $x_j.c \in r_j$.

The Planning mechanism dynamically (re-)defines the partitions (see below) whenever a new parameter-value arrives that is not represented by an existing partition.

Planning Algorithm. Algorithm 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 inputs of Algorithm 2 further include the available instances of the API and the set of the polynomial functions of the current predictive-models of the instances. If the current values of the parameters do not belong to the existing partitions, Algorithm 2 redefines the partitions by adjusting their endpoints (Algorithm 2 (5–11)) and accordingly redistributing their representative values[8] (Algorithm 2 (13–20)). Algorithm 2 adds the current values of the parameters to the proper partitions only if the partitions do not contain values (Algorithm 2 (21–24)). Algorithm 2 predicts the response times of the instances for the current parameter-values and returns back the instance that has the lowest predicted response-time (see Footnote 8).

[8] Due to the algorithmic simplicity, we do not specify the functions UPDATE, ADD and SELECT.

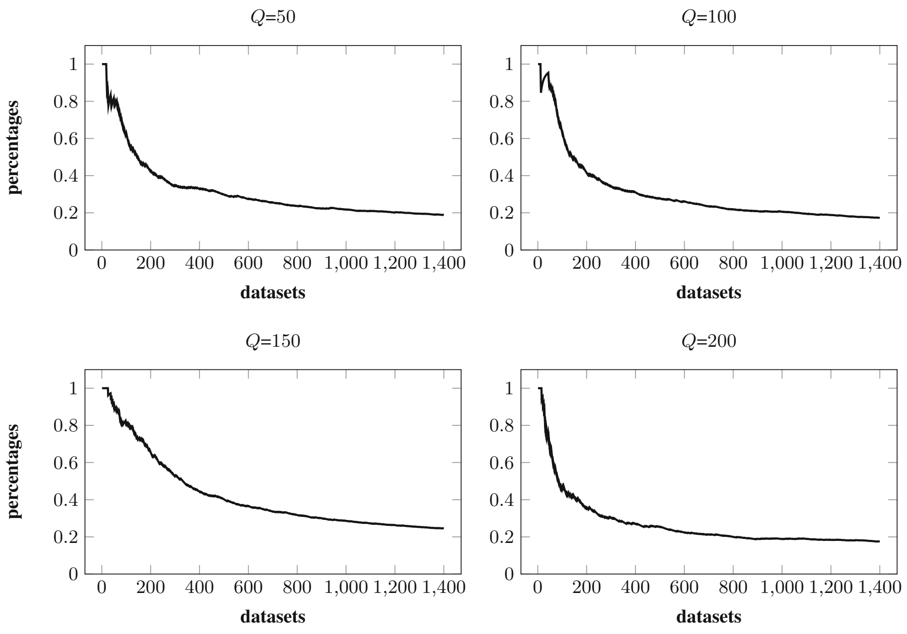

Fig. 4. The percentages of the datasets used for rebuilding predictive models.

Time Complexity. The complexity scales with the numbers N (parameters), Q (parameter partitions), Q (parameters values) and service instances, $\mathcal{O}(N * Q^2)$. Since the number of the instances of a single back-end is expected to be (in the order of tens) much lower than the numbers of the other factors (e.g., Q ranges in the order of hundreds), the complexity does not asymptotically scale with the number of instances.

5 Experimental Evaluation

We evaluate our approach on five benchmarks (Sect. 5.1). Prior to presenting the results, we set up below our experiments.

We extended an existing auction app[9] with a data-analytics back-end that uses the k-means clustering algorithm. We implemented a research-prototype (in Java) of the autonomic controller and we exposed it as a RESTful Web service. Datasets collected from UC Irvine machine-learning public repository[10] 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 Sect. 5.1, 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

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

[10] http://archive.ics.uci.edu/ml/index.php.

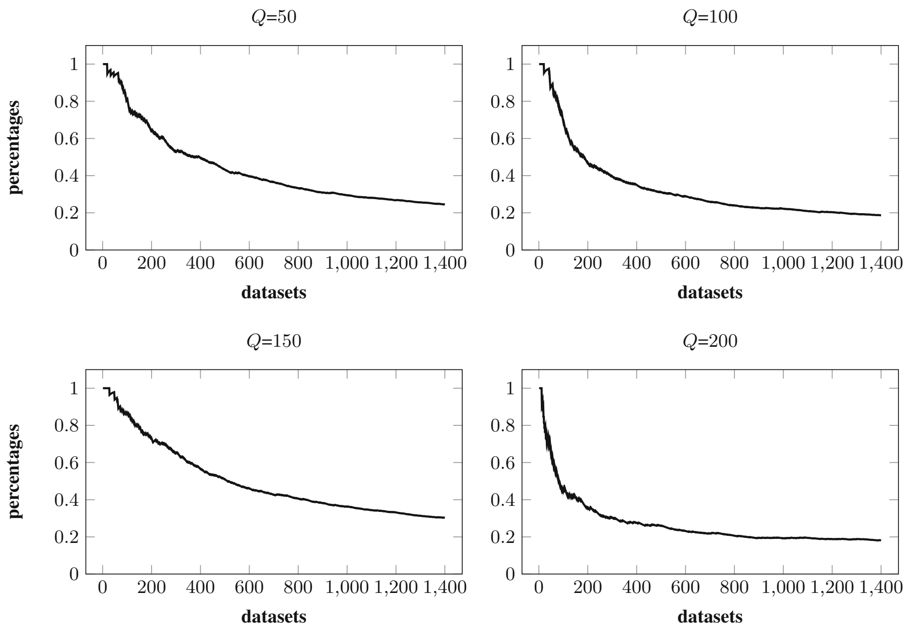

Fig. 5. The percentages of the monitoring response-times used for rebuilding predictive models.

an extra dataset, starting with one dataset and progressing with the remaining datasets in a random order.

The front-end of the app runs on a mobile phone[11], the edge instance and the autonomic controller on a laptop[12] (connected to the same LAN with the mobile phone), and the remote instance on a virtual machine deployed to the Google cloud[13]. All of the instances of the service back-end are exposed as RESTful Web services. We do not present experiments that use multiple edge/remote instances, since the complexity of our approach does not asymptotically scale with the number of instances (Sect. 4.2).

5.1 Evaluation Results

B1. *How many binding decisions are correct*?
For each dataset given as input to the controller, the latter makes a binding decision between the edge and the remote instances. We repeated this experiment for different numbers Q of partitions (Q affects how many times the predictive models are reconstructed). Figure 3 depicts the percentages of the correct binding-decisions (i.e., the number of the correct decisions is divided by the number of all of the

[11] 1.9 GHz CPU, 4 GB RAM, Android 8.0.

[12] 2.70 GHz CPU, Intel Core i5-5257U, 64-bit Windows 10 Home, 8 GB RAM.

[13] 2.2 GHz 2 vCPU, Intel Xeon E5 v4 (Broadwell) platform, 7.5 GB RAM, Windows server 2016 (the cost of renting a more powerful machine for our experiments was very high).

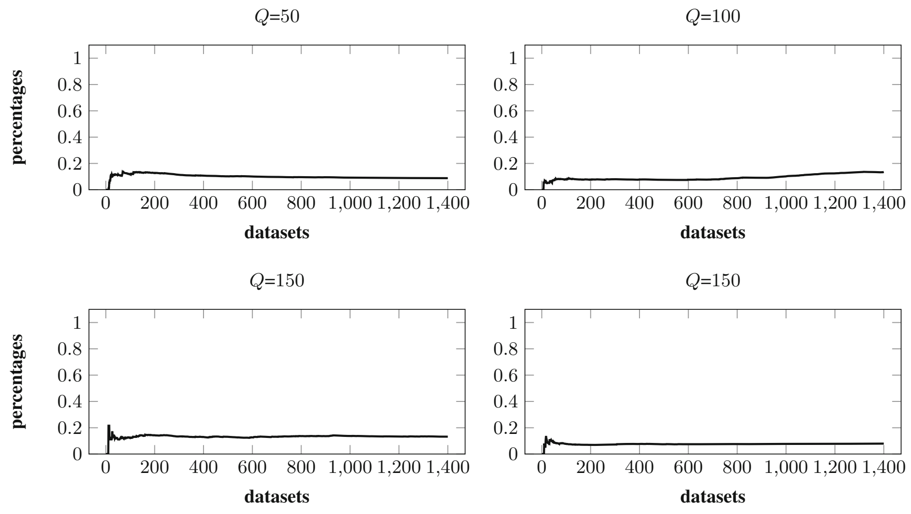

Fig. 6. The overhead added by the autonomic control-loop to the response time of the app.

decisions). We observe from the results that the percentages of the correct decisions are increasing with the increase of the number of the 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 previous experiment (i.e., the number of the used datasets is divided by the number of all of the datasets). The percentages of the used datasets for rebuilding models 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 used for rebuilding models. Thus, the inclusion of extra datasets from the repository in the experiment for rebuilding models would have been redundant.

B3. *How many monitoring response-times are used for rebuilding predictive models*?
Figure 5 depicts the percentages of the monitoring response-times of the edge/remote instances used in the previous experiments (i.e., the number of the used response-times is divided by all of the monitoring response-times). 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 used. Note that the curves of the percentages of the used datasets and the used response-times (Figs. 4 and 5) are analogous.

B4. *What is the overhead to the response-time of the app added by our approach*?
We present in Fig. 6 the percentages of the overhead to the response time of the app added by the autonomic mechanisms (i.e., the execution time of the mechanisms is divided by the response time of the app). The overhead comes to the 10%–15% of the response time of the app. Note that the overhead is higher in the first datasets

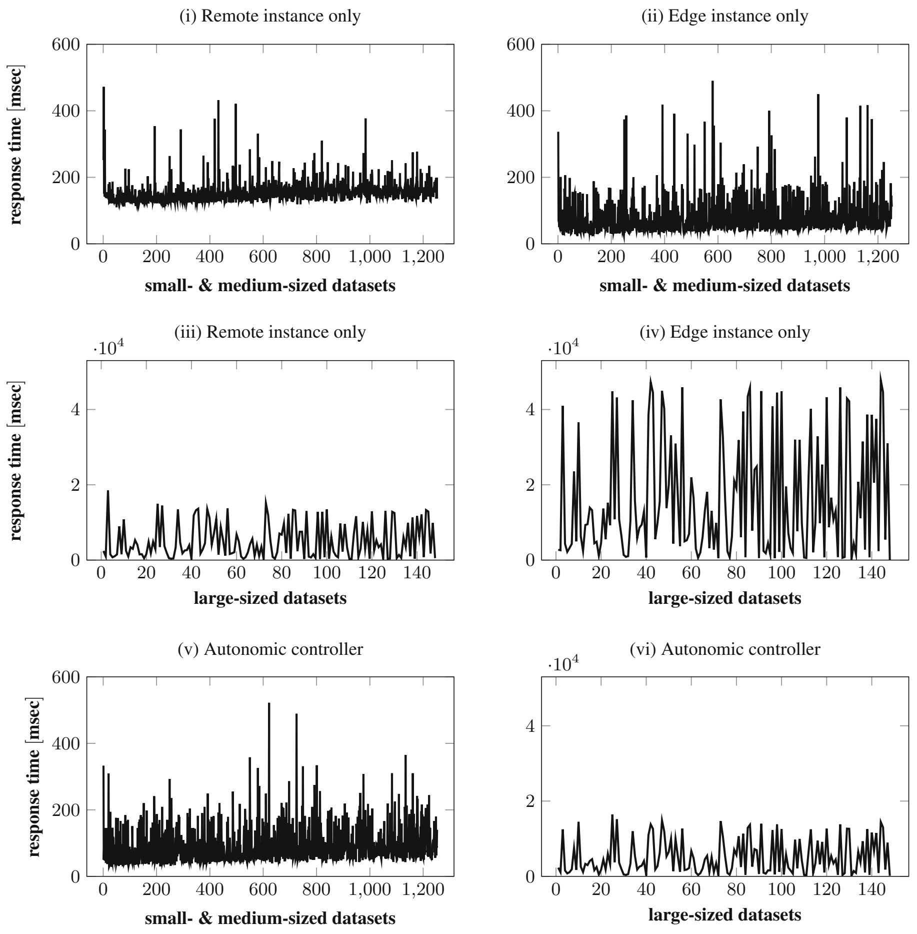

Fig. 7. The response time of the app when it uses the edge/remote instances or the controller.

because the reconstruction of the predictive models and the parameter partitions occur more frequent for these datasets.

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[14]. The first two charts of Fig. 7 present the response times of the app when it uses the remote instance only or the edge instance only for small- and medium-sized datasets. The next two charts present the response times of the app for large-sized datasets[15]. The last two charts present the response times

[14] 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.
[15] The scale of the y-axis in the first two charts is different from the scale in the next two charts.

of the app when it uses the controller. Comparing Fig. 7(vi) against Fig. 7(iii), we observe that the controller selects the remote instance for large-sized datasets. On the contrary, comparing Fig. 7(v) against Fig. 7(ii), we observe that the controller 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 controller selects for small- and medium-sized datasets the edge instance (instead of the original option 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 first 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 to 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 when and where the monitoring measurements were made.

7 Conclusions and Future Work

We contributed with the specification of the conceptual model and the algorithmic mechanisms of an autonomic controller for edge/remote instances of a mobile backend. The evaluation results 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 added overhead comes to the 10%–15% of the response time of an app, and the app response-time 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 and the network properties/load. A final direction is to enhance the autonomic controller 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. Kaur, P., Goyal, M., Lu, J.: Pricing analysis in online auctions using clustering and regression tree approach. In: Cao, L., Bazzan, A.L.C., Symeonidis, A.L., Gorodetsky, V.I., Weiss, G., Yu, P.S. (eds.) ADMI 2011. LNCS (LNAI), vol. 7103, pp. 248–257. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27609-5_16
2. Plebani, P., et al.: Information logistics and fog computing: the DITAS approach. In: International Conference on Advanced Information Systems Engineering, pp. 129–136 (2017)
3. Varghese, B., et al.: Realizing edge marketplaces: challenges and opportunities. IEEE Cloud Comput. **5**(6), 9–20 (2018)
4. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. IEEE Comput. **36**(1), 41–50 (2003)
5. Dheeru, D., Karra Taniskidou, E.: UCI machine learning repository (2017)
6. Ashouri, M., Davidsson, P., Spalazzese, R.: Cloud, edge, or both? Towards decision support for designing IoT applications. In: International Conference on Internet of Things: Systems, Management and Security, pp. 155–162 (2018)
7. Brogi, A., Ferrari, G.L., Forti, S.: Secure cloud-edge deployments, with trust. CoRR, abs/1901.05347 (2019)
8. Brogi, A., Forti, S.: Qos-aware deployment of IoT applications through the fog. IEEE Internet Things J. **4**(5), 1185–1192 (2017)
9. Deng, R., Lu, R., Lai, C., Luan, T.H., Liang, H.: Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. IEEE Internet Things J. **3**(6), 1171–1181 (2016)
10. Brogi, A., Forti, S., Ibrahim, A.: Deploying fog applications: how much does it cost, by the way? In: International Conference on Cloud Computing and Services Science, pp. 68–77 (2018)
11. Mohan, N., Kangasharju, J.: Edge-fog cloud: a distributed cloud for Internet of Things computations. In: Cloudification of the Internet of Things, pp. 1–6 (2016)
12. Mohan, N., Kangasharju, J.: Edge-fog cloud: a distributed cloud for Internet of Things computations. CoRR, abs/1702.06335 (2017)
13. Brogi, A., Forti, S., Ibrahim, A.: How to best deploy your fog applications, probably. In: International Conference on Fog and Edge Computing, pp. 105–114 (2017)
14. Gupta, H., Dastjerdi, A.V., Ghosh, S.K., Buyya, R.: iFogSim: a toolkit for modeling and simulation of resource management techniques in the Internet of Things, edge and fog computing environments. Softw. Pract. Exp. **47**(9), 1275–1296 (2017)
15. Tran, D.H., Tran, N.H., Pham, C., Kazmi, S.M.A., Huh, E.-N., Hong, C.S.: OaaS: offload as a service in fog networks. Computing **99**(11), 1081–1104 (2017)
16. Guo, X., Singh, R., Zhao, T., Niu, Z.: An index based task assignment policy for achieving optimal power-delay tradeoff in edge cloud systems. In: IEEE International Conference on Communications, pp. 1–7 (2016)
17. Yousefpour, A., Ishigaki, G., Jue, J.P.: Fog computing: towards minimizing delay in the Internet of Things. In: IEEE International Conference on Edge Computing, pp. 17–24 (2017)
18. Saurez, E., Hong, K., Lillethun, D., Ramachandran, U., Ottenwälder, B.: Incremental deployment and migration of geo-distributed situation awareness applications in the fog. In: ACM International Conference on Distributed and Event-Based Systems, pp. 258–269 (2016)
19. Richardson, L., Ruby, S.: Restful Web Services, 1st edn. O'Reilly, Sebastopol (2007)
20. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall, Upper Saddle River (2005)
21. Fokaefs, M., Stroulia, E.: Using WADL specifications to develop and maintain REST client applications. In: 2015 IEEE International Conference on Web Services, pp. 81–88 (2015)

22. Smola, A.J., Vishwanathan, S.V.N.: Introduction to Machine Learning. Cambridge University Press, Cambridge (2008)
23. Goldsmith, S., Aiken, A., Wilkerson, D.S.: Measuring empirical computational complexity. In: International Symposium on Foundations of Software Engineering, pp. 395–404 (2007)
24. Huang, L., Jia, J., Yu, B., Chun, B.-G., Maniatis, P., Naik, M.: Predicting execution time of computer programs using sparse polynomial regression. In: International Conference on Neural Information Processing Systems, pp. 883–891 (2010)
25. Athanasopoulos, D., Pernici, B.: Building models of computation of service-oriented software via monitoring performance indicators. In: International Conference on Service-Oriented Computing and Applications, pp. 173–179 (2015)