Self-Adaptive and Online QoS Modeling for Cloud-Based Software Services

Tao Chen, *Member, IEEE* and Rami Bahsoon, *Member, IEEE*

Abstract—In the presence of scale, dynamism, uncertainty and elasticity, cloud software engineers faces several challenges when modeling Quality of Service (QoS) for cloud-based software services. These challenges can be best managed through self-adaptivity because engineers' intervention is difficult, if not impossible, given the dynamic and uncertain QoS sensitivity to the environment and control knobs in the cloud. This is especially true for the shared infrastructure of cloud, where unexpected interference can be caused by co-located software services running on the same virtual machine; and co-hosted virtual machines within the same physical machine. In this paper, we describe the related challenges and present a fully dynamic, self-adaptive and online QoS modeling approach, which grounds on sound information theory and machine learning algorithms, to create QoS model that is capable to predict the QoS value as output over time by using the information on environmental conditions, control knobs and interference as inputs. In particular, we report on in-depth analysis on the correlations of selected inputs to the accuracy of QoS model in cloud. To dynamically selects inputs to the models at runtime and tune accuracy, we design self-adaptive hybrid dual-learners that partition the possible inputs space into two sub-spaces, each of which applies different symmetric uncertainty based selection techniques; the results of sub-spaces are then combined. Subsequently, we propose the use of adaptive multi-learners for building the model. These learners simultaneously allow several learning algorithms to model the QoS function, permitting the capability for dynamically selecting the best model for prediction on the fly. We experimentally evaluate our models in the cloud environment using RUBiS benchmark and realistic FIFA 98 workload. The results show that our approach is more accurate and effective than state-of-the-art modelings.

Index Terms—Software quality, search-based software engineering, self-adaptive systems, machine learning, cloud computing, performance modeling

1 Introduction

Cloud software engineering paradigm is gaining momentum as evident by the tremendous use of cloud-based software services. Software-as-a-Service (SaaS) in the cloud often run on top of a software stack within the Platform-as-a-Service (PaaS) layer [1]. They are also supported by the Virtual Machines (VM) and hardware running at the Infrastructure-as-a-Service (IaaS) layer [2]. To offer scalability and elasticity under changing environment conditions (e.g., workload, size of incoming job etc.), cloud providers often have the capability to dynamically scale various internal control knobs, providing on-demand configurations of software (e.g., threads of service) and hardware resources (e.g., CPU and memory of VM) in a shared infrastructure. In this work, we term both control knobs and environment conditions in the cloud as *cloud primitives*.

The elasticity of cloud has caused a paradigm shift in the way we manage cloud-based software services. However, by design time, it would be difficult for software engineers and cloud engineers to anticipate the dynamic changes in workload and the runtime demands of these cloud-based software services. This fact implies that it becomes more complex to assure the Quality of Service (QoS) during the engineering process. By QoS, we refer to the non-functional attributes (e.g., Throughput) experienced by the end-users of cloud-based software services.

With such context in mind, the key problem, which cloud/service providers face is how to manage runtime QoS by auto-scaling to the best set of control values on the fly. In particular, the fundamental challenge is how to dynamically link QoS with the primitives in cloud, which we address in this paper. The QoS models generally take values of cloud primitive as inputs and predict the likely QoS value as outputs. An accurate QoS model in the cloud can serve as a powerful tool that assists software/cloud engineers or other automated agents to diagnose the cause of violation on QoS requirements; and more importantly, to compare and reason about elastic auto-scaling strategies in the cloud.

The majority of the existing approaches for QoS modeling in cloud has been either *static* (i.e., analytical, e.g., [3], [4] and simulation based, e.g., [5], [6]) or *semi-dynamic*, e.g., [7], [8], [9]. The former is being static in the sense that the expression of models are fixed, and therefore, they are insensitive to the QoS fluctuations at runtime; this is due to the entire modeling process has relied on manual and offine analysis. On the other hand, the semi-dynamic approaches focus on adaptive and dynamic modeling for the magnitude of primitives in correlation to QoS, which means the model changes with respect to the QoS fluctuations. However, their selection of primitives to determine the feature inputs of models has been manual and offine, resulting fixed inputs for the models. Thus, they suffer limited self-adaptivity.

• *The authors are with CERCIA, School of Computer Science, University of Birmingham, Birmingham B15 2TT, United Kingdom. E-mail: {t.chen, r.bahsoon}@cs.bham.ac.uk.*

Manuscript received 20 Jan. 2015; revised 22 Aug. 2016; accepted 6 Sept. 2016. Date of publication 19 Sept. 2016; date of current version 22 May 2017. Recommended for acceptance by G.P. Picco.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSE.2016.2608826

This work is licensed under a Creative Commons Attribution 3.0 License. For more information, see http://creativecommons.org/licenses/by/3.0/

1.1 The Challenges and Current Limitations

In the following, we identify several important challenges for QoS modeling in the cloud, which have not been or have only been partially considered in previous work.

Fine-Grained QoS Modeling. There can be different cloud-based software services running on a VM, each with its own QoS requirements. Fine-grained QoS modeling is challenging as more heterogeneity (e.g., derivatives of QoS requirements and service characteristics etc.) need to be considered. However, existing static and semi-dynamic modeling tend to focus on the mean QoS of the entire VM, limiting the model accuracy for individual software service.

Dynamic and Uncertain QoS Interference. QoS modeling in the cloud suffers from the problem of QoS interference. QoS interference refers to scenarios where a software service exhibits wide disparity in its QoS performance that depends on the dynamic behaviors of its neighbors. In particular, we distinguish two major causes of interference, these are: co-located service interference and co-hosted VM interference. The former is an inherent issue from the traditional cluster computing, where multiple applications/services running on the same operating system can suffer contention on the shared memory/cache, and therefore cause interference [10]. This is also true for multi-core systems [11]. The latter, on the other hand, is a significant unique problem in cloud computing, where virtualization has been used as the basis. This is because in such scenario, certain aspects of the underlying infrastructure are shared amongst the co-hosted VMs on a machine (e.g., last level cache of CPU and memory bandwidth), henceforth it can result in contention and create the chances for interference, as evident by many recent work [12], [13], [14]. Given that it can be extremely difficult to completely eliminate QoS interference or it can be too expensive to do so [13], it is crucial to consider and handle the interference when modeling QoS in the cloud. Here, the challenge lies in the difficulty to dynamically incorporate information about interference in the modeling, especially when the QoS interference is dynamic and uncertain in nature—it is difficult to know when contention would occur and what the degree of such contention is. However, existing work either considers co-hosted VM interference only (e.g., [13]) or completely ignores the presence of QoS interference (e.g., [7]), which is unrealistic.

Dynamic and Uncertain QoS Sensitivity. The core of QoS modeling is how to model its sensitivity with respect to the primitives in cloud. By QoS sensitivity, we are interested in *which* (e.g., are CPU and Throughput correlated?), *when* (i.e., at which point in time they are correlated?) and *how* (i.e., the magnitude of primitives in correlation) the primitives correlate with QoS. Given the nature of cloud, QoS sensitivity is dynamic and uncertain, i.e., runtime changes occur in terms of *which*, *when* and *how* primitives correlate with QoS. Specifically, the challenges of QoS sensitivity in the modeling can be attributed to two important phases, namely *primitives selection* and *QoS function construction*:

1) Primitives Selection: To model QoS and its sensitivity in the cloud, a fundamental task is to adaptively determine what are the primitives that should be used as feature inputs of the model (i.e., which and when the primitives correlate with QoS). To show a simple example of the dynamics and uncertainties in primitives selection, in Fig. 1, we vary the workload of a service while keeping that of the co-located services and co-hosted VMs unchanged, we can see that the Response Time of the said service tends to be insensitive to CPU at the beginning hence it cannot provide relevant information about the QoS. However, after the *18th* interval, the Response Time gradually become more affected by the CPU as the workload change by time, which is uncertain in nature; this becoming even more true in the cloud when there is uncertain QoS interference, i.e., the workload of neighbor services/VMs changes. Therefore, the primitives selection needs to cope with the dynamics and uncertainties in QoS sensitivity. Given that the selected inputs have a great impact to the model accuracy (as we will show in Section 4), it is important to select a right set of primitives. In particular, too limited inputs may not provide enough information of relevance to the QoS (i.e., the information that drives the changes in QoS), which restricts the model accuracy and applicability. On the other hand, too many inputs can generate noise in the modeling, because it introduces irrelevant information and large redundancy in the inputs (i.e., the same information has been provided by more than one selected primitives, thus it becomes noise), this will downgrade the model accuracy [15] and generate unnecessary overhead. Though some machine learning algorithms are proved be be resistant to noise, e.g., those with regularization [16], we believe that the benefits gained from primitives selection is vast, e.g., improved accuracy, more intuitive model and faster modeling time. The challenge here is how to dynamically select the most significant set of primitives as inputs, including the information of QoS interference from neighboring services and VMs, which provides good model accuracy and adequate complexity.

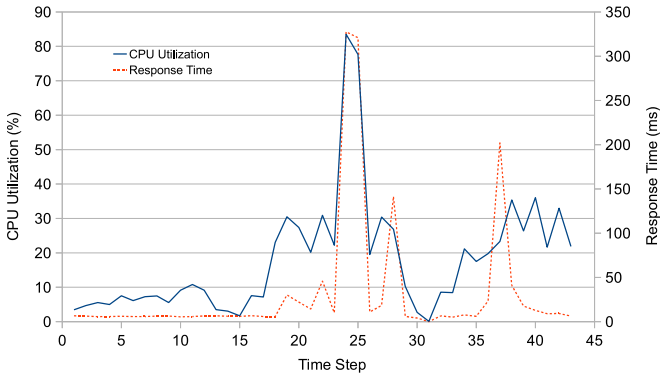


Fig. 1. The exampled fluctuations of CPU utilization and Response Time in cloud.

Nevertheless, existing static and semi-dynamic approaches for QoS modeling in the cloud rely on fixed and manual analysis to select the primitives as inputs, which are often offline. A widely applied approach is to reduce the possible primitives space based on empirical observations and domain specific assumptions, e.g., most work [7], [17], [18] consider only hardware resources. However, this may mislead the QoS modeling as it can ignore some highly relevant features, e.g., the software configurations, which can interplay with the hardware provision to influence QoS [8], [19], [20]. In addition, ignoring QoS interference can result in inaccurate models. Even though the offline and manual selection is achieved at a good accuracy, the runtime dynamics and uncertainties can become a problem as there is no guarantee that the selected primitives are the best for the entire service life time. Until recently, few

techniques [8], [9] have been proposed for self-adaptive primitives selection in the cloud. However, they implicitly tackle redundancy and regard each primitive equally in the selection. We refer these techniques as single-learner in the remaining of this paper. In Section 4, we will show why these single-learner based techniques tend to be limited in accuracy.

2) QoS Function Construction: Another important task in modeling QoS and its sensitivity is to adaptively determine how the primitives correlate with QoS by means of mathematical function. To show a simple example of the dynamics and uncertainties in QoS function construction, we use the above setup in Fig. 1. As we can see, from the *18th* interval onwards, the Response Time of the service is becoming more sensitive to CPU till *30th* where the sensitivity is starting to decrease. This shows that the Response Time is always sensitive to CPU for the period, but the magnitude tends to be different depends on the uncertain changes of workload from time to time. Again, this becomes more complex in the cloud when it involves changing workloads of neighbor services/VMs. All These facts imply that the modeling needs to be able to handle the dynamic and uncertain magnitude of primitives in the correlation, which is a challenge. Consequently, the static QoS modeling approaches tends to be insufficient, because the effectiveness of these approaches is restricted by their simplified and fixed assumptions on the environment and service's internal operations [7]. On the other hand, the semi-dynamic approaches are capable to handle this challenge as they are grounded on sound machine learning algorithms, which are dynamic and self-adaptive in nature. These approaches are single-learner based as they rely on a single learning algorithm. Nevertheless, in Section 5, we will show that a single learning algorithm can be suitable only for certain QoS trends. Consequently, a significant drawback of these approaches is that, for any given scenarios, they require the engineers to predetermine the suitable learning algorithm. This can entail manual and intensive investigation rendering it as an expensive process. Moreover, a predetermined algorithm does not cater for unexpected or envisioned changes in QoS at runtime. Now, the challenge becomes how to efficiently and dynamically determine the best learning algorithm for a scenario.

1.2 The Contributions and Organization

In this paper, we report on a set of integrations and extensions to our prior work [21], [22]. To overcome the aforementioned challenges in the cloud, we present a QoS modeling approach, which is fully dynamic, self-adaptive and capable for online modeling. Since our modeling approach does not rely on any assumptions of the software service's internal structure, the resulted model are agnostic to the type of software service hosted within VM. Specifically, our novel contributions can be summarized as the follows:

First, we have described the unique challenges to the online QoS modeling in the cloud (as in Section 1.1).

Second, we abstract a fine-grained and generic QoS model to handle dynamic and uncertain QoS sensitivity; and to incorporate information of the uncertain QoS interference caused by the software services co-located on a VM and the VMs co-hosted on a Physical Machine (PM).

Third, we present an in-depth analysis on the correlations of selected primitives to the model accuracy for primitives selection in the cloud; in particular, we show how the model accuracy can be affected by the cumulative changes of the information relevance to QoS and the information redundancy of the selected primitives. We discovered that, without special treatment, these cumulative changes cannot correctly quantify the effects of primitives to model accuracy for the entire input space. Drawing from the observations obtained on this analysis, we propose a self-adaptive and online technique, namely hybrid dual-learners, to determine *which* and *when* primitives correlates with the QoS on the fly. The idea is that we aim to select the most significant set of primitives which can improve accuracy in the modeling. To avoid misleading caused by the cumulative changes, we partition the possible primitives space into two sub-spaces; the learner in each sub-space uses different primitives selection techniques based on symmetric uncertainty [23] and the selected sets of these two learners are combined. We design four variations of our technique, each of which uses different formulations to express the difference between relevance and redundancy of the selected primitives.

Fourth, we present a suitability analysis of different learning algorithm for QoS function construction on different QoS attributes. Particularly, we have examined three widely used learning algorithms as exemplars, these are: Artificial Neural Network (ANN) [24], Auto-Regressive Moving Average with eXogenous inputs model (ARMAX) [25] and Regression Tree (RT) [26]. We discovered that a single learning algorithm can perform significantly different depends on the case. Motivated by this fact, we develop a self-adaptive and online solution, namely adaptive multi-learners, to dynamically model *how* the primitives correlates with the QoS. Precisely, multiple learners that apply different learning algorithms are used to build a bucket of models. By doing so, the proposed solution is not only able to dynamically correlate the selected primitives to the QoS, but also to adaptively select the best learning algorithm and its resulted model during prediction in cloud.

Fifth, we implement our modeling approach based on an autonomic architecture in the cloud. We experimentally evaluate the approach under four commonly used QoS attributes, these are: Response Time, Throughput, Availability and Reliability. In addition, we have used the well-known RUBiS [27] benchmark and the FIFA 98 [28] workload to assess various aspects of our approach, including accuracy, stability, sensitivity to the online data size and efficiency. The results reveal that our approach is overall more accurate, more stable and reduce the error quicker than the other approaches; while generating acceptable overhead.

The paper structure is organized as the follow: Section 2 decomposes the problem of QoS modeling and presents the model. Section 3 describes our architecture and overview of the approach. Section 4 specifies the hybrid dual-learners approach for primitives selection; and the analysis about how the relevance and redundancy of selected primitives influence model accuracy, which drive our designs. Section 5 illustrates how different learning algorithms perform under different QoS attributes and fluctuations; we then specify the adaptive multi-learners technique for QoS function construction. Subsequently, we report on the experiments and evaluation in Section 6. Sections 7, 8, and 9 present threats to validity, related work and conclusion respectively.

TABLE 1
The Basic Notations for QoS Modeling in Cloud

S_{ij}	The jth service-instance of the ith concrete service.
$QoS_k^{ij}(t)$	The kth QoS attribute of S_{ij}, and its value (e.g., mean response time) at interval t.
f_k^{ij}	The QoS function for the kth QoS attribute of S_{ij}.
$SP_k^{ij}(t)$	The selected primitives matrix of S_{ij} at t, its column contains the most relevant and significant inputs for the QoS, including the primitives that tend to directly influence the QoS (e.g., the threads of the corresponding service-instance); and the primitives that belong to the co-located service-instances and the co-hosted VMs. The row indicates the number of order, denoted as q, which represents how many historical data points need to be used as inputs for improving model accuracy.
δ	Any other inputs, e.g., historical time-series QoS points and tuning variables etc., that improve model accuracy.
$CP_a^{xy}(t)$	The value of the ath control primitive for S_{xy} at interval t, e.g., CPU, memory and thread etc.
$EP_b^{mn}(t-1)$	The value of the bth environmental primitive for S_{mn} at interval t-1, e.g., workload etc.

2 Models and Problem Description

2.1 Cloud System Model

We assume that cloud-based applications are composed of services, each has different QoS requirements and external environment changes (e.g., changes in workload). Often, multi-tiers applications and services in the cloud can have multiple replicas for various purposes, e.g., service differentiation and load balancing etc. Therefore we assume that each tier in a multi-tiers application, consisting of concrete services $S_1, S_2, \ldots, S_i$, can have multiple replicas deployed on different VMs even PMs. In this work, we refer to the replicas of concrete services as *service-instances*: the jth service-instance of the ith concrete service is denoted by S_{ij}. Multiple service-instances are deployed on a cloud software stack running on VM, which can be setup using various control knobs. These control knobs can be either shared amongst the service-instances running on a VM (e.g., CPU of the VM) or specific to one service-instance (e.g., threads of a service-instance). The basic notations used in this section are listed in Table 1.

Unlike existing work, which focus on modeling for the entire application and VM, we aim to create fine-grained QoS models for each service-instance. In particular, the resulted models should cope with the QoS interferences at both inter-VMs and inter-services level. In addition, instead of modeling the effect of VM-level provisioning (i.e., add/remove a VM), we focus on the effect of fine-grained provisioning and configuration inside VM (e.g., CPU of a VM and/or thread of a service-instance). This would provide more flexible use of the model, e.g., for vertical scaling. It is wise to consider vertical scaling before horizontal scaling (i.e., add, remove or migrate VMs) as the former is often much more efficient than the latter.

It is worth noting that, apart from the co-hosted services and co-located VMs, QoS interference can also occur due to contention on the functionally dependent services. For instance, S_{11} and S_{31} (both running on different PMs) can be both dependent on S_{21} (e.g., a database service). This implies that S_{11} and S_{31} incur QoS interference. However, we discovered that in such case, the primitives of $S_{31}1$ tend to be insignificant in the QoS modeling of S_{11} as the same information has already been expressed by the primitives of S_{21}, which is also part of the invocation. As a result, we consider the co-hosted services and co-located VMs as the primary causes of QoS interference.

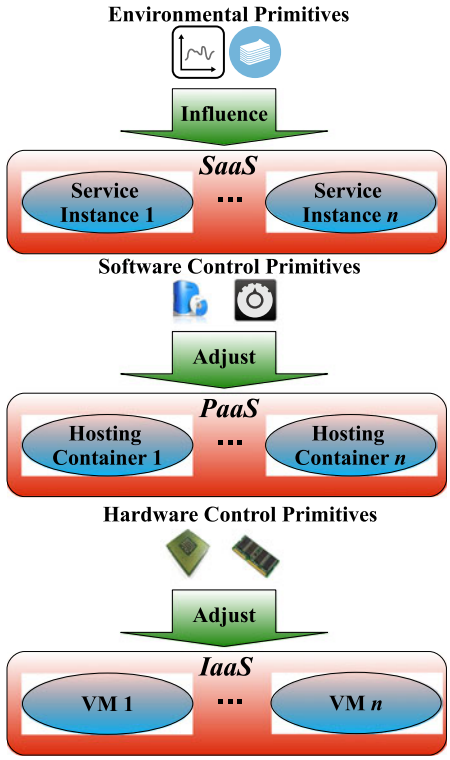


Fig. 2. Overview of the cloud primitives.

2.2 The Cloud Primitives for Building Models

The primitives in cloud serve as the fundamental inputs of a QoS model. Without lose of generality, we decompose the notion of primitives into two major domains: these are *Control Primitive (CP)* and *Environmental Primitive (EP)*. Control Primitives are the internal control knobs and can be either software or hardware, which can be managed by the cloud providers to support QoS. Specifically, software control primitives are software tactics and the key configurations in cloud; such as the number of threads in thread pool of service/application, the buffer size and load balancing policies etc. Whereas, hardware control primitives are computational resources, such as CPU and memory. As shown in Fig. 2, software and hardware control primitives rely on the PaaS and IaaS layers respectively. In particular, it is non-trivial to consider software control primitives when modeling QoS in the cloud as they have been shown to be highly relevant features for QoS [8], [19], [20]. On the other hand, Environmental Primitives refer to the external stimulus that cause dynamics and uncertainties in the cloud; for examples, workload and unpredictable incoming data etc. If the cloud provider is able to control the presence of the stimulus, then these can be considered as control primitives.

To improve accuracy and prevent noises, selecting the right primitives as inputs is critical for QoS modeling in the

cloud. However, the difficulty is that the primitive inputs, which are relevant and useful for modeling QoS, tend to be dynamic. In such context, possible inputs of a QoS model can be the primitives that tend to directly influence the QoS (e.g., the threads of the corresponding service-instance); it can also include the primitives that belong to the co-located service-instances and the co-hosted VMs; Specifically, all possible primitives inputs for modeling the QoS attributes of a service-instance form a space, which is termed *possible relevant primitives space*. This space can be defined by:

Rule 1. *A primitive belongs to the possible relevant primitives space for modeling the QoS of S_{ab} if it can be classified into one of the following groups:*

1) *It is a software control or environmental primitive of S_{ab}.*
2) *It is a hardware control primitive of the VM that runs S_{ab}.*
3) *It is a software control or environmental primitive of S_{cd}, given that S_{ab} has direct functional dependency on S_{cd}.*
4) *It is a hardware control primitive of the VM that runs S_{cd}, given that S_{ab} has direct functional dependency on S_{cd}.*
5) *It is a software control or environmental primitive of S_{cd}, given that S_{cd} is co-located with S_{ab} on the same VM.*
6) *It is a hardware control primitive of a VM, which is co-hosted with the VM that runs S_{ab}.*

The problem here is how to select on the fly a right subset of primitives from the space as the inputs of QoS models. The aim is to improve the model's accuracy by taking relevance and redundancy of the primitives into account. In Section 4, we will present detailed analysis and solution for selecting the right primitives.

Another important decision to mention is that, for each control primitive, we need to decide on whether the configuration value or the demand value should be used in the modeling. By configuration value, we refer to the upper/lower bound of control primitive. However, it is generally impossible to guarantee that the configured value (e.g., CPU cap) can be fully utilized. Such fact obfuscates the sensitivity of QoS to its primitives as using the configuration values to model QoS would take those idle proportions of provisions into account. As a result, using configuration values as inputs is ill-suited in our case. To cope with this issue, we apply the demand values of control primitives (e.g., real-time percent usage of CPU) as inputs, which better reveal QoS sensitivity. Moreover, modeling QoS with demand values implies that our model is likely to determine the minimal requirement of configurations for achieving certain QoS objectives. This will potentially improve the elasticity of software configuration and hardware provision in cloud, when our modeling approach is used in cloud management. It is worth noting that certain dimensions of control primitives (e.g., thread) can be controlled for each service-instance individually, whereas others (e.g., CPU and memory) are shared on a VM, in which case an identical value would be used for modeling the QoS of all service-instances deployed on such VM.

Instead of using multiple metrics for each primitive and QoS, e.g., CPU percentage and instructions-per-second for measuring CPU of a VM, we follow the state-of-the-art assumption [7] that only one metric is used for each primitive and QoS in the modeling; the proper metric can be chosen by the software/cloud engineers based on certain constraints in the cloud environment, e.g., whether it is supported by the hypervisor. We leave the study of multidimensional metrics as future work.

2.3 Dynamic and Interference Aware QoS Model

To tackle the aforementioned challenges of QoS modeling in the cloud, we define a generic QoS model. Formally, the model at the tth sampling interval is expressed as

$$QoS_k^{ij}(t) = f_k^{ij}(SP_k^{ij}(t), \delta), \tag{1}$$

where $QoS_k^{ij}(t)$ is the kth QoS attribute of S_{ij}, and its value that used in the modeling is represented by a given metric (e.g., mean Response Time) at t. f_k^{ij} is the QoS function for the kth QoS attribute of S_{ij}, and it is changed at runtime using learning algorithms, as we will see in Section 5. δ refers to any other inputs (e.g., historical time-series QoS points and tuning variables etc) required by the algorithm to train apart from the cloud primitives. We denote the input $SP_k^{ij}(t)$ in (1) as the selected primitives matrix of $QoS_k^{ij}(t)$ at t, formally depicted in (2)

$$SP_k^{ij}(t) = \begin{pmatrix} CP_a^{xy}(t) & \cdots & EP_b^{mn}(t-1) & \cdots \\ \vdots & \ddots & \vdots & \ddots \\ CP_a^{xy}(t-q+1) & \cdots & EP_b^{mn}(t-q) & \cdots \end{pmatrix}. \tag{2}$$

This matrix contains the primitive inputs of $QoS_k^{ij}(t)$ which are dynamically selected from the possible relevant primitives space for the QoS attributes of S_{ij}, as we will see in Section 4. More concretely, the column entries indicate the selected primitives for the QoS. $CP_a^{xy}(t)$ denotes the ath control primitive of S_{xy} and $EP_b^{mn}(t-1)$ means the bth environmental primitive of S_{mn} respectively. The actual values of $CP_a^{xy}(t)$ and $EP_b^{mn}(t-1)$ in the modeling are measured by given metrics (e.g., expected CPU percent usage and mean request rate) at t and $t-1$, respectively. q determines the number of row entries, which indicates the use of how many historical time-series points of the selected primitives as inputs. We observed that the best value of q depends on the learning algorithm that trains f_k^{ij}; in particular, it is better to set q as constant for certain algorithms (e.g., $q = 1$ for ANN and RT); however for the others (e.g., ARMAX), we found that q should be determined during training via hill-climbing optimization, which starts with $q = 1$, then automatically increase the number of row entries one by one during training till the accuracy cannot be further improved. To improve numeric stability for both continuous and discrete data, we normalized all data values to the range between 0 and 1 before the modeling.

It is easy to see that (1) and (2) provide generic and intuitive formulations for modeling QoS in the cloud. Precisely, to model $QoS_k^{ij}(t)$, the objective of our fully dynamic, self-adaptive and online modeling approach consists of two-phases: (i) a primitives selection phase that determines the content of $SP_k^{ij}(t)$ at runtime; and (ii) a QoS function construction phase that trains function f_k^{ij} on the fly.

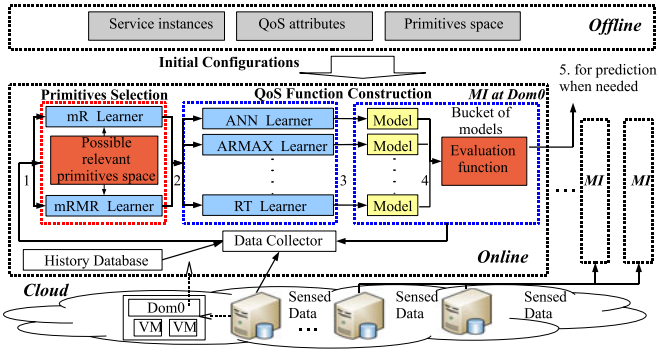


Fig. 3. Overview of the modeling approach in the cloud.

3 Overview of the Modeling Approach in Cloud

As shown in Fig. 3, the approach is realized as middleware using autonomic architecture with a feedback loop. The service-instances running on the VMs of a PM are managed by a dedicated Middleware Instance (MI), which is attached to the root domain (e.g., Dom0 [29]) of this PM. Each MI is self-adaptive as the feedback loop runs continually to keep the models updated.

Our approach is designed for online scenarios; the only offline preparation is to define the current service-instances, their QoS and classification of the primitives in the spaces (i.e., using Rule 1). This preparation can be easily done by the software/cloud engineers and it should be updated accordingly if changes occur. The approach can be also used offline in situations where conducting offline modeling in advance can be beneficial to the online models. Within the feedback loop, *Data Collector* continually monitors and stores data samples of QoS and primitives from the service-instances/VMs running on a PM, and those from the other PMs in the presence of functional dependency. This can be achieved by accessing the cloud sensors or log files. It is worth noting that the modeling interval can be longer than the sampling interval; that is to say, the frequency of data collection do not need to be the same as the frequency of modeling, in which case the sampled data can be stored in a history database and retrieved when needed.

Upon each modeling interval, for each QoS attribute of a service-instance, all historical data is then passed to the primitives selection phase for determining which and when primitives correlate with QoS at runtime (step 1). Here, we have used two learners to select primitives from two sub-spaces as motivated by our analysis in Section 4. At step 2, the selected sets of primitives are combined and sent to the QoS function construction phase, where multiple learners are used to model how the primitives correlate with QoS online (step 3). At step 4, each QoS attribute is associated with a bucket of models produced by candidate learners and an evaluation function; in addition, the weights in the evaluation function will be updated. This bucket can be then used by, e.g., an *Autoscaler* for performing prediction at any time (step 5). Upon prediction when given a set of inputs, the evaluation function is used to select the best model in the bucket (see Section 5).

4 Primitives Selection in the Cloud

As shown in (1) and (2), to dynamically model $QoS_k^{ij}(t)$ at runtime, we first determine *which* and *when* the underlying primitives should be included as column entries in $SP_k^{ij}(t)$ for the QoS modeling.

One straightforward solution to the primitives selection problem is to search the best set of primitives using a given learning algorithm that guarantee to produce the best accuracy for the said algorithm; this is regarded as the *wrapper* approach [30]. Nevertheless, given that the learning algorithm needs to be run many times during the selection process, it is clear that such approach can introduce large overheads in terms of both resource and latency. As a result, the wrapper approach is ill-fit for online QoS modeling in the cloud. In this work, we focus on an alternative approach that is more efficient and capable to select primitives independent of the learning algorithms, namely the *filter* [30].

Traditionally, selecting the primitives as model inputs for QoS modeling in the cloud has been done using fixed and manual sensitivity analysis (e.g., [7], [17]), or single-learner based approach [8] as they consider all the possible primitives in the space equally. However, there has been no explicit definition of the objectives for primitives selection process in the cloud; it has been implicitly known as to select certain relevant primitives (e.g., the top two relevant primitives) for modeling QoS without considering redundancy. In addition, existing primitives selections in the cloud are mostly driven by empirical observations and domain specific assumptions— there has been no explicit or quantitative studies about the correlation of selected primitives to the model accuracy. In particular, it is not clear how the relevance and redundancy of selected primitives can affect the accuracy when modeling QoS in the cloud.

In this section, we clearly define the objectives of primitives selection for QoS modeling in the cloud. We also present a set of experimental analysis on the relevance and redundancy of selected cloud primitives in relation to model accuracy. Driven by the observations from the conducted analysis, we propose a self-adaptive and online solution for primitives selection, namely hybrid dual-learners.

4.1 The Objectives in Primitives Selection

We define two main objectives for the primitives selection, namely: selecting relevant primitives and selecting useful primitives from the possible relevant primitives space. It is well-known that one set of primitives can result in better model accuracy than another set, given that both sets have the same relevant primitives of the QoS and the former has no or less irrelevant primitives (i.e., those that cannot influence QoS) than the latter [30]. Therefore the aim of the first step in primitives selection is to select all relevant primitives, we call this as the problem of *relevant primitives* selection and it can be easily resolved by using relevance measurement, as we will show. However, selecting only the relevant primitives is likely to have rich redundancy in the selected primitives, which can negatively affect the model accuracy [15]. Therefore, the crucial challenge is how to select an even better set after the irrelevant primitives have been eliminated, considering each relevant primitive can be good for providing relevance, but bad for having

redundancy with each others. We refer to this problem as the *useful primitives* selection problem. In this problem, our aim is to select a set of primitives that improve the model accuracy. In particular, this can be achieved by reaching a right balance between relevance and redundancy. It is easy to see that a useful primitive is definitely a relevant primitive, but the reverse is not always true.

4.2 Quantifying Relevance and Redundancy

To quantify the relevance of a primitive to the QoS and the redundancy between a pair of primitives, we have used Symmetric Uncertainty (SU), which is a fundamental concept in information theory [23]. SU measures the degree of relevance between two time series variables by producing a value ranges from 0 to 1, where a greater value implies higher relevance. At one extreme, the value between a QoS attribute and a primitive is 1 indicating that all information of the primitive is correlated with the QoS (and vice versa). At the other extreme, the value of 0 implies that changes in the primitive's behavior are independent of that of the QoS (i.e., irrelevant primitive). Formally, the symmetric uncertainty between two discrete, time-series variables is calculated by

$$U(X,Y) = \frac{2 \times I(X,Y)}{H(X) + H(Y)} \tag{3}$$

$$I(X,Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p(x) \times p(y)} \right) \tag{4}$$

$$H(X) = -\sum_{x \in X} p(x) \log \left(p(x) \right), \tag{5}$$

where X and Y are the value vectors of a primitive (e.g., CPU) and a QoS attribute (e.g., Throughput), respectively; $I(X,Y)$ shows the formula for mutual information between them and $H(X)$ expresses entropy (we have used 2 as the log base). x and y refer to a pair of primitive and QoS value at the same sampling interval.[1] $p(x,y)$ is the joint probability between the values in a pair; $p(x)$ is the marginal probability of a particular primitive (or QoS) value. In the following, we call a primitive as relevant primitive to a QoS attribute if it results in non-zero SU value to such QoS. By using (3), it is straightforward to measure the relevance of a primitive to QoS. As for redundancy, we consider it as the relevance between a pair of primitives, which can be also easily quantified using (3).

Bear in mind that a single SU value is very helpful for filtering the primitives that have relevance below threshold (e.g., filter irrelevant primitives), if it is known that these primitives can cause downgrade of the model accuracy; it is also useful for conducting pair-wised comparison on the relevance of two individual primitives to the QoS; or for comparing the redundancy of two individual primitives to the other same primitives. It is known that these comparisons can provide correct information about the relative effects of relevance and redundancy of two individual primitives to the model accuracy [15]. That is to say, it is known that (i) A can help to produce better model accuracy than B if B has zero SU value to the QoS while A has non-zero value; or (ii) A can provide better accuracy than B if A is more relevant (greater SU value) to the QoS and both of them has the same redundancy value to each selected primitives; or (iii) if there is only one selected primitives C, then A can provide better accuracy than B given that the redundancy between A and C is less than that between B and C (i.e., smaller SU value), in addition, A and B has the same relevance to the QoS.

Nevertheless, the single SU value and pair-wised comparison are insufficient for selecting useful primitives as they cannot consider both relevance and redundancy simultaneously in the selection. In addition, it cannot properly quantify the effects of combinatorial relevance and redundancy to model accuracy for a whole set of selected relevant primitives. This means given two sets of selected relevant primitives, such comparison cannot determine which set will produce better model accuracy during the selection. Our problem requires a measurement that copes with those issues. As a result, we need to study and select useful primitives by comparing the cumulative representation of relevance and redundancy for any possible sets of selected relevant primitives.

There can be two forms of cumulative representation: first we can consider multivariate probability distribution for a given set of selected relevant primitives, in which case (3) would be changed into the following formula:

$$U(X_1, X_2, \ldots, X_n, Y) = \frac{2 \times I(X_1, X_2, \ldots, X_n, Y)}{H(X_1, X_2, \ldots, X_n) + H(Y)}, \; X_n \in S, \tag{6}$$

where $[X_1, X_2, \ldots, X_n]$ denotes vectors of n different primitives that has been selected; and S denotes the set of selected primitives. (6) expresses both relevance and redundancy as they can be handled by the multivariate probability functions. However, this method has some serious drawbacks: (i) the number of online data samples can be insufficient for correctly calculating the probability and (ii) the multivariate joint probability calculation often involves computing the inverse of the high-dimensional covariance matrix, which is computationally expensive and thus it is an ill-suited solution in our case. Alternatively, we can compute the cumulative SU values of relevance and redundancy. By cumulative SU values, we refer to the cumulative combination (i.e., total or average) of the single SU values for the primitives in a given set of selected relevant primitives [15]. An example of relevance is shown below

$$\textit{Relevance of a selected set} = \sum_{X \in S}^{n} U(X,Y). \tag{7}$$

This cumulative combination involves a bivariate probability only and thus it is more appropriate for filtering at runtime. In addition, it is highly intuitive and the nature of cumulative combination implies its light computational efforts. The cumulative representation for redundancy can be similarly applied. In this work, we call these representations as cumulative relevance and cumulative redundancy.

Recall that in selecting useful primitives, we aim to improve the model accuracy by balancing the relevance and redundancy of selected primitives. With this in mind, it is

1. To preserve simplicity and avoid the expensive calculation of continuous mutual information, we firstly normalise the continuous time series data to the range of 0 and 100, we then discretize it by rounding each value to the nearest integer.

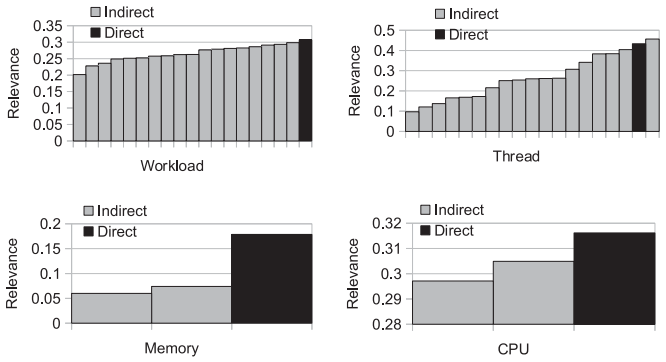


Fig. 4. The relevance of different primitives for the exampled service-instance. *X*-axis plots different primitive dimensions; *y*-axis denotes their SU values.

easy to see that even if we incrementally select (add) the relevant primitives one at a time, the validity and usefulness of cumulative relevance and redundancy rely on the following assumption (in the next section, we will experimentally verify this assumption):

Assumption 1. *For QoS modeling in the cloud, the model accuracy, represented by error, is negative to the difference between cumulative relevance and redundancy if the cumulative relevance is bigger (i.e., the bigger the difference, the smaller error); or being positive to such difference if the cumulative redundancy is bigger (i.e., the bigger the difference, the bigger error).*

Indeed, if this assumption does not hold, it means that the cumulative SU values cannot correctly differentiate and quantify the effects of some relevant primitives to the model accuracy, and this will significantly mislead the selection process. That is to say, given two sets of selected relevant primitives A' and B'; A' should result in better accuracy than B' if the cumulative relevance of A' is greater than that of B' while the cumulative redundancy in A' is smaller than that in B'. However, when Assumption 1 does not hold, B' can actually result in better accuracy than A'. Consequently, the situation can mislead the selection process as it may eliminates some highly useful primitives that help to improve the model accuracy. In the following, we will analyze the effects of selected primitives w.r.t. model accuracy and verify Assumption 1 for QoS modeling in the cloud.

4.3 Relevance and Redundancy Analysis on Primitives Selection

To study the correlation of selected primitives to the accuracy for modeling QoS in the cloud, we have conducted several analysis on the relevance and redundancy of selected primitives by means of experiments (see Section 6 for the detailed setup). In particular, we have carefully analyzed the relevance between possible primitives and QoS from the experiments— we first select the relevant primitives and then we rank them based on their relevance to the QoS. We found that the only constant observation across many QoS attributes and service-instances is that for each feature dimension (i.e., thread, CPU, Memory and Workload), certain primitives are more relevant to the QoS than all or most of the others. As an example, Fig. 4 shows the relevance (measured by (3)) for Response Time of a service-instance for different feature dimensions, calculated by averaging the values from all 350 intervals in one run. We performed *Wilcoxon Signed-Rank* test (two-tailed) for all comparisons. The resulted p values are smaller than 0.05, which confirms the statistical significance of the results. We discovered that the more relevant primitives are the ones that can directly influence the corresponding service (dark bars), e.g., the thread of the service and CPU of the VM; on the other hand, the less relevant primitives are the ones that can only interfere the service and its QoS via contention (light bars), e.g., the thread of co-located service and CPU of co-hosted VM. Such observation indicates that the former is more important to the QoS than the latter as QoS interference can only occur when the contention is quite significant [13]. These facts motivate us to partition the possible relevant primitives spaces into two sub-spaces, namely *direct primitives space* and *indirect primitives space*. By leveraging the classifications in Rule 1, the former is defined by:

Rule 2. *A primitive belongs to the direct primitives space for modeling the QoS of S_{ab} if it is in group 1,2,3 or 4 from Rule 1.*

It is clear to see that the direct primitive space contains primitives that can directly influence the QoS, which means they tend to provide different aspects of information. On the other hand, the indirect primitives space contains information about the QoS interference. Consequently, the indirect primitive can be defined as:

Rule 3. *A primitive belongs to the indirect primitives space for modeling the QoS of S_{ab} if it is in group 5 or 6 in Rule 1.*

It is worth noting that the indirect primitives space should generally be larger than the direct primitives space as it is sensitive to the number of co-located service and co-hosted VMs, which can be expended largely in the cloud. It is possible that both direct and indirect primitives space have irrelevant primitives, which can be easily eliminated.

Next, to verify whether the Assumption 1 is valid for the case of QoS modeling in the cloud, we have conducted a set of analytical experiments to evaluate how the accuracy changes with respect to the changes of cumulative relevance and redundancy. In particular, while keeping the total number of primitives and services unchanged, we gradually add more relevant primitives as the selected inputs (from higher relevance to lower relevance) to the modeling process. For each set of selected primitives, the model accuracy and cumulative values are calculated by averaging the results from all 350 intervals in one run. We have used all the three learning algorithms (i.e., ANN, ARMAX and RT) and assessed the accuracy using SMAPE [31]. It has been shown that SMAPE is intuitive, stable and more resilient to outliers than the other metrics [32].

In summary of the experiments, we have obtained four major observations: (i) within the direct primitives space, Assumption 1 does not hold. This is due to the fact that the direct primitives space contains different underlying primitives that directly influence the QoS, hence they can usually provide different aspects of information about a QoS attribute, which cannot be correctly quantified by cumulative SU value. Surprisingly, we also found that (ii) for inter direct and indirect primitives space, Assumption 1 does not hold either; (iii) however, within the indirect primitives

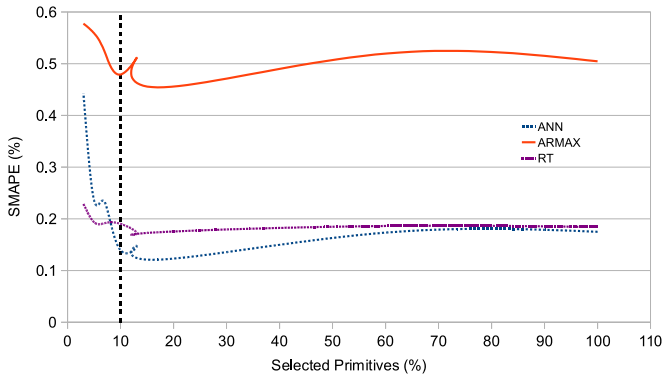


Fig. 5. The fluctuation on model accuracy as the number of selected primitives increase.

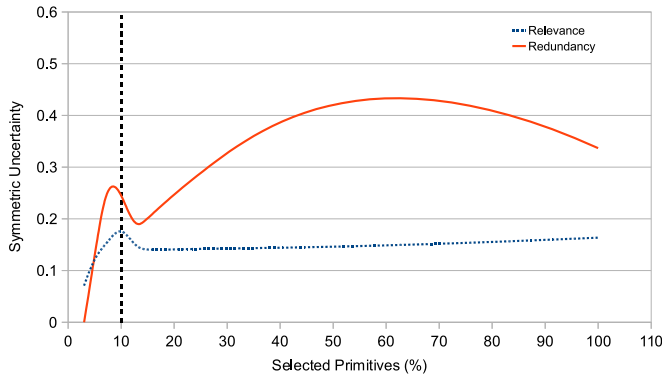


Fig. 6. The fluctuation on cumulative average of relevance and redundancy as the number of selected primitives increase.

space, Assumption 1 is valid. We believe that the reason for observations (ii) and (iii) is due to the fact that different direct primitives provide different aspects of information about the QoS and they influence the QoS directly. Whereas all the indirect ones can only do so via interference and contention; henceforth, they can only provide information on contention which can be regarded as one aspect of information that influences QoS. Obviously, this aspect of information is different to those in the direct primitive space. These observations also imply that the cumulative SU values can only quantify the effects of primitives to model accuracy, when they provide the same aspect of information. The final observation (iv) is that, although the overall relevance in direct primitives space is smaller than that of the indirect primitives space (as the former is smaller in size), the resulted model accuracy when using direct primitives is generally better than the use of indirect ones. This is a typical consequence of redundancy: the overall redundancy in the indirect primitives space tends to cause more negative effects on model accuracy than that of the direct one. Such observation means that even when redundancy is considered, the direct primitives can be more important than the indirect ones in the modeling. However, we observed that the best accuracy is achieved by the combination of direct and indirect primitives. This means consider proper information of QoS interference in the modeling can be quite beneficial for accuracy.

We now explain the process of analysis in details by referring an example to simplify the exposition. In particular, we report on the Response Time of a service-instance, but similar results have been observed on many other instances. To avoid noise caused by the irrelevant primitives, we have considered only relevant primitives in the analysis. Fig. 5 shows how the accuracy tends to change with the cumulative distribution of selected primitives in the modeling. Fig. 6 expresses the changes of the cumulative average of relevance (dashed blue line) and redundancy (solid red line) as the number of selected primitives increases. Similarly, Fig. 7 shows the changes of the cumulative total of relevance and redundancy with respect to the number of selected primitives. It is worth noting that, it can be hard to interpret the cumulative relevance and redundancy using cumulative total, as they are on significantly different scales, especially when the number of selected primitives increase. Therefore, we have normalized the data in the way that the scales of both values are in the range between 0 and 1.

We initiate the process by adding the direct primitives before the indirect ones as the former can be relatively smaller in size, which causing minimal noise when the number of primitives increases. In Figs. 5, 6, and 7, the trend between 0 and 10 percent of the x-axis shows the effects of adding direct primitives while the remaining shows the effect of adding indirect ones. From Fig. 6, we can see that the increase of cumulative redundancy tends to be larger than the increase of cumulative relevance, but they become close again as they reach the 10 percent. We obtained similar results from Fig. 7 for the cumulative total of relevance and redundancy. This means that, if Assumption 1 is true from 0 to 10 percent, then the error is expected to increase gradually and smoothly before it drops slightly toward 10 percent. Nevertheless, we observed rather contradictory results on the accuracy curve of Fig. 5— for all the three learning algorithms, the error drops almost linearly from 0 to 10 percent, which means that in the direct primitives space, Assumption 1 does not hold.

Next, we can see that similar result also occur at the initial stages when adding the indirect primitives, particularly between 10 and 20 percent of the x-axis. Precisely, both Figs. 6 and 7 indicate that from around 13 percent, the cumulative relevance increase almost linearly and the cumulative redundancy increase following a logarithmic behavior. This means that if Assumption 1 is true, the error is expected to become larger from 13 percent. This is contradicted with what is shown in Fig. 5— the error continues to drop till it reaches the best point at around 13 to 17 percent, and the accuracy stabilizes up to the 20 percent x-axis. Given that the number of

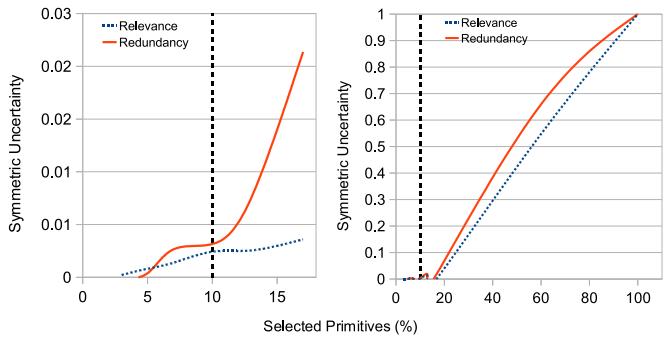


Fig. 7. The fluctuation on cumulative total of relevance and redundancy as the number of selected primitives increase.

primitives from both the direct and indirect primitives spaces is close (i.e., 10 percent of the total number of relevant primitives for each space), theses observations reveal that for the inter direct and indirect primitive spaces, Assumption 1 does not hold either. In addition, the accuracy trend implies that a combination of all direct primitives and some indirect ones yields better accuracy as it is important to consider interference in the modeling.

Finally at Fig. 6, we can see that from 20 percent and onwards, the cumulative relevance increases slightly and linearly whereas the cumulative redundancy tend to exhibit logarithmic and nonlinear behavior in its increase it increases from 20 percent and drops by 60 percent. Similar trend can be observed from Fig. 7— at around 60 percent, the increasing slope of cumulative redundancy becomes steeper towards the curve of cumulative relevance, which keeps increasing linearly. As a result, if Assumption 1 is true, then the error should become larger from 20 to 60 percent; while from 60 percent onward, the error should start to drop slightly and smoothly. This is almost what we can observe from Fig. 5 for the three learning algorithms. Since the effects of direct primitives becomes weaker (after 20 percent) when more indirect primitives are involved in the modeling, the results indicate that in the indirect primitives space, the Assumption 1 is indeed valid. Another observation is that using the direct primitives tends to lead to better accuracy than using of indirect primitives.

To summarize, we can conclude that Assumption 1 is true for intra-indirect primitives space. However, for inter-direct and indirect spaces, this assumption does not hold; for intra-indirect primitives space, this assumption is invalid either. We believe the reason being is that Assumption 1 can be easily violated when there are certain primitives providing different aspects of information to the QoS. This means a single-learner based technique (i.e., considering all primitives equally) tends to be insufficient for the primitives selection, because it can merely follow one of the three patterns below: (i) select relevant primitives from one sub-space (e.g., direct primitives space), which can lose highly relevant information from other sub-spaces in the modeling; (ii) select relevant primitives from the entire possible relevant primitives space and unavoidably introducing too much unwanted redundancy; or (iii) select useful primitives from the entire possible relevant primitives space, in which the cumulative relevance and redundancy will mislead the selection process. The observations also indicate that in the fixed and manual primitives selection, one should be extremely cautious to consider every possible combination of the primitives in the analysis, which makes the process extremely expensive and complicated. This is especially true in situations where potential QoS interference needs to be considered, in which case the number of possible primitives increases dramatically. Further, even when such process takes place, the validity of offine result cannot be guaranteed due to dynamic and uncertain nature of cloud.

All these facts urge the need for a self-adaptive and online primitives selection for modeling QoS in the cloud, which we address in this paper. Given the cumulative relevance and redundancy can mislead the selection when Assumption 1 does not hold and the fact that it is very difficult to efficiently handle the selection without cumulative representations, we have decided to avoid the misleading selection by partitioning the space. To better tackle the problem of relevance and redundancy in primitives selection, we intend to partition the primitives that provide different aspects of information to the QoS into sub-spaces, and select the useful primitives from each sub-space independently using cumulative relevance and redundancy.

4.4 The Hybrid Dual-Learners for Primitives Selection

To adaptively and dynamically select primitives as the model inputs online, we design a runtime filtering mechanism based on symmetric uncertainty, which has the advantage to assess the effects of selected primitives on model accuracy without actually training a model. Based on the analysis in Section 4.3, we use multi-learners in order to avoid the aforementioned issues caused by single-learner based technique. In particular, we partition the primitives that provide different aspects of information on the QoS into sub-spaces; this will result in $k+1$ partitions, where k is equal to the number of primitives in the direct primitives space; while the remaining one partition refers to the indirect primitive space. The objective is to select useful primitives from each sub-space independently using dedicated learners and then produce an ensemble results as the selected inputs for modeling. By doing so, we aim to produce a model with adequate model complexity and improved accuracy.

Inspired by [15], for each sub-space, we formalize a Maximal Relevance Minimal Redundancy (MRMR) learner using cumulative relevance and redundancy. This learner aims to continually select the primitives that maximize

$$max\ \Phi(S,Y),\quad s.t.\ U(X,Y) > 0, X \in S, \tag{8}$$

where X corresponds to the value vector of a primitive and Y to the value vector of QoS attribute. S denotes the associated sub-space; U is the function of symmetric uncertainty in (3). Mathematically, the objective function Φ can have four possible variations, depends on whether we use total or average to represent cumulative SU values; and whether we apply multiplicative or additive formulation to represent the difference between cumulative relevance and redundancy. Specifically, we obtain several variations of the objective function in (8)

Total and multiplicative:

$$\frac{\sum_{X\in S}^{n} U(X,Y)}{1+\sum_{X,X'\in S} U(X,X')} \tag{9}$$

Average and multiplicative:

$$\frac{\sum_{X\in S}^{n} U(X,Y)\times(n-1)}{n^2-n+2\times\sum_{X,X'\in S} U(X,X')} \tag{10}$$

Total and additive:

$$\sum_{X\in S}^{n} U(X,Y) - \sum_{X,X'\in S} U(X,X') \tag{11}$$

Average and additive:

$$\frac{\sum_{X\in S}^{n} U(X,Y)}{n} - \frac{2\times\sum_{X,X'\in S} U(X,X')}{n^2-n}, \tag{12}$$

where X' is the value vector of another primitive. n is the number of primitives, which has been already selected. It is clear to see that the constraint filters all the irrelevant primitives and this can be done easily. In this work, we apply incremental random search to optimize these functions for simplicity; however, it can be easily replaced by more sophisticated algorithms. In Section 6, we will experimentally compare these variations.

Given that the Assumption 1 does not hold indirect primitive space, we apply dedicated MRMR learner for each subspace independently. However, because there is only one primitive exist for each k sub-space, the objective here is equivalent to select the relevant primitives from all k subspaces, therefore these sub-spaces can be merged into one and the multiple MRMR learners can be simplified to a single Maximal Relevance (MR) learner, which aims to continually select the primitives that maximize

$$max\ \Psi(D,Y), \Psi = \sum_{X \in D}^{n} U(X,Y),\ s.t. U(X,Y) > 0, \quad (13)$$

where D denote the associated direct primitive space and all other notations are the same as (8). Again, the constraint filters all the irrelevant primitives.

It is worth noting that in case of selecting relevant primitives, certain forms of cumulative relevance are applicable as long as the cumulative relevance is positive to the number of selected primitives. This is because irrelevant primitives can benefit nothing but degrading accuracy [15], [30]. As shown in (13), the problem of selecting relevant primitives can be represented by maximizing the total SU value of relevance, subject to a constraint that the relevance of each selected primitive is greater than 0. This is because the cumulative relevance does increase positively with the increasing number of selected primitives. However, if we replace the cumulative relevance function to maximize average SU value, this can mislead the selection. In such case the cumulative relevance to the number of selected primitives can be negative. For example, if we have a two primitives set with 0.5 and 0.3 relevance each, they will have smaller cumulative average of relevance than that of a one primitive set with 0.5 relevance, but greater than that of a three primitives set with one 0.5 and two 0.3 relevance.

As for indirect primitives space, we use a MRMR learner for this sub-space, given that the Assumption 1 is true and the indirect primitive space tends to provide the same aspect of information to the QoS.

Eventually, we only need to partition the possible relevance primitives space into two sub-spaces, each of which employs learners with different primitives selection techniques (i.e., MR and MRMR learner). The final results are combined to form the selected useful primitives. We call this as the hybrid dual-learners technique. An algorithmic description of the technique is illustrated in Algorithm 1.

4.5 Comparing to State-of-the-Art Feature Selection Algorithms

Our symmetric uncertainty based hybrid dual-learners approach can be treated as similar to the Kolmogorov-Smirnov based and Information Value Ranking algorithms for feature selection. However, instead of following their specific algorithmic steps (e.g., select the most relevant feature first and then remove redundant based on that), we have formulated the problem as general optimization functions which, in turn, can be solved by many off-the-shelf optimization algorithms. Here, we have used randomized optimization for simplicity, but more sophisticated algorithms can be easily adopted. This will provide better flexibility for tackling primitives selection in cloud QoS modeling, which is an important, yet often ignored problem in the literature. In contrast to the other sensitivity analysis and tree-based importance analysis, our approach is lightweighted and intuitive, yet still effective without heavy human intervention for analyzing and tuning. Our approach has also been specifically tuned for QoS modeling in the cloud based on our observations regarding the effects of direct and indirect primitives in the modeling.

Algorithm 1. Hybrid Dual-Learners for Primitives Selection

Inputs:
given the value vector Y of a QoS attribute QoS_k^{ij}, the associated direct primitives space D and indirect primitives space ID
Declare:
C_{direct}—the collection of selected direct primitives
$C_{indirect}$—the collection of selected indirect primitives
Outputs:
the column entries of the selected primitives matrix SP_k^{ij}(t)
1: **start primitives selection**
2: $C_{direct} := \emptyset$, $C_{indirect} := \emptyset$,
3: $C_{direct} := argmax\ \ \Psi(D,Y)$ via (13)
4: $C_{indirect} := argmax\ \ \Phi(S,Y)$ via one from (9), (10), (11), and (12)
5: **end primitives selection**

Although one could argue that specific regularization algorithms (e.g., lasso and ridge), which shrinks the coefficients of each input according to its importance instead of the "cutting-off" them, might be effective and accurate, applying an universal feature selection can lead to the following benefits:

- Since the dimensionality of inputs are reduced, it helps to produce faster training and less computational cost for the learning algorithms. This is important for efficient and scalable online QoS modeling in the cloud.
- Our primitives selection method aims to "cutting-off" the useless and irrelevant inputs primitives (as identified by the symmetric uncertainty metric). This provides simpler and intuitive models, which in turn, helps the cloud engineers to identify the dependent QoS attributes, i.e., those that can be influenced by the same inputs. This is also useful when using the QoS models in the decision making process of cloud autoscaling.
- It is flexible to support many learning algorithms/models, as opposed to the fixed model in regularization driven algorithms (e.g., linear model or tree). This can help to improve the accuracy for those learning algorithms that lacks regularization (e.g., ANN).

TABLE 2
The SMAPE (%) of Learning Algorithms on Different QoS Attributes and Relative Standard Deviation

QoS Attribute (RSD)	ANN	ARMAX	RT
Response Time (4.197)	12.28	29.61	16.31
Throughput (0.663)	11.93	13.38	21.89
Reliability (0.012)	0.21	0.03	0.28
Availability (0.010)	0.37	0.03	0.43

5 QoS Function Construction in the Cloud

Recall from (1), once the primitives in $SP_k^{ij}(t)$ have been selected, our next goal is to determine *how* those primitives correlate with $QoS_k^{ij}(t)$ in the QoS function $f_k^{ij}(t)$.

Existing work has considered variety of learning algorithms for QoS function construction, ranging from simple linear model [33] to complex nonlinear ones [7]. These algorithms are self-adaptive and dynamic in nature thus they are capable to deal with dynamic and uncertain magnitude of primitives in the correlation. In this section and by means of experiments, we study the accuracy of the most widely used single learning algorithms (i.e., ANN, ARMAX and RT.) for QoS modeling in the cloud. In particular, we assess the accuracy of the learning algorithms over four different QoS attributes—Response Time, Throughput, Reliability and Availability (see Section 6.2 for their detailed definitions). Finally, we present a self-adaptive and online solution for QoS function construction, namely adaptive multi-learners, to address the issues discovered.

5.1 Suitability Analysis of Learning Algorithms on QoS Function Construction

For simplicity of exposition, we illustrate the results for a service-instance for the three learning algorithms over the four QoS attributes. We have used the variation in (7) for primitives selection. To better interpret the result with respect to different trends of QoS attributes, we apply Relative Standard Deviation (RSD) to measure the fluctuation of the QoS in a relative manner, calculated as: RSD = σ/μ, where σ is the standard deviation and μ is the mean of all measured QoS values. We can observe from Table 2 that the RSD value of the QoS attribute can be sorted by the following ascending order: Availability, Reliability, Throughput to Response Time; this means the trend of Response Time being the most fluctuated one. At the other extreme, the trend of Availability being the most stable one. As shown in Table 2, we can clearly see that the accuracy achieved by a learning algorithm differs significantly from case to case—ANN is the best for Response Time and Throughput while the ARMAX is the best for Reliability and Availability. In particular, the results of ARMAX reduces the error to 0.03 percent for Reliability and Availability; while ANN tends to be significantly better than ARMAX for Response Time and RT for Throughput. Beside, even though RT perform the worst for most of the cases, it can still largely reduce the error in contrast to ARMAX at the case of Response Time.

An interesting discovery from Table 2 is that, if we interpret the accuracy in conjunction to the RSD of different QoS attributes, we can see that the ANN tends to perform better than ARMAX on Throughput and Response Time where the fluctuations of trend are relatively large; and this improvement tends to increase from Throughput to Response when the trend becomes more fluctuated. On the other hand, ARMAX tends to produce better accuracy than ANN on Reliability and Availability, where the fluctuations of trend are relatively small; and this improvement tends to increase from Reliability to Availability when the trend becomes more stable. These observations reveal that nonlinear model like ANN can better handle the dynamic and uncertain magnitude of primitives in the correlation leading to better accuracy when the fluctuation of the QoS increases, whereas the linear ARMAX produces less error as such fluctuation decreases.

All these experimental results suggest that the learning algorithms perform quite differently depending on the QoS fluctuation trends and primitives combination; henceforth, we cannot reach a conclusion that a certain algorithm is generally the best learning algorithm for QoS modeling in the cloud. This indicates that given the generality of the proposed QoS model, the single learner is limited as it is difficult to determine which learning algorithm to use without expensive and intensive analysis. In addition, even when such process is performed, the offine analysis can still become invalid at runtime. Therefore, it is desirable to build a self-adaptive mechanism that not only able to adaptively model the magnitude of selected primitives to the QoS, but also dynamically select the suitable algorithm based on the runtime trend of a QoS attribute.

5.2 The Adaptive Multi-Learners for QoS Function Construction

Given the fact that most machine learning algorithms are self-adaptive and dynamic in nature, the crucial challenge here is how to adaptively determine the best learning algorithm for QoS function construction. To this end, we propose an adaptive multi-learners technique for updating QoS function $f_k^{ij}(t)$ on the fly and predicting the QoS values, as mentioned in Fig. 3. The technique has two main processes, namely training and prediction. At the training process, we simultaneously apply different learners to train the same QoS function, but each of the learners uses different learning algorithm to build a model. At the prediction process, we evaluate these learning algorithms by comparing the resulted models within the bucket on the fly; the model of the best learning algorithm is used to predict QoS.

One of the most critical design decisions is to determine the evaluation function that compares the models produced by candidate learners. The basic method would be based on global mean error of all historical samples. However, as shown by Kundu et al. [7], given a set of primitive values as inputs, the most accurate model using these inputs might not be the one that has the best global error. This is because the accuracy of a model can be sensitive to the local construct of given input values, including the variation of possible combination, scale and granularity, etc. As a result, our evaluation function aims to compare both the local error of a given inputs set produced by a model and the global error of the said model. In this work, we have used SMAPE for measuring the error, but other metrics can be used easily.

An algorithmic description of the training process has been shown in Algorithm 2. At the training process, as the

collected online data increases, we continually train two QoS models for each learner (lines 2-5): (i) A *main-model* that uses 100 percent of the collected online data; (ii) A *sub-model*, which is trained based on 70 percent of the total collected data. The sub-model is used to test local and global error for its *main-model* of a learner. In particular, it tests the QoS prediction error on the remaining 30 percent testing data— the split of training and testing data follows standard machine learning approach for testing generalization errors. These generalization errors and their corresponding samples (i.e., the observed values of all selected primitives and QoS at each interval) within the testing data serve as the *local error patterns* of the *main-model*. Finally, the *main-model*, *sub-model* and *local error patterns* are put in a bucket.

Algorithm 2. Training Process in Adaptive Multi-Learners

Inputs:
given the column entries of SP_k^{ij}(t) from Algorithm 1 and a set of candidate learning algorithms
Declare:
$\langle M_{main}, M_{sub}, L\rangle$ —a vector of *main-model*, *sub-model* and the corresponding *local error pattern* bucket—a collection of model vectors
Outputs:
a bucket of model vectors for a QoS attribute QoS_k^{ij}
1: **for each** candidate learning algorithm **simultaneously do**
2: *find* the optimal number of row entries, i.e., the value of q in (2), for SP_k^{ij}(t) if it has not been predefined for this learning algorithm
3: *train* main-model M_{main} and sub-model M_{sub} based on the required inputs defend by SP_k^{ij}(t)
4: *test* the sub-model for building local error pattern L
5: bucket := bucket $\cup \langle M_{main}, M_{sub}, L\rangle$
6: **end for**

An algorithmic description of the prediction process has been shown in Algorithm 3. The prediction process is triggered when there is need to perform prediction. In particular, the best *main-model* in the bucket is used as the final model to predict QoS. To calculate the local error of a main-model, we leverage the prediction error of its *sub-model* for each sample within the testing data, as recorded in the local error patterns (lines 3-9). When given a set of inputs (i.e., new values of the selected primitives) for predicting QoS, the local error of a main-model is determined by extrapolating the similarity between the given set of inputs and each sample from local error patterns; the error of the *most similar* sample is used as the local error (lines 4-7). To this end, we apply symmetric uncertainty based euclidean Distance to measure the similarity. As shown in (14), d is the distance of the given set of inputs against a sample in the local error patterns

$$d = \sqrt{\sum_{x \in X} (SU_x \times (p_x - p'_x)^2)}, \tag{14}$$

px and $p'x$ respectively denote the value of xth selected primitive in the given set of inputs and the value of the same primitive in a sample from local error patterns. SU_x is the symmetric uncertainty value between the xth primitive and the QoS attribute. The sample results in the smallest d is the one that we are seeking, then its corresponding error is used as the local error of the *main-model* (line 9).

Algorithm 3. Prediction Process in Adaptive Multi-Learners

Inputs:
given a set of inputs P and the bucket from Algorithm 2
Declare:
S—the current sample
$S_{selected}$—the most similar sample to P
d—the distance between P and the current sample
$d_{smallest}$—the smallest distance between P and a sample
E_{local}—the local error of the current *main-model*
E_{global}—the global error of the current *main-model*
E—the final error of the current *main-model*
$E_{smallest}$—the smallest final error of a *main-model*
$M_{selected}$—the selected *main-model* for prediction
Outputs:
the predicted QoS value of QoS_k^{ij}
1: **start prediction**
2: **for each** $\langle M_{main}, M_{sub}, L\rangle$ in the bucket of QoS_k^{ij}(t) **do**
3: **for each** sample S in the *local error pattern* L of M_{sub} **do**
4: *calculate* distance d between P and S using (14)
5: **if** $d_{smallest} > d$ **then**
6: $d_{smallest} := d$, $S_{selected} := S$
7: **end if**
8: **end for**
9: *get* the error of $S_{selected}$ as the local error E_{local} of M_{main}
10: *get* the global error E_{global} of M_{main}
11: *evaluate* final error E of M_{main} using (15)
12: **if** $E_{smallest} > E$ **then**
13: $E_{smallest} := E$, $M_{selected} := M_{main}$
14: **end if**
15: **end for**
16: *predict(P)* using the selected *main-model* $M_{selected}$
17: **end prediction**

On the other hand, the global error of a *main-model* is the mean errors of all samples within the 30 percent testing data produced by its *sub-model* (line 10). Finally, the evaluation function selects the best *main-model* for a given set of inputs by examining on both the local and global error of all *main-models* in the bucket, as formally depicted in (15) (lines 11-14)

$$E^i = \alpha \times E^i_{local} + \beta \times E^i_{global}, \tag{15}$$

where E^i, E^i_{local} and E^i_{global} denote the final, local and global error of the ith *main-model* respectively. α and β are two heuristics expressing the relative importance of local and global errors. We have set the initial value of α and β as 0.1, which means the local and global error are equally important from the beginning. The selected *main-model* and its learning algorithm for a given inputs is the one that has the smallest (line 16). To capture the right weight of local and global errors, α and β are updated via 16 when new data is collected

$$\alpha = \alpha + \Delta\alpha, \quad \beta = \beta + \Delta\beta$$
$$s.t. \begin{cases} \Delta\alpha = e_{\alpha=0,\beta=1} - e_{\alpha=1,\beta=0} & \text{if} \quad e_{\alpha=1,\beta=0} < e_{\alpha=0,\beta=1} \\ \Delta\beta = e_{\alpha=1,\beta=0} - e_{\alpha=0,\beta=1} & \text{if} \quad e_{\alpha=1,\beta=0} > e_{\alpha=0,\beta=1}. \end{cases} \tag{16}$$

Specifically, $e_{\alpha=1,\beta=0}$ is the prediction error of new data produced by the selected *main-model* when $\alpha = 1$ and $\beta = 0$.

Similarly, $e_{\alpha=0,\beta=1}$ is the error produced by the selected *main-model* when $\alpha = 0$ and $\beta = 1$. In this way, the error that is more useful in the selection will gradually gain more importance. This updating process has been illustrated in Algorithm 4.

Algorithm 4. Update α and β in the Evaluation Function

Inputs:
newly measured values vector P_{sample} and QoS value y for a QoS attribute of selected primitives QoS_k^{ij}
1: **start update when newly data is available**
2: *predict* (P_{sample}) using Algorithm 3 when $\alpha = 1, \beta = 0$
3: *predict* (P_{sample}) using Algorithm 3 when $\alpha = 0, \beta = 1$
4: *calculate* the errors of the values from step 2 and 3 against y
5: *calculate* $\Delta\alpha$ and $\Delta\beta$ using (16)
6: *update* α and β using (16)
7: **end update**

As mentioned in Section 5.1, we employ three different learning algorithms (i.e., ARMAX, ANN and RT) in the adaptive multi-learners. Our technique is flexible as new algorithms can be added or old algorithms can be removed/substituted if needed. The online training of these learning algorithms follows standard procedure; in the following, we briefly explain the applied learning algorithms.

Auto-Regressive Moving Average with eXogenous inputs—ARMAX [25] models the correlation between QoS and primitives as a linear relation and it captures the time-series information into the model. In this work, we train the ARMAX using linear Least Mean Square (LMS) approach [34]; and the q in (2) is determined using hill-climbing algorithm that starts with $q = 1$, then automatically increase the number of row entries one by one during training till it reaches good accuracy.

Artificial Neural Network—ANN [24] is a powerful supervised learning algorithm, which is capable for modeling complex nonlinear correlations. This is achieved by weighting the inputs and transforming them using activation function to produce the output. In this work, we use three layers and Sigmoid function in the network as this setup tends to relief the issue of local minima. ANN is trained using a well-known technique, namely the Resilient backPROPagation (RPROP) [35]. We found that use $q = 1$ (i.e., no time series information) can produce the best result; and the right number of hidden neurons is determined using hill-climbing algorithm during training till the accuracy cannot be further improved.

Regression Tree—RT [26] is a learning algorithm that maps the relation of primitives and QoS into a tree-like structure, in which leaves represent class labels and branches express conjunctions of features to reach these labels. Training is completed via Classification and Regression Trees (CART) technique [36] and we found that use $q = 1$ (i.e., no time series information) can produce the optimal results.

5.3 Selected Model versus Ensemble Model

One could argue that an ensemble method may yield better accuracy than dynamic selected model. However, a single ensemble method has been also shown to be quite sensitive to the variance of performance of the candidate learning algorithms, that is, its performance can drop significantly when more candidate learning algorithms are used [37]; while dynamic selected model tends to be resilient to this issue. Perhaps, instead of dynamically selecting from a set of models, dynamically selecting one from a set of ensembles may be a more promising way to the problem. However, this will introduce extra overhead for the online QoS modeling as the possible number of ensembles increases dramatically w.r.t. the number of candidate learning algorithms. We plan to systematically compare (single and multiple) ensemble method and the dynamic selected model for cloud QoS modeling in our future work.

6 Experiments and Evaluations

To evaluate our modeling approach, we experimentally benchmark our results against other single-learner and manual techniques. Specifically, the primary intention of the experiments is to validate the approach against the following criteria:

- *Accuracy:* By comparing with various other state-of-the-art modeling approaches, we intend to examine whether the hybrid dual-learners and the adaptive multi-learners can achieve better accuracy.
- *Stability:* We intend to assess the stability of the accuracy achieved by our approach under different scenarios, i.e., different QoS attributes and learning algorithms, in contrast to the other competitors.
- *Sensitivity of accuracy to online data size:* We examine the sensitivity of accuracy of the proposed approach to the available online data size. The purpose is to evaluate how quick the model accuracy changes with respect to the increase in data size.
- *Overhead:* We intend to evaluate the overhead of our approach in terms of the latency in the modeling, for both the primitives selection phase and the QoS function construction phase.

In addition to the assessment of accuracy under different QoS attributes and/or learning algorithms using SMAPE, we also intend to examine the overall accuracy and stability for all the considered scenarios. However, given the assumption that the scenarios are equally important, simply calculate the average or sum of all SMAPE can mislead the results. This is because different QoS attributes produce different scale of the prediction error, e.g., the error for predicting Throughput tends to be much larger than that for Reliability; therefore a technique/learning algorithm that performs better for Throughput will more likely to dominate the overall results. To cope with this issue, we use the summation of normalized SMAPE to illustrate the overall accuracy of a competitor, as shown below

$$Overall\ Accuracy = 100 \times \sum_{i}^{n} \frac{e_i}{e_{i,mean}}, \tag{17}$$

whereby e_i is the SMAPE of a competitor for the ith QoS attribute and/or learning algorithm and $e_{i,mean}$ is the mean SMAPE of all competitors under such scenario; n is the total number of QoS attribute and/or learning algorithm. In this way, the errors under each scenario are formatted into the same scale where smaller value indicates better overall

TABLE 3
The Examined QoS Attributes and Primitives

QoS and Primitives		Description
Output	Response Time (ms)	The average leaped time between a service-instance receives and replies a request.
	Throughput (req/min)	The average rate of completed requests.
	Reliability (%)	The percentage of requests that being completed less than a threshold. (30 ms)
	Availability (%)	The percentage of time that the average response time above a threshold. (60 ms)
CP input	CPU (%)	Observed average CPU utilization of a VM.
	Memory (MB)	Observed average Memory utilization of a VM.
	Thread (no. of req)	Observed maximum concurrent threads of a service-instance. (a modified control knob of Tomcat's *maxThread*-property)
EP input	Workload (req/min)	Observed average request rate of a service-instance.

accuracy. Similarly, we assess the stability of a competitor via the summation of normalized distance to the best competitor under each scenario, formally calculated by

$$Stability = 100 \times \sum_{i}^{n} \frac{e_i - e_{i,best}}{e_{i,worst} - e_{i,best}}, \tag{18}$$

where $e_{i,best}$ and $e_{i,worst}$ are the SMAPE produced by the best and worst competitor respectively, under the ith QoS attribute and/or learning algorithm. The remaining notations are the same as (17). Again, smaller value indicates better stability across different scenarios.

6.1 Experiments Setup

We conducted experiments on private cloud using a cluster of PMs, each of which has Intel i7 2.8 GHz Quad Cores and 4 GB RAM. The PMs use Xen v3.0.3 [29] as the hypervisor and the modeling process is running on Dom0. To eliminate the interference caused by modeling, we allocated one CPU core and 1.2 GB RAM to Dom0, which tends to be sufficient. Our approach is implemented based on Encog [38] and Apache Mathematics [39] using Java JDK 1.6. To simulate QoS interference caused by the VMs while not exhausting resources, we run three co-hosted VMs on each PM; the remaining resources are evenly allocated to the co-hosted VMs. All VMs run linux kernel v2.6.16.29.

Our experiments leverage RUBiS [27], which is a cloud-based application consists of 26 co-located software services using the *eBay.com* model. For simplicity, we have used three RUBiS snapshots, each of which consists of a two-tiers (i.e., application and database tiers) based RUBiS application; the three RUBiS snapshots differ in terms of the database volume size ranging from to 5 GB data. Each RUBiS snapshot is deployed with a software stack including Tomcat v6.0.28 and MySQL v3.23.58 on each co-hosted VM of a master PM; and we have implemented sensors deployed on each service-instance and VM for sending the online data to *Data Collector*. For each RUBiS snapshot on the master PM, the application tier is replicated to all other servant PMs in the cloud; these replicas are connected to the database on the master PM for handling any database related requests. Finally, each of the three RUBiS snapshots and its replicas are linked to its dedicated load balancer. Three client emulators are used and they apply read/write pattern to generate requests for each load balancer. Here, we have considered two types of read/write pattern: a read-intensive pattern where read to write ratio is around 9:1; and a write-intensive one, i.e., read to write ratio is 1:1.

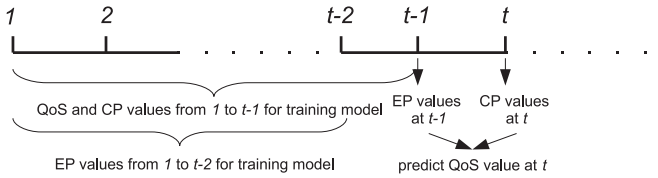


Fig. 8. The timing in evaluation of accuracy.

To simulate a realistic workload within the capacity of our testbed, we vary the number of clients proportionally according to the FIFA98 workload [28], which is compressed in the way that the fluctuation of a day in the trend corresponds to 200s in our case. This setup can generate up to 400 parallel requests, hence the compression is realistic and large enough to simulate QoS interference in cloud.

6.2 The QoS Attributes, Primitives and Evaluation Procedure

The concrete QoS attributes and primitives depend on scenarios. For the simplicity of exposition, we have selected commonly used QoS attributes and primitives in the evaluation, but it is worth noting that our approach is not limited to these dimensions. As listed in Table 3, these QoS attributes and primitives are per-service except for CPU and memory as they are shared on a VM. For each service-instance running on a VM of the master PM, a QoS model can at most has:

- four direct primitives—CPU, memory, thread and workload of the corresponding service-instance and VM.
- 54 indirect primitives—2 (thread and workload) $\times$ 25 (co-located service-instances) $+$ 4 (CPU and memory of another two co-hosted VMs).

This combination gives us a maximum of 58 possible relevant primitives for each service-instance.

At runtime, we examine the accuracy of one interval ahead prediction for each experiment run: by the end of interval t, the QoS models are trained based on historical data up to $t-1$ (up to $t-2$ for environmental primitives), and then we use the observed primitives values at t (at $t-1$ for environmental primitives) to predict the QoS value at t, which is finally used to compared with the actual QoS value via SMAPE. The timing regarding how the series of data are used in our QoS modeling approach has been illustrated in Fig. 8. The sampling and modeling intervals are both

TABLE 4
The Compared Primitives Selection Techniques

	Primitive Space for MR	Primitive Space for MRMR	Description
HYBRID-V1	four direct primitives	54 indirect primitives	using (11) for MR and (7) for MRMR.
HYBRID-V2	four direct primitives	54 indirect primitives	using (11) for MR and (8) for MRMR.
HYBRID-V3	four direct primitives	54 indirect primitives	using (11) for MR and (9) for MRMR.
HYBRID-V4	four direct primitives	54 indirect primitives	using (11) for MR and (10) for MRMR.
HYBRID	four direct primitives	54 indirect primitives	using (11) for MR and (7) for MRMR.
SINGLE-MR	all 58 primitives	N/A	using (11) for MR.
SINGLE-MRMR	N/A	all 58 primitives	using (7) for MRMR.
MANUAL	N/A	N/A	fixed and offline selection that statically uses certain primitives (CPU and memory in our case) as inputs, e.g., [7], [33]—we modified the model from per-VM to per-service.
SINGLE-MR-DIRECT	four direct primitives	N/A	using (11) for MR and consider direct primitives only.

120 secs with the total of 500 intervals, where the first 150 intervals use a static and stable workload trend aiming at providing some essential data for the modeling; whereas the rear 350 intervals follow the FIFA98 trend. This setup can generate one new sample per interval for updating the model. For all accuracy related experiments, we examine the SMAPE for the rear 350 out of 500 intervals in one experiment run; we calculate the mean accuracy of all service-instances on one VM of the master PM and the reported results are computed by averaging 10 runs. We have performed *Wilcoxon Signed-Rank* test (two-tailed) for all comparisons. The resulted p values are smaller than 0.05, which confirms the statistical significance of the results.

6.3 Accuracy of Hybrid Dual-Learners for Primitives Selection

To assess the effectiveness of our hybrid dual-learners technique for primitives selection w.r.t. accuracy, stability and model complexity. To start with, we first compare the four variations of our hybrid dual-learners technique, as shown in Table 4. We report the results by following the evaluation procedure described in Section 6.2. Specifically, we apply three widely used learning algorithms (i.e., ANN, ARMAX and RT) for QoS function construction on all the QoS attributes.

From Table 5, we can see that for both workload patterns, HYBRID-V1 tends to produce the best accuracy overall, but it has marginal difference to HYBRID-V3 on the write-intensive pattern. As for stability, it is clear that the HYBRID-V1 achieves the best results. We observed that all four variations produce the same model complexity. Table 5 also show the detailed accuracy results under each of the 12 scenarios. For both workload patterns, HYBRID-V1 produces the best results for most of the cases on Response Time and Throughput; whereas for other two QoS attributes, the best variation tends to be different.

Next, we use HYBRID-V1 (we refer to as HYBRID for simplicity), as the representative of our hybrid dual-learner technique, to compare against various other self-adaptive and online selection techniques that are categorized as single-learner based (i.e., SINGLE-MR, SINGLE-MRMR and SINGLE-MR-DIRECT); and the manual selection technique, denoted as MANUAL, that has been widely used in existing static and semi-dynamic QoS modeling approaches (e.g., [7], [33]). Their detailed explanations can be found in Table 4.

From Table 6, for the write-intensive workload, we can see that the overall accuracy of all self-adaptive and online techniques is better than that of the manual one, except for the SINGLE-MR. This is because SINGLE-MR focuses on

TABLE 5
Comparing Variations of Hybrid Dual-Learners for Primitives Selection

			Write-intensive workload				Read-intensive workload			
			HYBRID-V1	HYBRID-V2	HYBRID-V3	HYBRID-V4	HYBRID-V1	HYBRID-V2	HYBRID-V3	HYBRID-V4
SMAPE (%)	Response Time	ANN	**12.28**	12.8	12.81	12.48	**13.51**	15.44	15.14	15.35
		ARMAX	**29.61**	30.56	29.84	36.59	44.85	45.62	**44.36**	45.68
		RT	**16.31**	18.1	16.39	16.37	**17.42**	21.19	20.26	21.58
	Throughput	ANN	**11.93**	12.29	12.67	13.59	**13.75**	15.73	15.84	15.55
		ARMAX	**13.35**	13.55	14.02	15.12	**15.02**	17.91	17.99	17.9
		RT	21.89	21.8	**21.14**	24.2	**22.07**	24.74	25.87	26.22
	Reliability	ANN	0.21	0.21	0.21	**0.2**	**0.32**	0.42	0.44	0.44
		ARMAX	**0.03**	**0.03**	**0.03**	**0.03**	**0.03**	0.04	0.05	0.04
		RT	0.28	**0.26**	0.29	0.27	0.38	**0.3**	0.43	0.33
	Availability	ANN	0.37	**0.36**	**0.36**	0.43	**0.61**	0.69	0.7	0.7
		ARMAX	0.03	**0.02**	0.03	0.03	**0.05**	0.06	0.06	0.06
		RT	0.43	0.45	**0.41**	0.55	0.68	0.64	**0.63**	0.65
Overall Accuracy (%)			**1170.2**	1180.99	1170.66	1278.15	**1083.95**	1214.5	1264.87	1236.69
Stability (%)			**321.7**	466.83	357.4	787.52	**198.02**	884.78	940.97	996.03
Complexity (number of inputs)			6 to 8	6 to 8	6 to 8	6 to 8	5 to 8	5 to 8	5 to 8	5 to 8

(The best is highlighted).

TABLE 6
Comparing Hybrid Dual-Learners with Single Learner for Primitives Selection

			Write-intensive workload					Read-intensive workload				
			HYBRID	SINGLE-MR	SINGLE-MRMR	MANUAL	SINGLE-MR-DIRECT	HYBRID	SINGLE-MR	SINGLE-MRMR	MANUAL	SINGLE-MR-DIRECT
SMAPE (%)	Response Time	ANN	**12.28**	16.12	13.03	17.8	12.92	**13.51**	15.9	21.84	30.73	15.49
		ARMAX	**29.61**	37.56	29.77	33.81	32.36	**44.85**	51.44	56.47	53.46	48.52
		RT	16.31	19.74	**15.4**	19.25	17.91	**17.42**	20.21	20.01	21.09	20.73
	Throughput	ANN	**11.93**	13.45	16.82	14.26	12.55	**13.75**	16.59	30.18	18.56	15.75
		ARMAX	**13.35**	15.29	17.53	14.17	13.6	**15.02**	17.04	17.31	17.87	17.66
		RT	21.89	23.52	23.17	**19.88**	19.94	**22.07**	24.4	30.79	25.81	24.51
	Reliability	ANN	0.21	0.55	0.35	**0.16**	0.17	**0.32**	1.2	0.45	0.36	0.55
		ARMAX	0.03	0.05	**0.02**	**0.02**	**0.02**	**0.03**	0.04	0.05	0.06	0.06
		RT	0.28	0.29	**0.24**	0.35	0.37	0.38	0.59	0.57	**0.37**	0.45
	Availability	ANN	0.37	0.39	**0.34**	0.36	0.36	**0.61**	0.78	0.65	0.68	0.72
		ARMAX	0.03	0.04	**0.02**	0.03	**0.02**	0.05	0.07	**0.03**	0.05	0.05
		RT	**0.43**	**0.43**	**0.43**	0.55	0.56	0.68	**0.64**	1.32	0.83	0.77
Overall Accuracy (%)			**1090.37**	1452.81	1132.38	1185.85	1138.59	**956.7**	1308.6	1315.1	1231.0	1188.9
Stability (%)			**246.73**	882.85	357.44	547.75	420.97	**60.3**	690.4	786.1	656.4	575.4
Complexity (number of inputs)			6 to 8	40 to 44	2 to 3	2 to 2	4 to 4	5 to 8	30 to 44	2 to 3	2 to 2	4 to 4

(The best is highlighted).

information relevance only and can introduce too much redundancy. This result indicates that even though the dynamics and uncertainties in QoS function construction can be captured by the considered learning algorithms, it is still important to properly handle the runtime dynamics and uncertainties in primitives selection. In particular, a carefully designed self-adaptive and online selection technique can lead to better accuracy than the manual selection technique; however an inappropriate one (i.e., the SINGLE-MR) can only make the accuracy worse off. Among the self-adaptive and online primitives selection techniques, we also observe that although the SINGLE-MRMR ignores the fact that Assumption 1 does not hold in some parts of the space and hence mislead the selection, it tends to produce better accuracy overall in contrast to that of SINGLE-MR and SINGLE-MR-DIRECT. This is because they have been affected by more serious issues: SINGLE-MR has too much redundancy while SINGLE-MR-DIRECT does not explicitly consider information about QoS interference. Finally, we can see that our HYBRID produces the best accuracy overall. Specifically, in contrast to those single-learner based techniques, HYBRID has better overall accuracy than that of SINGLE-MR-DIRECT because it considers extra information about interference in the modeling, which tends to be important for improving accuracy. In addition, it is also overall more accurate than SINGLE-MR and SINGLE-MRMR, because it is capable to select useful primitives based on both relevance and redundancy while still prevent misleading the selection process. This is achieved by partitioning the possible relevance primitives space.

As for the overall accuracy under read-intensive workload, SINGLE-MR and SINGLE-MRMR are less accurate than SINGLE-MR-DIRECT; they are even much worse than the manual technique. This implies that the rich redundancy and the misleading selection cause more serious issues as when compared to write-intensive workload pattern. For our HYBRID technique, we can note that it again achieves the best accuracy overall, which is a consistent result on both workload patterns.

The stability of the techniques is also illustrated in Table 6; it is easy to see that our HYBRID technique produces the best result for both workload patterns, meaning that it is the most robust one under different scenarios. As for complexity shown in the same Table, the HYBRID can be slightly more complex than the others, except for SINGLE-MR. However, the benefits here is that the model's overall accuracy is better and more stable than others with respect to the QoS attributes and the learning algorithms.

Table 6 also show the detailed accuracy results for each of the 12 scenarios. Again, we can see that for both workload patterns, the HYBRID produces the best results for most of the cases on Response Time and Throughput, which are highly fluctuate; but the best for Reliability and Availability tend to vary. This is because the Reliability and Availability trends tend to fluctuate less than that of Response Time and Throughput. Therefore, the sensitivity of certain learning algorithms to the number of inputs are amplified; this can easily lead to over-fitting when the model complexity increases, which will significantly influence the model accuracy. However for Reliability and Availability, the differences of accuracy between HYBRID and the best one ranges from 0.01 to 0.05 percent, which is marginal as when compared to the improvement that HYBRID offers.

According to all these results, we can conclude that although HYBRID does not constantly produce the best accuracy for every learning algorithms and QoS attributes, it tends to produce the best overall accuracy; it is also the most robust and stable technique in the presence of variability introduced by different learning algorithms and QoS trends. In particular, HYBRID provides better accuracy when QoS fluctuates, while leaving the model complexity adequate. It is also worth noting that having a self-adaptive and online primitives selection process promotes numerous other benefits, e.g., reduce the needs for complex human analysis and can be easily adapted to many learning algorithms etc.

6.4 Accuracy of Adaptive Multi-Learners for QoS Function Construction

To evaluate our adaptive multi-learners technique (denoted as ADAPTIVE) for QoS function construction, we follow the evaluation procedure described in Section 6.2. For different QoS attributes, we compare the accuracy and stability of ADAPTIVE with that of the other online learning

TABLE 7
Comparing Adaptive Multi-Learners with Single Learning Algorithms for QoS Function Construction

		Write-intensive workload				Read-intensive workload			
		ADAPTIVE	ANN	ARMAX	RT	ADAPTIVE	ANN	ARMAX	RT
SMAPE (%)	Response Time	13.72	**12.28**	29.61	16.31	13.82	**13.51**	44.85	17.42
	Throughput	12.72	**11.93**	13.35	21.89	14.16	**13.75**	15.02	22.07
	Reliability	**0.03**	0.21	**0.03**	0.28	**0.03**	0.32	**0.03**	0.38
	Availability	**0.03**	0.37	**0.03**	0.43	**0.05**	0.61	**0.05**	0.68
Overall Accuracy (%)		**192.8**	474.90	285.20	646.20	**179.01**	488.89	322.83	609.27
Stability (%)		**101.51**	156.17	114.32	323.25	**94.81**	171.75	115.26	312.48

(The best is highlighted).

algorithms that assume single learner (i.e., ANN, ARMAX and RT), which has been widely studied in existing semi-dynamic QoS modeling approaches e.g., [7], [33]. In all the cases, we have used HYBRID for primitives selection.

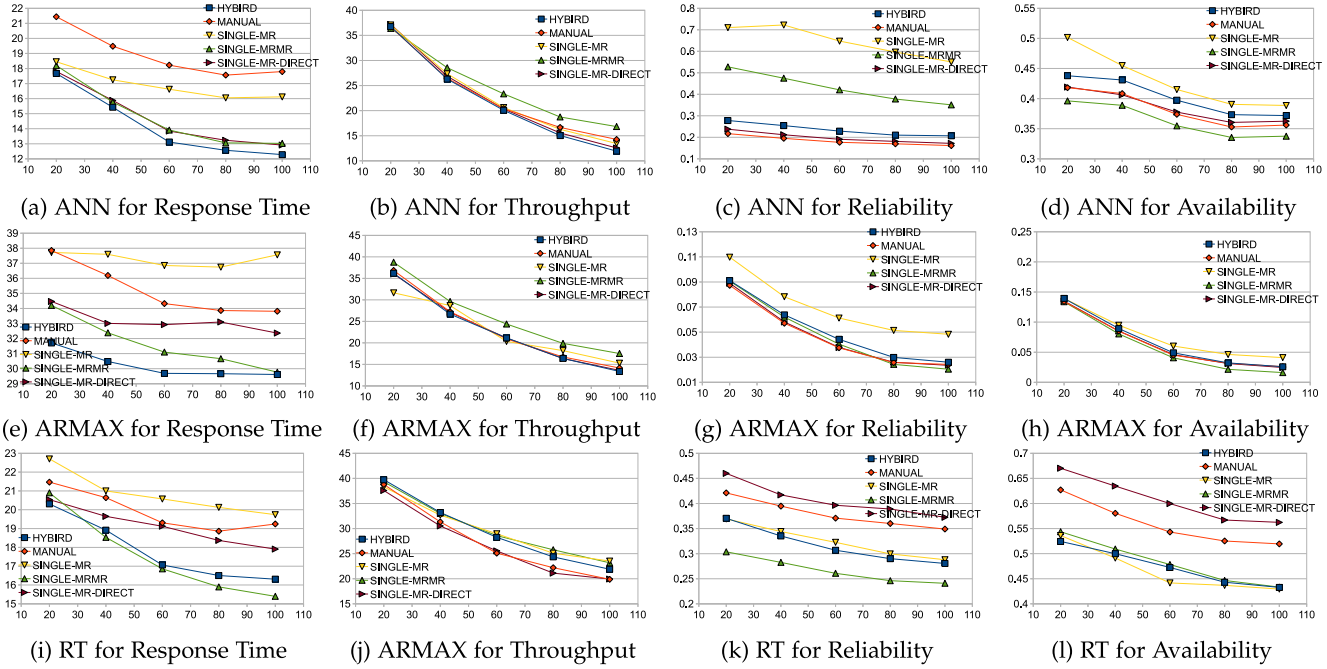
From Table 7, we can clearly see that ADAPTIVE produces the best accuracy overall for both workload patterns. It is also the most stable and robust against different QoS attributes. Detailed accuracy results for each scenario can be also found on Table 7. Here, we observe similar results on both workload patterns: for Response Time and Throughput, the ANN is the best learning algorithm in contrast to ARMAX and RT; We can see that ADAPTIVE is also much better than ARMAX and RT, but being slightly worse than ANN. These results indicate that although the ADAPTIVE might occasionally produce false positive/negative for selecting the best learning algorithm, it is still able to produce very closed accuracy to the best learning algorithm for a QoS attribute. In cases of Reliability and Availability, we can see that the ADAPTIVE is able to produce the same prediction error as the best learning algorithm, which is ARMAX. This result means that the ADAPTIVE successfully determines the best learning algorithm along the QoS trend.

In summary, we can note that although the algorithms behave differently depends on different QoS trends, our adaptive technique can still continuously select the suited one to predict QoS and result in good accuracy; it is also the most stable on different QoS trends. Moreover, our self-adaptive and online solution eliminates the need of heavy human intervention for identifying the suitable learning algorithm, reducing the errors caused by human analysis.

6.5 Sensitivity of Accuracy to Online Data Size

Next, we evaluate the sensitivity of accuracy to the online data size for our approach. This sensitivity expresses how quick accuracy changes as the available data samples increase. Instead of doing one-interval-ahead prediction, we sequentially split the data size of the entire 350 intervals into training data set and testing data set containing 70 and 30 percent of the original data respectively. The training data set is then further divided into different portions based on the order of time series: 20, 40, 60, 80, and 100 percent. These portions serve as the actual training data applied for building the QoS models which are, in turn, used to make prediction over the 30 percent testing data set. In the following, we report on the results for the write-intensive pattern over 10 runs. Similar observation has been registered for the read-intensive workload pattern.

Fig. 9 shows the sensitivity of accuracy to data size for the HYBRID and other single learner-based and manual

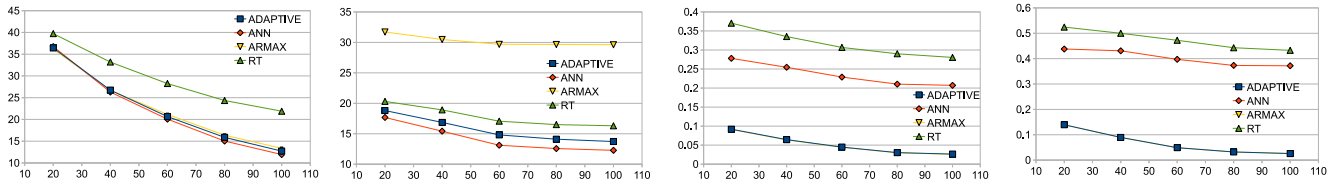


(a) ANN for Response Time (b) ANN for Throughput (c) ANN for Reliability (d) ANN for Availability

(e) ARMAX for Response Time (f) ARMAX for Throughput (g) ARMAX for Reliability (h) ARMAX for Availability

(i) RT for Response Time (j) ARMAX for Throughput (k) RT for Reliability (l) RT for Availability

Fig. 9. Sensitivity of model accuracy to online data size for each primitives selection technique. The *y*-axis denotes SMAPE (percent); *x*-axis expresses the online data size (percent).



(a) Response Time (b) Throughput (c) Reliability (d) Availability

Fig. 10. Sensitivity of model accuracy to online data size for each learning algorithm. The *y*-axis denotes SMAPE (percent); *x*-axis expresses the online data size (percent).

selection techniques. We note that all primitives selection techniques lead to better accuracy as the data size increases, given the fact that all selected primitives are more or less relevant to the QoS. In most of the cases, the sensitivity of model accuracy to data size has been similar for all the primitives selection techniques. In addition, the comparative accuracy under limited data do not differ much as to what had been reported in Section 6.3. However we found that in certain cases (e.g., Figs. 9a and 9e), particularly for fluctuated QoS trends, the accuracy produced by HYBRID clearly has the greatest sensitivity to data size; or being more sensitive than most of the other selection techniques. We also discovered that in these cases, HYBRID tends to produce better or similar accuracy in contrast to the other selection techniques, even when the data size is limited. These observations imply that, in contrast to the other approaches, HYBRID can still further improve the accuracy quicker as the data samples increase, while maintaining relatively less or similar error under limited data size.

Fig. 10 illustrates the sensitivity of accuracy to data size for the ADAPTIVE and other single learner-based learning algorithms. Again, all learning algorithms gradually improve on accuracy as the data size increase. The sensitivity of ADAPTIVE has been similar to most of the others for Response Time and Reliability (i.e., Figs. 10a and 10c). However, for Throughput and Availability (i.e., Figs. 10b and 10d), our ADAPTIVE and the best learning algorithms (i.e., ANN and ARMAX) tends to improve accuracy slightly quicker than the others while maintaining relatively less error under limited data size. We can also observe that, in contrast to the corresponding best single learning algorithm for each QoS attribute, the accuracy of our ADAPTIVE has the same or similar sensitivity to the online data size.

6.6 Efficiency

To assess the overhead of our approach, we compare the latency of HYBRID to other single learner-based techniques, which has been considered in the experiments for primitives selection; we also examine the latency of ADAPTIVE to that of ANN, ARMAX and RT for QoS function construction. Because the latency can be varied depends on the characteristics of the service and data size, we have used an instance of the service named *SearchItemsByCategory* as the example given that it exhibits the most fluctuated workload. The experiments are performed using the rear 10 out of 500 intervals and we report on the average results of all QoS attributes over 10 runs.

Fig. 11 shows the performance overhead for different primitives selection techniques. We can see that under both workload patterns, the HYBRID (0.68s and 0.65s) has relatively bigger overhead as when compared to SINGLE-MR and SINGLE-MR-DIRECT; but it is smaller to that of SINGLE-MRMR. We have observed that this is due to the majority of overhead is caused by the optimization process of (6), which is not part of the process in SINGLE-MR and SINGLE-MR-DIRECT. However, such extra overhead of HYBRID is generally acceptable as it is still less than 1 sec. For the case of QoS function construction, Fig. 12 illustrates the best and worst cases for all learning algorithms. In particular, for both patterns, ANN generally produces bigger overhead as when compared to ARMAX and RT. This is because the ANN is fundamentally more complex than the other two. For both the best and worst cases, the ADAPTIVE has relatively similar overhead to that of ANN; this is expected as the ADAPTIVE needs to wait for the completion of all simultaneously running learning algorithms before determine the best one to use. In conclusion, the overhead of our modeling approach is acceptable under the

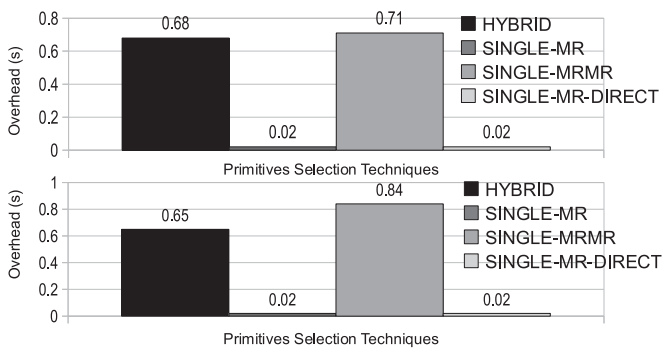


Fig. 11. Overhead for primitives selection on write-intensive (top) and read-intensive workload (bottom).

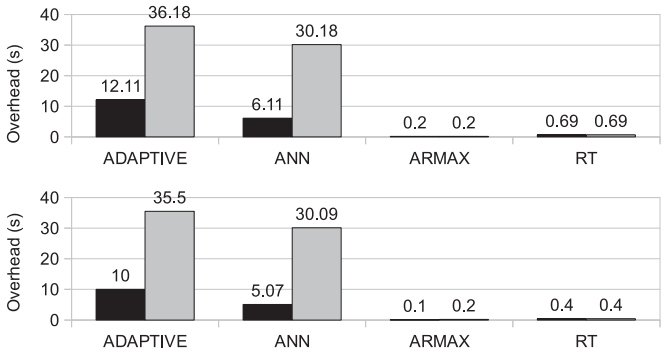


Fig. 12. Overhead for QoS function construction on write-intensive (top) and read-intensive workload (bottom).

sampling and modeling interval of 120 s, and thus it is efficient enough to be performed online.

6.7 Model Exploitations

As mentioned, the resulted QoS models can serve as powerful tools and foundation to achieve better self-adaptivity in cloud computing, benefiting both the cloud providers and consumers. Among others, the most significant exploitation and benefits are:

- The models allow one to achieve more concise and isolated decision making for elastic autoscaling, where the independent QoS attributes (i.e., those that do not sensitive to the same control primitives) can be reasoned about in separated local processes without affecting each others. This will yield less overhead but still guarantee the overall quality. We have quantitatively evaluated such benefit in our other work [40].
- The models make identification of the sources of contention and QoS interference easier. This is because the primitives, which are selected as the input of most QoS attributes, are likely to be the ones that cause serious contention and QoS interference. Consequently, they might need to be tuned more carefully than the others.
- The models create the foundation to reason about the effects of the different amount of scaling on different QoS attributes, the likely consequence of QoS interference and the possible trade-offs, i.e., answering the related "what-if" questions. This will, in turn, provide better governance and assurance of the autoscaling actions (e.g., vertical/horizontal scaling) that are subsequently performed. Interesting reader can refer to our other work for details on this topic [41], [42].

7 Threats to Validity

The main threats to validity of our approach are associated with its scalability, which can be discussed in two folds: the horizontal scalability w.r.t. the number of input dimensions; and the vertical scalability, which is concerned with the number of data samples associated with these inputs.

- *Horizontal Scalability:* the number of inputs dimensions can directly influence the overhead of symmetric uncertainty based optimization. However, one benefit of our formulation of the primitives selection problem is that it can be easily adopted with many optimization algorithms, including those that capable to handle high number of variables and those that provide approximated result under NP-hard problem. At the stage of the QoS function construction, our hybrid learners approach has ensured a lower number of inputs, which has proven to be effective and scaling the modeling process. As shown in the experiments, the learning algorithms has to deal with only six to eight out of 58 input primitives. This is only around 10 percent of the original set in the modeling process. Further, unlike many other non-cloud related problems (e.g., DNA analysis) where the number of features can be a million, the number of inputs dimensions in the cloud environment is likely to be much less, e.g., it might not exceed an hundred, depending on how many VMs and services are co-running and their functional dependencies. Thus, our approach has acceptable scalability in such a context.
- *Vertical Scalability:* In the primitives selection phase, the number of data samples can only influence the calculation of symmetric uncertainty. Given that we have used a cumulative relevance and redundancy representation, the overhead of such calculation is linear to the number of data samples. In addition, we have observed that the overhead is negligible in the primitive selection phase when the number of data samples increases. However, vertical scalability might suffer bottleneck at the QoS function construction phase. Therefore a *forgotten strategy* is desired when there is no need to take too much data into account. To achieve such goal, one could set a threshold to the maximum number of historical intervals to be recorded. Once such threshold is exceeded, the QoS function construction process can apply cross-validation to examine if dropping data from the oldest intervals would affect the model accuracy. For example, if the reduction in accuracy is less than 1 percent error, then such data can be removed.

8 Related Work

8.1 Analytical Modeling

Analytical models have been widely used for QoS modeling in the cloud, these models are built offline based on theoretical principles and assumptions. Among others, queuing theory (e.g., queuing networks and layered queuing networks) is one of the most popular technique used in existing work. In this approach, QoS is usually modeled as a mathematical function with respect to the CPU and the likely distribution of workload [3], [4], [43]. For example, [3] describe an approach to decompose QoS into hardware CPs, where a multi-station queuing network is used to analyze the correlation between demand and performance related QoS. Their work is not specific for cloud computing however. [4] is cloud specific and focus on analyzing QoS model for each tier of an application with respect to the resources via queuing network. Dependability models are another widely used techniques for modeling in the cloud. This approach focus on the modeling of stable states for QoS attributes. [44] utilize Stochastic Petri Net with interacting sub-models to create the correlation between availability and primitives in cloud (e.g., design parameters and mean-time-to-failure etc). [45] apply Markov model to correlate QoS with different deployment strategies, i.e., different configurations and resource provisions. Finally, black-box models are also popular, in which the QoS is modeled based on empirical knowledge or statistical data of history [19], [46], [47]. For instance, Emeakaroha et al. [46] propose a black-box framework to manage QoS in the cloud. Their QoS models are static expressions, which are formula constructed based on empirical knowledge. All of the aforementioned analytical approaches are

static, closed-form QoS models and they often require in-depth knowledge of the likely behaviors of the service that being modeled. Consequently, their effectiveness is restricted to the assumptions of system's internal operations; such static nature makes these approaches limited in coping with the dynamic and uncertain QoS sensitivity in cloud. In addition, the resulted models are coarse-grained and can be difficult to incorporate additional information, such as QoS interference and software control primitives.

8.2 Simulation Based Modeling

Various simulators exist for creating QoS models; here, the simulations are usually expensive and thus they are used in an offine manner. Being the basis of many other simulators, CloudSim [5] allows to simulate performance with respect to the resources usage in cloud. CDOSim [48] is an extension of CloudSim aiming to model the correlation of QoS to hardware provision of VMs and the costs. It relying on OMG model and reverse engineering techniques. Likewise, GridSim [6] is another simulator that models the relationship of QoS and environmental conditions, i.e., the occurrences of events. Similar to the analytical approaches, simulation based modeling is also static and restricted to the assumptions made in the simulators, e.g., distribution of workload, distribution of the expected QoS performance and architecture of the hardware infrastructure. In addition, it is difficult to simulate QoS interference as it is usually hard to assume its distribution.

8.3 Machine Learning Based Modeling

The increasing complexity of managing services in the cloud makes the modeling difficulty far beyond the capability of human analysis. To this end, recent works have been leveraging on the advances of machine learning algorithms, e.g., simple linear regression [12], [13], ARMAX [33], [49], ANN [7], [17], [18], nonlinear regression [50], RT [16] and change-point detection [51] etc. For example, [33] and [49] apply linear ARMAX regression online to express correlation between performance and primitives for VM-based applications. Similarly, [7] and [18] leverage offine model training of ANN while [17] rely on online ANN for QoS modeling in the cloud. Nevertheless, they do not intend to discuss dynamic and uncertain QoS sensitivity. Despite QoS interference being core to the problem of QoS modeling in the cloud, there has been very little attempts: feedback control is often applied with Multiple-Input and Multiple-Output (MIMO) model to handle the QoS interference for QoS modeling in cloud [12], [13], [14]. Notably, the research discussed in [13] focuses on linear MIMO modeling of performance and interference in the cloud. Zhu and Tung [14] also intend to model QoS interference using fuzzy rules and Support Vector Machine (SVM). These approaches consider VM-level interference whereas our approach takes dynamic service-level interference into account. Hybrid solutions are also exist: [52] adapt Kalman-flter with linear regression to model QoS and they cluster the resulted models. Unlike the others, which are targeting per VM/application models, [53] propose a hybrid, fine-grained performance modeling where linear AR is used to predict demand of primitives and Kalman-flter is applied to tune the actual model. However, all those approaches are semi-dynamic as the primitives selection has been manual and fixed, as a result, they require extensive human analysis and investigation. Often, these approaches ignore the importance of primitives selection and QoS interference (for both service level and VM level). In addition, they do not take software control primitives from the PaaS into account.

To cope with the issue of primitives selection, [9] describe an online primitives selection technique using the combination of wrapper and filter for modeling QoS in cloud; the models are application specific. However, as mentioned, the wrapper can introduce large overhead and it is highly dependent to the learning algorithm applied. In addition, they have ignored QoS interference. Similar attempt has been conducted by Bu, Rao and Xu [8], in which the QoS is modeled for each VM. They consider software control primitives and use Simplex Reduction to do dynamic primitives selection online, the QoS function construction is handled by reinforcement learning. Both [9] and [8] are regarded as single-learner based because they consider each primitive equally in the space. In contrast, our approach works on a finer model.

A single learning algorithm is usually applied for QoS function construction when modeling QoS in the cloud, which can be limited under certain QoS trends. Alternatively, [54] propose a way to predict the utilization of hardware control primitive using an ensemble solution where the results from different learning algorithms are combined in a weighted sum relation. However, their approach is highly sensitive to the similarity of candidate learners [37]. Our work on the other hand, dynamically select the best algorithm for predicting the correlation between QoS and its primitives.

9 Conclusion and Future Work

In this paper, we propose a self-adaptive and online approach for QoS modeling in the cloud. To tackle the dynamics and uncertainties related to QoS sensitivity and interference, we use hybrid dual-learners technique for primitives selection. We have presented a detailed study on how the relevance and redundancy of selected primitives influences the model accuracy, which drives our designs. On the other hand, we have showed that different learning algorithms perform significantly different depends on QoS attributes and their fluctuations. Therefore, we use an adaptive multi-learners technique for QoS function construction. In this way, we aim to dynamically select the best learning algorithms at runtime. The experiment results suggest that, in contrast to state-of-the-art QoS modelings, our approach produces better overall accuracy while having acceptable overhead; and it is more stable against the variability introduced by different scenarios. More importantly, the proposed approach eliminates the need for heavy human intervention, which can be complex and error-prone.

The implication of QoS modeling and its dynamic analysis to intelligent adaptation in the cloud are vast: the model can assist autonomic software agents in predicting causes of probable risks leading to QoS violations; reasoning about appropriate mitigation strategies and/or even planning for optimal QoS design and online adaptation strategies. Moreover, it can assist problems related to QoS self-management, self-adaptation, resource utilization and elastic autoscaling.

In future papers, we will report on novel applications benefiting from the proposed modeling approach. We hope that these insights can help to influence the agenda for more intelligent engineering in future cloud computing.

ACKNOWLEDGMENTS

This work was jointly supported by the EPSRC Grant (No. EP/J017515/1) on "DAASE: Dynamic Adaptive Automated Software Engineering", and the PhD scholarship from the School of Computer Science, University of Birmingham. The authors are grateful to Prof. Xin Yao for the constructive feedback on the preliminary version of this work.

REFERENCES

[1] Google app engine. [Online]. Available: http://code.google.com/appengine/
[2] Amazon elastic compute cloud. [Online]. Available: http://aws.amazon.com/ec2/
[3] Y. Chen, S. Iyer, X. Liu, D. Milojicic, and A. Sahai, "SLA decomposition: Translating service level objectives to system level thresholds," in *Proc. IEEE 4th Int. Conf. Autonomic Comput.*, 2007, p. 3.
[4] R. N. Calheiros, R. Ranjan, and R. Buyya, "Virtual machine provisioning based on analytical performance and QoS in cloud computing environments," in *Proc. IEEE Int. Conf. Parallel Process.*, 2011, pp. 295–304.
[5] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw. Practice Experience*, vol. 41, no. 1, pp. 23–50, Jan. 2011.
[6] R. Buyya and M. Murshed, "GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency Comput.: Practice Experience*, vol. 14, no. 13, pp. 1175–1220, 2002.
[7] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, and K. Dutta, "Modeling virtualized applications using machine learning techniques," in *Proc. 8th ACM SIGPLAN/SIGOPS Conf. Virtual Execution Environ.*, 2012, pp. 3–14.
[8] X. Bu, J. Rao, and C. Zhong Xu, "Coordinated self-configuration of virtual machines and appliances using a model-free learning approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 4, pp. 681–690, Apr. 2013.
[9] P. Xiong, C. Pu, X. Zhu, and R. Griffith, "vPerfGuard: An automated model-driven framework for application performance diagnosis in consolidated cloud environments," in *Proc. 4th ACM/SPEC Int. Conf. Perform. Eng.*, 2013, pp. 271–282.
[10] E. Ebrahimi, et al., "Parallel application memory scheduling," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2011, pp. 362–373.
[11] P. Radojkovic, et al., "Thread assignment of multithreaded network applications in multicore/multithreaded processors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 12, pp. 2513–2525, Dec. 2013.
[12] N. Gandhi, D. Tilbury, Y. Diao, J. Hellerstein, and S. Parekh, "MIMO control of an apache web server: Modeling and controller design," in *Proc. Amer. Control Conf.*, 2002, vol. 6, pp. 4922–4927.
[13] N. Ripal, K. Aman, and G. Alireza, "Q-clouds: Managing performance interference effects for QoS-aware clouds," in *Proc. 5th Eur. Conf. Comput. Syst.*, 2010, pp. 237–250.
[14] Q. Zhu and T. Tung, "A performance interference model for managing consolidated workloads in QoS-aware clouds," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, 2012, pp. 170–179.
[15] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 8, pp. 1226–1238, Aug. 2005.
[16] P. Xiong, Y. Chi, S. Zhu, H. J. Moon, C. Pu, and H. Hacigumus, "Intelligent management of virtualized resources for database systems in cloud environment," in *Proc. IEEE 27th Int. Conf. Data Eng.*, 2011, pp. 87–98.
[17] G. Kousiouris, D. Kyriazis, S. Gogouvitis, A. Menychtas, K. Konstanteli, and T. Varvarigou, "Translation of application-level terms to resource-level attributes across the cloud stack layers," in *Proc. IEEE Symp. Comput. Commun.*, Jun. 2011, pp. 153–160.
[18] S. Kundu, R. Rangaswami, K. Dutta, and M. Zhao, "Application performance modeling in a virtualized environment," in *Proc. IEEE 16th Int. Symp. High Perform. Comput. Archit.*, Jan. 2010, pp. 1–10.
[19] Y. Zhang, G. Huang, X. Liu, and H. Mei, "Integrating resource consumption and allocation for infrastructure resources on-demand," in *Proc. IEEE 3rd Int. Conf. Cloud Comput.*, Jul. 2010, pp. 75–82.
[20] J. Li, et al., "Profit-based experimental analysis of IaaS cloud performance: Impact of software resource allocation," in *Proc. IEEE 9th Int. Conf. Serv. Comput.*, 2012, pp. 344–351.
[21] T. Chen and R. Bahsoon, "Self-adaptive and sensitivity-aware QoS modeling for the cloud," in *Proc. 8th Int. Symp. Softw. Eng. Adaptive Self-Manag. Syst.*, 2013, pp. 43–52.
[22] T. Chen, R. Bahsoon, and X. Yao, "Online QoS modeling in the cloud: A hybrid and adaptive multi-learners approach," in *Proc. IEEE/ACM 7th Int. Conf. Utility Cloud Comput.*, 2014, pp. 327–336.
[23] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd ed. San Mateo, CA, USA: Morgan Kaufmann, 2005.
[24] W. S. Sarle, "Neural networks and statistical models," in *Proc. 9th Annu. SAS Users Group Int. Conf.*, 1994, pp. 1538–1550.
[25] G. E. P. Box and G. Jenkins, *Time Series Analysis, Forecasting and Control*. San Francisco, CA, USA: Holden-Day, 1990.
[26] L. Rokach and O. Maimon, *Data Mining with Decision Trees: Theory and Applications*. River Edge, NJ, USA: World Sci. Publishing Co., 2008.
[27] Rice University bidding systems. [Online]. Available: http://rubis.ow2.org/
[28] M. Arlitt and T. Jin, "A workload characterization study of the 1998 world cup web site," *Netw. Mag. Global Internetworking*, vol. 14, no. 3, pp. 30–37, May 2000.
[29] Xen: A virtual machine monitor. [Online]. Available: http://xen.xensource.com/
[30] G. H. John, R. Kohavi, and K. Pfleger, "Irrelevant features and the subset selection problem," in *Proc. 11th Int. Mach. Learn.*, 1994, pp. 121–129.
[31] B. E. Flores, "A pragmatic view of accuracy measurement in forecasting," *Omega*, vol. 14, no. 2, pp. 93–98, 1986.
[32] S. Makridakis and M. Hibon, *Evaluating Accuracy (or Error) Measures*. Fontainebleau, France: INSEAD, 1995.
[33] P. Padala, et al., "Automated control of multiple virtualized resources," in *Proc. 4th ACM Eur. Conf. Comput. Syst.*, 2009, pp. 13–26.
[34] B. Widrow and S. D. Stearns, *Adaptive Signal Processing*. Upper Saddle River, NJ, USA: Prentice-Hall, 1985.
[35] M. Riedmiller and H. Braun, "RPROP—A fast adaptive learning algorithm," *Int. Symp. Comput. Info. Sci. VII*, 1992, pp. 279–286.
[36] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Monterey, CA, USA: Wadsworth and Brooks, 1984.
[37] M. Hibon and T. Evgeniou, "To combine or not to combine: Selecting among forecasts and their combinations," *Int. J. Forecasting*, vol. 21, no. 1, pp. 15–24, 2005.
[38] Encog framework. [Online]. Available: http://www.heatonresearch.com/encog
[39] Apache mathematics library. [Online]. Available: http://commons.apache.org/math
[40] T. Chen and R. Bahsoon, "Symbiotic and sensitivity-aware architecture for globally-optimal benefit in self-adaptive cloud," in *Proc. 9th Int. Symp. Softw. Eng. Adaptive Self-Manag. Syst.*, 2014, pp. 85–94.
[41] T. Chen and R. Bahsoon, "Self-adaptive trade-off decision making for autoscaling cloud-based services," *IEEE Trans. Serv. Comput.*, 2015, doi: 10.1109/TSC.2015.2499770.
[42] T. Chen, R. Bahsoon, and G. Theodoropoulos, "Dynamic QoS optimization architecture for cloud-based DDDAS," *Procedia Comput. Sci.*, vol. 18, pp. 1881–1890, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050913005000
[43] J. Li, M. Woodside, J. Chinneck, and M. Litoiu, "CloudOpt: Multi-goal optimization of application deployments across a cloud," in *Proc. 7th Int. Conf. Netw. Serv. Manage.*, Oct. 2011, pp. 1–9.
[44] R. Ghosh, F. Longo, F. Frattini, S. Russo, and K. S. Trivedi, "Scalable analytics for IaaS cloud availability," *IEEE Trans. Cloud Comput.*, vol. 2, no. 1, pp. 57–70, Jan.-Jun. 2014.

[45] R. Jhawar and V. Piuri, "Fault tolerance management in IaaS clouds," in *Proc. IEEE 1st AESS Eur. Conf. Satellite Telecommun.*, Oct. 2012, pp. 1–6.
[46] V. Emeakaroha, I. Brandic, M. Maurer, and S. Dustdar, "Low level metrics to high level SLAs-LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments," in *Proc. Int. Conf. High Perform. Comput. Simul.*, Jun. 2010, pp. 48–54.
[47] I. Hwang and M. Pedram, "Portfolio theory-based resource assignment in a cloud computing system," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, Jun. 2012, pp. 582–589.
[48] S. Frey, F. Fittkau, and W. Hasselbring, "Search-based genetic optimization for deployment and reconfiguration of software in the cloud," in *Proc. Int. Conf. Softw. Eng.*, 2013, pp. 512–521.
[49] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," *IEEE Trans. Serv. Comput.*, vol. 5, no. 4, pp. 497–511, Oct.–Dec. 2012.
[50] R. Chiang and H. Huang, "TRACON: Interference-aware scheduling for data-intensive applications in virtualized environments," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, Nov. 2011, pp. 47:1–47:12.
[51] P. Bodík, R. Griffith, C. Sutton, A. Fox, M. Jordan, and D. Patterson, "Statistical machine learning makes automatic control practical for internet datacenters," in *Proc. Conf. Hot Topics Cloud Comput.*, 2009, Art. no. 12.
[52] H. Ghanbari, et al., "Tracking adaptive performance models using dynamic clustering of user classes," *SIGSOFT Softw. Eng. Notes*, vol. 36, no. 5, pp. 179–188, Sep. 2011.
[53] T. Zheng, M. Litoiu, and M. Woodside, "Integrated estimation and tracking of performance model parameters with autoregressive trends," in *Proc. 2nd ACM/SPEC Int. Conf. Perform. Eng.*, 2011, pp. 157–166.
[54] Y. Jiang, C.-S. Perng, T. Li, and R. Chang, "Self-adaptive cloud capacity planning," in *Proc. IEEE 9th Int. Conf. Serv. Comput.*, 2012, pp. 73–80.

Tao Chen received the PhD degree in computer science from the University of Birmingham, in 2016, for his research on self-aware and self-adaptive autoscaling system in cloud computing. His PhD degree was fully funded by the competitive scholarship for international students from the School of Computer Science, University of Birmingham, selecting only one international recipient each year. He is currently a research fellow in the School of Computer Science, University of Birmingham, United Kingdom. His research interests include performance/QoS modeling and tuning, self-adaptive software systems, search-based software engineering, cloud computing, services computing, and distributed computing. His work has been published in internationally renowned journals such as the *IEEE Transactions on Services Computing*, the *IEEE Computer*, and the *Information Sciences*. He has also published regularly in leading conferences of his filed, such as SEAMS, UCC, IEEE Cloud, and ICCS. He is a member of the IEEE.

Rami Bahsoon received the PhD degree in software engineering from University College London for his research on evaluating software architecture stability using real options and he attended London Business School for MBA-level studies in technology strategy and dynamics. He is a senior lecturer of software engineering (associate professor) and leads the software engineering for/in the Cloud Interest Group, University of Birmingham, Birmingham, United Kingdom. The group's research aims at developing architecture and frameworks to support and reason about dependable complex software systems, where the investigations span cloud computing architectures and their economics. He published extensively in the area of economics-driven software engineering, cloud software engineering, and utility computing and co-edited a book on *Software Architecture and Software Quality* and another on *Economics-Driven Software Architecture* (published by Elsevier). He is a member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.