The Journal of Systems and Software 159 (2020) 110428

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Run-time evaluation of architectures: A case study of diversification in IoT

Dalia Sobhy[a,b,*], Leandro Minku[a], Rami Bahsoon[a], Tao Chen[c], Rick Kazman[d,e]

[a] School of Computer Science, University of Birmingham, Birmingham, UK
[b] Computer Engineering, Arab Academy of Science and Technology and Maritime Transport, Alexandria, Egypt
[c] Department of Computer Science, Loughborough University, UK
[d] University of Hawaii, Honolulu, USA
[e] SEI/CMU, USA

ARTICLE INFO

Article history:
Received 1 April 2018
Revised 12 August 2019
Accepted 23 September 2019
Available online 24 September 2019

Keywords:
Run-time architecture evaluation
Runtime architecture evaluation
Software architectures for dynamic environments
Internet of things
IoT
Design diversity

ABSTRACT

Run-time properties of modern software system environments, such as Internet of Things (IoT), are a challenge for existing software architecture evaluation methods. Such systems are largely data-driven, characterized by their dynamism, unpredictability in operation, hyper-connectivity, and scale. Properties, such as performance, delayed delivery, and scalability, are acknowledged to pose great risk and are difficult to evaluate at design-time. Run-time evaluation could potentially be used to complement design-time evaluation, enabling significant deviations from the expected performance values to be captured. However, there are no systematic software architecture evaluation methods that intertwine and interleave design-time and run-time evaluation. This paper addresses this gap by proposing a novel run-time architecture evaluation method suited for systems that exhibit uncertainty and dynamism in their operation. Our method uses machine learning and cost-benefit analysis at run-time to continuously profile the architecture decisions made, to assess their added value. We demonstrate the applicability and effectiveness of this approach in the context of an IoT system architecture, where some architecture design decisions were diversified to meet Quality of Service (QoS) requirements. Our approach provides run-time assessment for these decisions which can inform deployment, refinement, and/or phasing-out decisions.

1. Introduction

Architectures of complex, scalable real-time systems are dynamic in nature and exhibit numerous uncertainties in their operation (Da Xu et al., 2014; Nahrstedt et al., 2016). This dynamism limits the effectiveness of design-time architecture evaluation approaches. In particular, Internet of Things (IoT) environments (Gubbi et al., 2013; Nahrstedt et al., 2016) are fundamentally different from classical ones for which most architecture evaluation methods, such as Cost-Benefit Analysis Trade-off Method (CBAM) (Kazman et al., 2001), Architecture Analysis Trade-off Method (ATAM) (Kazman et al., 2000), Architecture-level modifiability analysis (ALMA) (Bengtsson et al., 2004), were advanced. These systems are data-driven, characterized by their scale, hyper-connectivity, dynamism, and uncertainty in operation (with a constant stream of new devices, new services, and fluctuations in QoS provisions) (Issarny et al., 2016). Furthermore, IoT has constrained devices, which are mobile, variable in processing and computational power, and in some cases with limited network connectivity (Da Xu et al., 2014; Nahrstedt et al., 2016). In this context, the dynamic nature of IoT requires us to combine design-time evaluation with run-time when evaluating architecture design decisions. This is because design-time evaluation relies greatly on human experts, who can partially predict the fitness and the extent to which an architecture can cope with operational uncertainties, unanticipated usage scenarios, and emergent behaviors. Run-time evaluation of architectures can complement design-time evaluation methods to provide more informed assessment of design options and capture deviations from the design-time evaluation for technical and value potentials.

Intertwining and interleaving run-time evaluation with design-time has the potential to change ad hoc and “trial and error” practices for architecting complex, scalable, and dynamic software systems. Consider, for example the case where architects embed

* Corresponding author at: Computer Engineering, Arab Academy of Science and Technology and Maritime Transport, Alexandria, Egypt.
E-mail addresses: dms446@cs.bham.ac.uk, dalia.sobhi@aast.edu (D. Sobhy), L.L.Minku@cs.bham.ac.uk (L. Minku), r.bahsoon@cs.bham.ac.uk (R. Bahsoon), t.t.chen@lboro.ac.uk (T. Chen), kazman@hawaii.edu (R. Kazman).

https://doi.org/10.1016/j.jss.2019.110428
0164-1212/

Nomenclature

d_{ka}	Architecture decision for capability k implemented using component a
dao_i	i diversified architecture option
DAO	a set of diversified architecture options
$q_{dao_i}(t)$	Quality of a *dao* varying over time
$q'_{dao_i}(t)$	Normalized Quality of a *dao* varying over time
w_q	Weight of quality q
$B_{dao_i}(t)$	Benefit of a *dao* varying over time
$\mu_{dao_i}(t)$	Exponential Benefit of a *dao* varying over time
$c_{dao_i}(t)$	Cost of a *dao* varying over time
$c^{v}_{dao_i}(t)$	Costs of each variety v per *dao* (e.g. deployment cost, leasing cost, *etc*)
$\sigma^2_{dao_i}(t)$	Exponential Variance of a *dao* varying over time
$\sigma_{dao_i}(t)$	Exponential Standard Deviation of a *dao* varying over time
θ	Relative Importance of the past
α	Confidence level
$L_\lambda(dao_i, t)$	Loss Function shows how (un)desirable a *dao* is
$L'_\lambda(dao_i, t)$	Marginal loss of non-dominated *dao* varying over time
λ	A pre-defined parameter that controls the relative importance between $c_{dao_i}(t)$ and $\mu_{dao_i}(t)$
dao_{curr}	Current *dao*

design diversity (Avizienis and Kelly, 1984) into their solutions in an attempt to meet quality requirements under uncertainty and to mitigate risks and Service Level Agreement violations. Diversification (Avizienis and Kelly, 1984) encompasses design decisions and architecture tactics that can be used to adapt the system to unforeseen changes (Baudry et al., 2014b). For a given concern, architecture diversity "spices" the architecture with a variety of design decisions and strategies (e.g. the choice of data collection and processing strategies and tools), which can better cope with uncertainties at run-time. Diversified design decisions, whether planned or accidental (Baudry et al., 2014a), can be expensive; their behavior and value can be best evaluated at run-time. However, there are no systematic software architecture evaluation methods that intertwine design-time and run-time evaluation.

This paper addresses this gap by proposing a *novel run-time architecture evaluation method* suited for systems that exhibit uncertainty and dynamism in their operation. Our method uses machine learning and cost-benefit analysis at run-time to continuously profile architecture decisions for their added value, identify significant deviations from previously expected benefits, and automatically determine which architecture options have the optimal cost-benefit trade-off. As such, it can inform deployment, refinement and/or phasing out architectural decisions.

Despite the existence of various run-time approaches, such as self-adaptive systems (Esfahani and Malek, 2013) and models@run.time (Blair et al., 2009) that can benefit from run-time evaluation approach, there is no actual run-time architecture evaluation approach in the literature yet.

By run-time evaluation, we envision several potential scenarios of use to capture the dynamic behavior of the selected design decisions in relation to the quality attributes in question:

- Simulation can be a useful alternative to experimentation with real environments: IoT environments are often large scale: an architecture can embed large numbers of heterogeneous and diverse things, sensors and devices supporting some design decisions. As it would be prohibitively expensive to configure the architecture and to test these decisions exhaustively before deployment, the use of simulation can be useful for assisting the architect in what-if analysis prior to deployment, stress-testing the architecture with inputs that can go beyond the ones encountered in normal operation, abstract the analysis, and demonstrate the potential for scaling it.
- This approach can also work if run-time data of a given configuration is available. The architect will need to instrument the system with mechanisms for monitoring, logging, and profiling quality attributes of interest and design decisions supporting these attributes. This can be particularly useful for cases where the system is already deployed and further refinements are envisioned. Learning from the log of operation for example can be useful for profiling and analyzing the likely technical and value potentials of these decisions at run-time, whether diversified or not.
- A third scenario can be also possible, where the approach can be integrated in continuous development paradigms, such as DevOps, where run-time information from the operation side can provide feedback for development.

Each of the mentioned scenarios would require separate treatment and reporting to show how the approach can be applied. The scope of this paper is concerned with the case of using simulation to support the run-time analysis. Nevertheless, many of the observations can be applicable for the other cases. In particular, we adopt the *iFogSim* (Gupta et al., 2017) tool. *iFogSim* builds on Cloudsim (Calheiros et al., 2011); it provides the architect with the freedom of hierarchically composing fog devices, clouds, and data streams to simulate the technical and value potentials of selected decisions using normal usage and stress tests in relation to quality attributes of interest.

Our method makes the following contributions to the research literature:

- a run-time method for tracking the benefits of architecture design decisions using machine learning;
- a method inspired by multi-objective optimization to evaluate the cost-benefit trade-offs of architectural decisions at run-time.

Architecture diversification is a common practice, when architecting software for systems at scale, dynamism, and uncertainty in their operations. Our method investigates this phenomenon and formulates the problem of architecture diversity from run-time and economics-driven perspectives.

The remainder of this paper is structured as follows: Section 2 illustrates the necessary background to understand the approach. Section 3 then discusses the research questions which the approach aims to answer. Section 4 presents the proposed approach. Section 5 explains our case study, a motivating example in the context of IoT; it uses iFogSim tool. Section 6 reports on evaluation of the research questions and scalability of the approach, Section 7 presents further analysis of proposed approach. Section 8 discusses the application of our approach to analyse the usefulness of diversification. Section 9 provides discussion of the work and the threats to validity. Section 10 presents the related work. Section 11 concludes the work.

2. Background

In this section, we will provide the background necessary to understand the run-time evaluation approach.

2.1. Reinforcement learning

Reinforcement learning is an agent-oriented approach that learns using observed rewards, by mapping situations to actions

as an attempt to maximize a scalar reward signal (Sutton and Barto, 1998). Generally, it is a sequential decision-making process, where in every step, the agent chooses an action and the reward is computed. It aims at optimizing the rewards to get the best possible outcome. Further, there is no supervisor, hence the reward signal is the only metric for the determination of the right action to take. Moreover, reinforcement learning is a trial and error learning process, where agents estimate the value of actions with acceptable reward from the experiences of its environment. The major challenge of this type of learning is the trade-off between exploration and exploitation (Sutton and Barto, 1998). The former indicates the examination of non-optimal action, which may provide better reward in the future. The latter denotes implementing the optimal known decision to maximize the reward. For further elaboration, the exploitation process may result in missing an optimal decision, which may yield better cumulative future reward rather than the present best one. Whereas the cost of exploration may exceed its benefits in some cases. In Section 4.2, we will show how our approach could be understood from the context of reinforcement learning.

2.2. Time-decayed function

In Wang et al. (2015), exponential smoothing function was used as a time-decayed function to handle the challenges of online class imbalance learning. The proffered function is similar to reinforcement learning, which tracks the rewards resulting from actions via the occurrence probability (percentage) of examples belonging to a particular class. It is different from the traditional way of considering all observed examples equally, instead they are updated incrementally by using a time decay (forgetting) factor to emphasize the current status of data and weaken the effect of old data. In particular, the adoption of time-decay function is advantageous, due to the following (Holt, 2004; Kuncheva, 2004): (i) weakens the effect of old data; (ii) very east to compute; and (iii) minimum data is necessary. In this context, our approach leverages a time-decayed function to provide the architect with flexibility of tuning the relative importance of past versus recent observations, when valuing the benefit of diversified architecture options (further information on how time-decay function is used in our approach is illustrated in Section 4.3).

2.3. Change detection approaches

Generally, in machine learning, the notion of concept drift denotes a change in the underlying distribution of the data, which the approach is learning over time (Gama et al., 2004). Examples of application include making inferences based on financial data, energy demand and climate data analysis, web usage or sensor network monitoring, and so forth (Ditzler et al., 2015). Non-stationary and uncertain environments are challenged by their rapid changes (i.e. drifts). Therefore, change detection approaches are necessary to discover these concept drifts and hence perform the corresponding actions (Gama et al., 2014; Ditzler et al., 2015). In Ditzler et al. (2015), the change detection approaches are classified into the following families: Hypothesis Tests (HT), Change-Point Methods (CPM), Sequential Hypothesis Tests (SHT), and Change Detection Tests (CDT). The latter approaches determine variations in the underlying data through statistical methods (e.g. sample mean, variance, *etc*), but they differ on how data is processed (Ditzler et al., 2015).

HT and CPM operate on fixed-length observations, whereas SHT and CDT sequentially investigate the incoming observations. Therefore, HT and CPM are not suitable for applications operating in an online manner, due to the high complexity with respect to analyzing all observations at once. Though SHT is partially suitable for online observations, it has the following drawback: SHT examines the incoming observations until it has enough statistical confidence for decision-making (i.e. a "drift" or "no drift" is detected), but after the decision is made it stops processing the observations. This is a common pitfall in this type of sequential analysis, since the main objective is to continually collect information to ensure true change detection. The CDT attempted to overcome the prior limitations, since they are continuously tracking the observations (i.e. fully sequential manner), which in turn provide reduced computational complexity. To this end, our approach adopts the change detection test proposed by Gama et al. (2004), but for a different purpose than that of Gama et al. (2004).

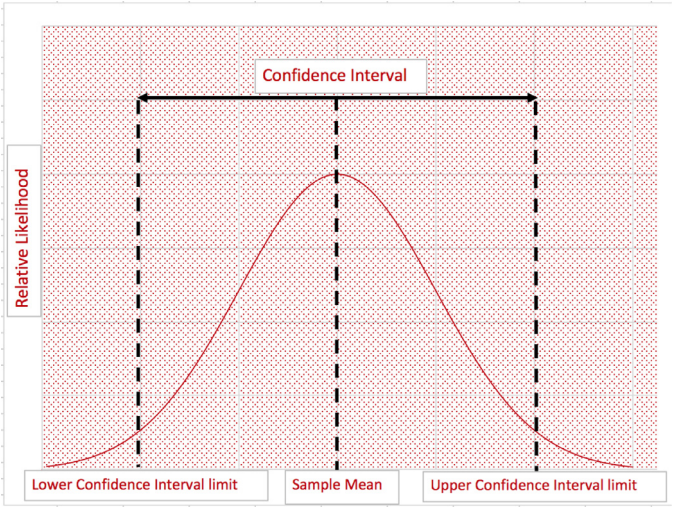


Fig. 1. A representation for the confidence interval statistical technique used by Gama et al. (2004) as the change detection test.

In Gama et al. (2004), the authors were interested in a supervised learning approach able to make accurate predictions in non-stationary environments. In this case, their approach first attempts to make a prediction, and then gains access to the true outcome from a supervisor. The average classification error of the predictions is then computed and monitored over time to detect changes in the probability distribution of the data. Changes are detected based on confidence intervals of the classification error average, as depicted in Fig. 1. In particular, if the classification error is within the boundaries of the confidence interval (i.e. the upper and lower limits of interval) then we are confident that there is no change, otherwise a change is detected. The upper and lower limits of interval are adjusted based on the level of confidence required (further explanation is found in Section 4.4). Our approach, on the other hand, can be understood in the framework of reinforcement learning, where there is no supervisor. We use the CDT to detect changes in the benefit of software architectures over time, rather than changes in the classification error as done in Gama et al. (2004).

2.4. Multi-objective optimization

When evaluating software architectures, some of the evaluation approaches appeal to multi-objective optimization (MOP) (Aleti et al., 2013). MOP is not trivial as the process involves optimizing multiple conflicting objectives. For this purpose, multi-objective evolutionary algorithms have been widely used, such as Non-dominated Sorting Genetic Algorithm-II (NSGA-II) (Deb et al., 2002). NSGA-II relies on the concept of non-dominance to decide which solutions are better. A non-dominated solution is a solution that is similar or better on all objectives, and strictly better

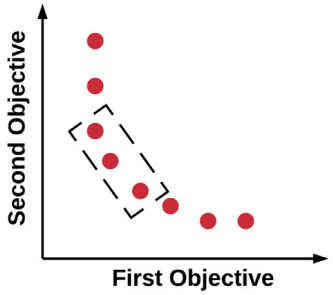


Fig. 2. An illustrative example of Pareto front and application of the knee point strategy. In the example, the knee points inside the dotted box have better chances to win (i.e. provide a more balanced trade-off between the two objectives).

in at least one objective (Rey Horn et al., 1994). Non-dominated solutions can thus be seen as better solutions than dominated ones. NSGA-II thus searches for the set of solutions that are non-dominated by any other solution, which can be referred to as Pareto front. Fig. 2 shows an illustrative example of Pareto front, where both the first and second objectives are to be maximized. NSGA-II then relies on humans to decide which of the solutions from the Pareto front to adopt for a given problem. However, this choice may not be easy.

Some MOP scenarios use a *knee point* strategy on their Pareto fronts (Fig. 2) to help choosing a solution. A knee point is "almost always the most preferred solution, since it requires an unfavourably large sacrifice in one objective to gain a small amount in the other objective" (Deb and Gupta, 2011). In particular, as in Fig. 2, moving in any direction out of dotted box may generate a small improvement in one objective, but with a large deterioration in other objective. Therefore, the knee point strategy promises to find the most balanced decision. Our approach adopted a knee point strategy inspired by NSGA-II for MOP. Further details on how this strategy is used in our approach are given in Section 4.5.

3. Research questions

Architecture diversification (Avizienis and Kelly, 1984; Baudry et al., 2014b) is commonly adopted for architecting dependable software through embedding more than one solution to realize a concern of interest. Our method investigates this phenomenon and formulates the problem of architecture diversity from run-time and economics-driven perspectives. In this context, we name each diversified architecture option *dao*, which is composed of several architecture decisions. Consider, for example a streaming application, where the architect could design different diversified architecture options to deal with uncertainties. In this application, a *dao* could comprise different fixed sensors (i.e. things) to collect streaming data and then process data to the cloud. Another *dao* could gather data instead from mobile sensors. We provide further description of *dao* in Sections 4.1 and 5. Deciding on which diversified architecture options to implement is not straightforward due to the uncertainties related to the dynamicity and the unbounded scalability of the things in composition. Our approach thus leverages design-time and run-time knowledge to embrace uncertainties. Based on design-time knowledge, the architects can decide on the options to be implemented. We use CBAM and options theory (Cox et al., 1979) to evaluate different architecture options potentials at design-time (Sobhy et al., 2016), where the architecture options providing high option value are considered for diversification of the IoT Case study (Section 5). However, as the environment is dynamic, value potentials can fluctuate at run-time.

This leads to the following questions, which our paper aims to answer:

RQ1 How to evaluate the benefit of each dao over time? A key requirement for determining the added value of a *dao* is through tracking its benefit over time based on its quality attributes (Q). In non-dynamic environments, tracking the benefit based on a simple average of its value at each time t could be sufficient. However, in dynamic environments, simple average may take a long time to reflect changes (Wang et al., 2015) in benefit. A method to enable tracking the current benefit over time is necessary.

RQ2 How can run-time evaluation determine changes in dao's value over time and inform subsequent decisions? To properly support decision-making, a run-time architecture evaluation approach should be able to identify when changes in the benefit of a *dao* are truly significant. A decision to change the software architecture based on an insignificant change in benefit would lead to an unstable system. Moreover, when a significant change is detected, run-time evaluation should be able to identify which *dao* provides a better trade-off between benefit and cost. Therefore, a method to detect both significant changes and balanced trade-offs is desired.

To answer these questions, we propose a run-time evaluation approach inspired by self-adaptive systems. The approach is able to profile situations where options can be more effective and provide continuous updates on their value potentials. Specifically, to answer *RQ1*, our approach adopts an *exponential time-decay function* inspired by reinforcement learning (Wang et al., 2015). This function enables us to track the current benefit of a *dao* by weakening the effect of old data (i.e. emphasize on the recent versus past observations). To answer *RQ2*, our proposed approach adopts *change detection tests* to check whether the benefit of the *dao* currently being used is getting significantly worse (Gama et al., 2004). If it is significantly worse, a method inspired by the multi-objective optimization literature (Branke et al., 2004; Deb et al., 2002) is adopted to identify the *dao* with the optimal trade-off between cost and benefit. Based on the profiling of options over time, it is also possible to determine which ones are not fit for the purpose, and hence could be phased out.

In addition, our approach aims to improve the evaluation of software architectures which could be better evaluated at run-time. The evaluation framework is generic enough to be applied to various architectures realizations, including (but not limited to) model-driven architectures, software product line architectures, cloud-based architectures, self-adaptive systems, *etc*. As one of the evaluation procedures, our approach evaluates which *dao* provides the best trade-off between cost and benefit. We analyse the effectiveness of such evaluation in Section 6 by checking what would happen if one was to adopt a run-time selection system (called *Informed-Selection System*) that simply selects the *dao* evaluated as the best one based on our architecture evaluation approach. If our approach's evaluation is successful, such run-time selection approach should lead to better trade-offs between cost and benefit than other selection approaches that ignore the evaluation provided by our approach.

The steps of the approach are summarized in Fig. 3. Here, the design-time evaluation is the one proposed in Sobhy et al. (2016), which uses options theory (i.e. an economics-driven approach) (Hull, 2006) to evaluate the diversified architectural options and shortlist the initial diversified architecture options for run-time evaluation. The run-time evaluation is our proposed approach, which is discussed in Sections 4.3–4.5.

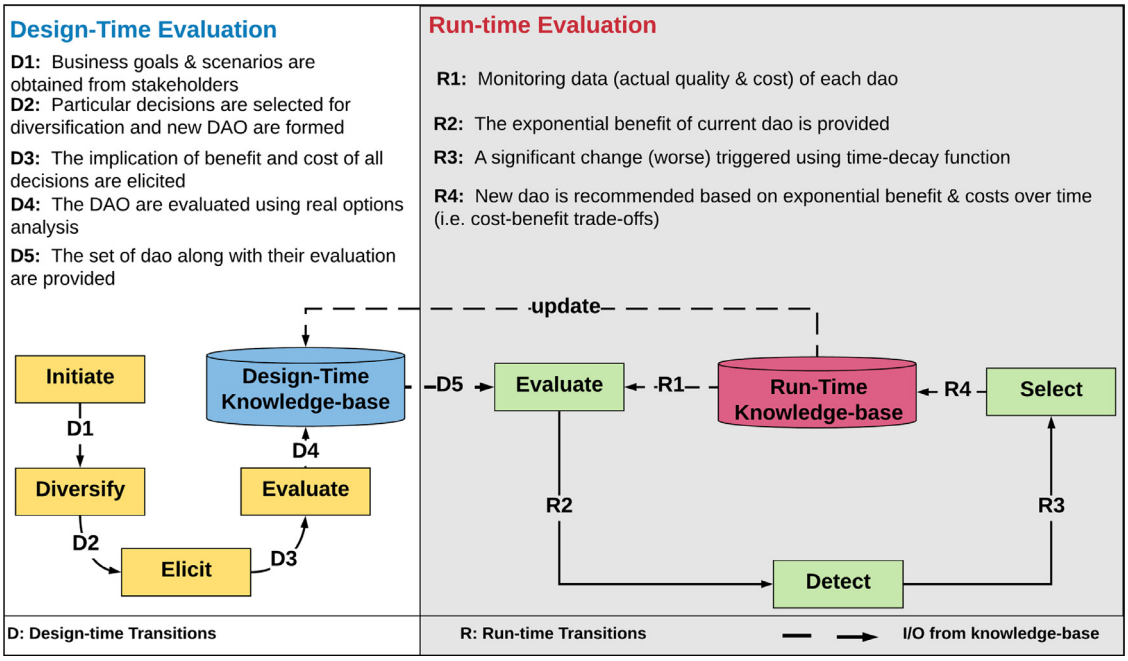


Fig. 3. Steps of the approach, where the design-time evaluation (Sobhy et al., 2016) forms the initial design decisions and run-time evaluation complements it.

4. Proposed approach

This section explains our proposed approach. Sections 4.3 explains how our approach addresses RQ1. Sections 4.4 and 4.5 explain how our approach addresses RQ2.

4.1. Diversified architecture options (DAO)

Design diversity (Avizienis and Kelly, 1984; Hawthorne and Perry, 2004; Avižienis et al., 2004; Baudry et al., 2014b; Sobhy et al., 2016) is used to design for dependability under uncertainty: the greater the uncertainty, the more diversity the architects may need to apply to improve performance. We denote a software architecture which embeds diversity as diversified architecture option *dao* and the set of *dao* as *DAO*. A *dao* implements a set of diversified decisions to meet some quality goals and trade-offs. Consider a set of architecture decisions D, where a decision $d_{ka} \in D$; k denotes a particular capability, including connectivity, data collection, data management, *etc*; and a indicates the software architecture component and connection that implements this capability k. For example, in an IoT system, architecture decisions for the capability of data collection d_1 could be performed either through fixed d_{11}, mobile d_{12}, or fixed+mobile sensors d_{13}. Another example is data processing could be performed either in cloud d_{21}, or fog+cloud d_{22}. Therefore, dao_i could collect data from fixed and mobile sensors (d_{13}) and processes it in cloud-fog (d_{23}). Other examples of *DAO* are depicted in Section 5 through Table 2. In the IoT system case, the diversity in each *dao* can refer to using fixed and/or mobile fog devices for data collection capability; using different cloud providers and heterogeneous fog devices for data processing capability.

4.2. The proposed approach in the context reinforcement learning

Our approach can be understood in the framework of Reinforcement Learning as follows. The actions are the suggested *DAO*. The reward is the benefit at a given timestep, observed by taking the action. At each timestep, the approach suggests an action, and the reward of this action is computed. Learning occurs by keeping track of the time-decayed benefit average, which is used to decide which action to suggest through a greedy deterministic policy, based on calculating the knee point that indicates the *dao* with the best trade-off between cost and benefit.

However, from the context of software architecture evaluation, we need to do the following: (1) perform reinforcement learning in an online way where bad actions chosen during, e.g., early stages of the learning process, can have serious inappropriate consequences to the architecture evaluation; (2) consider multiple goals, rather than a single one; and (3) avoid instability due to the frequent switching between actions aiming to maximize the reward. Therefore, we cannot adopt existing reinforcement learning approaches out of the box.

To deal with (1), we adopt simulators and/or monitor the diversified architecture options in parallel. We also then employ a pre-defined rule to decide which action to take (i.e. which *dao* to suggest) based on the modelled rewards. This rule designed to minimize the chances of making poor architecture recommendations. To handle (2), we get inspiration from the MOP literature (Branke et al., 2004). In our case, the suggestion of which *dao* to adopt is performed in an innovative way, based on non-dominated solutions (a key concept adopted by many MOP algorithms), knee point strategy, and the marginal loss (Section 4.5). To manage (3), we make use of change detection tests to avoid instability by changing the current suggested action only if the benefit of the currently adopted *dao* decays significantly.

In a nut shell, our approach performs *optimization guided by machine learning strategies (i.e. strategies derived from reinforcement learning)*, where a decay function is adopted that continually learns and updates the aggregated benefit of monitored quality values forming the exponential benefit. It then selects the balanced (knee) diversified architecture options when needed based on the learned exponential benefits.

4.3. Evaluating the diversified architecture options

The goal of the evaluation is to *determine what are the diversified architecture options, the quality attributes of interest and how they will react over time*. This step is divided into further sub-steps as explained below and summarized in Algorithm 1.

ALGORITHM 1: Evaluate().

Input: diversified architecture options *DAO*, number of *DAO* ($|DAO|$), quality $q_{dao_i}(t)$, number of quality attributes $|Q|$, weight of each quality attribute w_q, relative importance of the past θ

Output: exponential benefit $\mu_{dao_i}(t)$, standard deviation $\sigma_{dao_i}(t)$, cost $c_{dao_i}(t)$

{Initialization:} $q = 1 : |Q|$

1 **for** *i = 1 to* $|DAO|$ **do**

2 Compute quality $q_{dao_i}(t)$ based on Table 1

{For negative quality attribute (i.e. the lower the better, e.g. Response Time):}

3 **if** $q_{dao_i}(t)$ *is a negative quality &* $q^{max} - q^{min} \neq 0$ **then**

4 $q'_{dao_i}(t) = \frac{q^{max} - q_{dao_i}(t)}{q^{max} - q^{min}}$

{For positive quality attribute (i.e. the higher the better, e.g. Throughput):}

5 **else if** $q_{dao_i}(t)$ *is a positive quality &* $q^{max} - q^{min} \neq 0$ **then**

6 $q'_{dao_i}(t) = \frac{q_{dao_i}(t) - q^{min}}{q^{max} - q^{min}}$

7 **else**

8 $q'_{dao_i}(t) = 1$

{Check constraints' violation:}

9 **if** *any* $q'_{dao_i}(t)$ *violates constraints* **then**

10 $B_{dao_i}(t) = 0$

else

11 Compute benefit $B_{dao_i}(t) = \sum_{q \in Q} w_q * q'_{dao_i}(t)$

end

12 Quantify cost $c_{dao_i}(t) = \sum c^v_{dao_i}(t)$

13 Compute exponential benefit $\mu_{dao_i}(t) = \theta \mu_{dao_i}(t-1) + (1-\theta) B_{dao_i}(t)$

14 Determine variance $\sigma^2_{dao_i}(t) = \theta \sigma^2_{dao_i}(t-1) + (1-\theta) * (B_{dao_i}(t) - \mu_{dao_i}(t))^2$

15 Determine standard deviation $\sigma_{dao_i}(t) = \sqrt{\sigma^2_{dao_i}(t)}$

end

1. *Identifying the diversified architecture options and quality attributes of interest:* This step is inspired by the formulation of CBAM (Kazman et al., 2001; Nord et al., 2003), except for the fact that the benefit and cost of options vary over time. It also explicitly considers that diversified architecture options are composed of architecture decisions, which are defined by the architects (Fig. 4). In particular, the following must be specified:
   - The set K of capabilities selected for diversification. Examples of capabilities are data collection, connectivity, processing, routing topology, and data management.
   - The set D of architecture decisions. An architecture decision $d_{ka} \in D$ specifies an architecture component a to implement a given capability $k \in K$.
   - Set of diversified architecture options *DAO*, where $|DAO|$ represents the number of *DAO* and each $dao_i \in DAO$ is composed of a set of architecture decisions.

Table 1
Aggregate Functions. The *Max*, *Min* and Σ operations are over all architecture decisions that are connected to each other in the specified way (parallel or sequence).

QoS Attribute	Parallel	Sequence
Response Time	$Max(q_{ka})$	$\sum q_{ka}$
Energy Consumption	$\sum q_{ka}$	$\sum q_{ka}$
Cost	$\sum c_{ka}$	$\sum c_{ka}$

   - The set Q of qualities of interest. Response time and energy consumption are examples of quality attributes of interest.
   - Each d_{ka} has a cost and quality which may vary over time. The cost of each architecture decision is $c_{ka}(t)$ and quality is $q_{ka}(t)$, where ka identifies a decision $d_{ka} \in D$, c is a measure of cost, $q \in Q$ is a measure of quality and t is a time stamp. In particular, each architecture decision d_{ka} will be associated to one measure of cost and $|Q|$ measures of quality at each time stamp.
2. *Assessing the weight of each quality of interest*: A ranking weight (w_q) must be chosen by the stakeholders to reflect the relative importance of each quality attribute to the runtime benefit of the system as depicted in Fig. 4, which should satisfy Eq. (1):

$$\sum w_q = 1; \forall q : w_q \geqslant 0 \tag{1}$$

3. *Quantifying the benefit of DAO over time:* The benefit $B_{dao_i}(t)$ of dao_i at timestamp t is a measure of the contribution of each quality attribute to value creation, i.e., added value. In other words, each $B_{dao_i}(t)$ is a function of $q_{dao_i}(t)$, $\forall q \in Q$, whereas each $q_{dao_i}(t)$ is a composite of the quality $q_{ka}(t)$ of each of its component architecture decisions $d_{ka} \in dao_i$. To compute $q_{dao_i}(t)$, the qualities of the architecture decisions need to be aggregated based on how they are connected to each other (line 2). Table 1 depicts some aggregate functions for quality of service (QoS).
   We monitor the benefits of *dao* as a whole. The modeling for *dao* qualities follows linear aggregation and is consistent with online QoS modeling approaches that have been widely adopted in the service computing community (e.g., Zeng et al., 2004; Ramírez et al., 2017). Though the use of linear aggregation for qualities is the widely adopted practice in service community, it is acknowledged to be limited when capturing dependencies of decisions affecting qualities. Additional online aggregation functions, for capturing dependencies of decisions affecting qualities, is a non-trivial problem; its solution will constitute a significant contribution to both the software architecture and services community, which is worth separate reporting due to the complexity of its treatment.
   Each $q_{dao_i}(t)$ has one constraint, which follows one of the following possible formats: $q_{dao_i}(t) \leqslant q^{max}$, $q_{dao_i}(t) \geqslant q^{min}$, $q^{min} \leqslant q_{dao_i}(t) \leqslant q^{max}$.
   To place all quality attributes in the same scale (line 3–8), Eq. (2) is for scaling of negative quality values (i.e. the lower the better, e.g. response time), whereas Eq. (3) could be used for scaling positive quality values (i.e. the higher the better, e.g. throughput). $q'_{dao_i}(t)$ denotes the normalized value of a given q of a particular dao_i at time t.

$$q'_{dao_i}(t) = \begin{cases} \dfrac{q^{max} - q_{dao_i}(t)}{q^{max} - q^{min}} & if\ q^{max} - q^{min} \neq 0 \\ 1 & if\ q^{max} - q^{min} = 0 \end{cases} \tag{2}$$

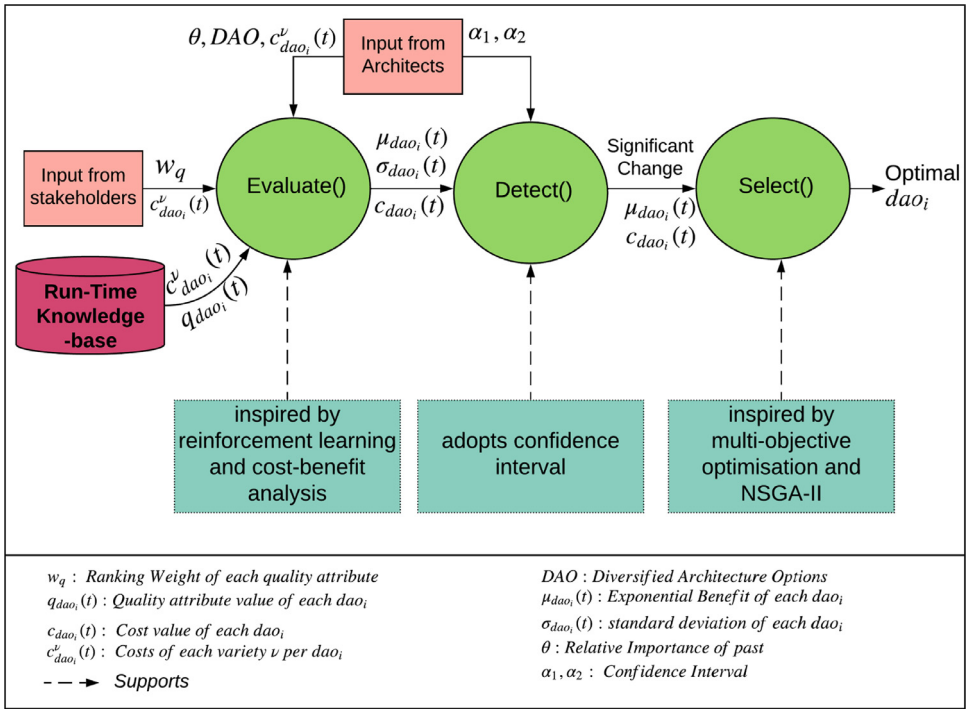


Fig. 4. An overview of the operational procedures of run-time evaluation approach.

$$q'_{dao_i}(t) = \begin{cases} \dfrac{q_{dao_i}(t) - q^{min}}{q^{max} - q^{min}} & if\ q^{\max} - q^{\min} \neq 0 \\ 1 & if\ q^{\max} - q^{\min} = 0 \end{cases} \quad (3)$$

Therefore, the benefit of a diversified architecture option can be computed using Eq. (4), if none of its quality attributes violates any constraint (line 11). If a *dao* at time *t*, violates any constraint (line 9), its benefit is set to zero (line 10).

$$B_{dao_i}(t) = \sum_{q \in Q} w_q * q'_{dao_i}(t) \quad (4)$$

This step uses the run-time knowledge (i.e. observed QoS) to compute the benefit of each *dao*, which is then used as input to step 5 (Fig. 4).

4. *Quantifying the cost of DAO:* CBAM (Nord et al., 2003) extends ATAM (Architecture Trade-off Analysis method) (Kazman et al., 2000) with explicit focus on the costs and benefits of the architecture decisions in meeting scenarios related to quality attributes. Our consideration for the cost is situation dependent. As an example, the cost can relate to one or more dimensions of interest. This can include the cost of configuration, deployment, testing, leasing, execution etc. These costs can be estimated using parametric models, back-of-the-envelope estimation, reliant on experts (i.e. architects and other stakeholders) and their judgment, analogy, *etc*, as well as run-time knowledge (i.e. monitoring tools). Unlike CBAM, our approach considers the switching costs between options, which could include the configuration, license cost, *etc*. This is in addition to the operating costs, such as costs of deploying and maintaining the *dao*. The cost associated with an architecture option at time *t* is denoted by $c_{dao_i}(t)$ (line 12), which is computed using Eq. (5) (Fig. 4).

$$c_{dao_i}(t) = \sum c^{\nu}_{dao_i}(t) \quad (5)$$

where ν: considers variety of costs (e.g. deployment cost, leasing cost, *etc*). Further, the approach receives monetary values for cost, which could then be normalized in the same way as negative quality values in Eq. (2). In line of our previous contribution (Sobhy et al., 2016), the assumption here is that some critical concerns may require alternative design decisions and these need to be modeled as part of the architecture. The decision of which alternative design decisions to consider for run-time evaluation is taken at design-time; it is expected that an additional effort may materialize as a result. The premise is that providing alternative options can render benefits in the running software system that could overcome the extra modeling cost. At run-time, the system can incur an additional switching cost.

5. *Determine the exponential benefit of DAO over time:* The benefit of a given option is monitored using a time-decay function inspired by reinforcement learning (Wang et al., 2015). This function enables the architect to continuously learn the current benefit of a given *dao* over time. At a given timestep *t*, the exponential benefit $\mu_{dao_i}(t)$ of a *dao* is the time-decayed average of its benefit, incrementally computed based on all timesteps up to *t* (line 13), as seen in Eq. (6). This average allows us to tune the importance given to the present and past observations of the benefit and to learn more about the behavior of *DAO* over time.

How much emphasis is given to the present/past is controlled by a pre-defined parameter θ. In particular, the relative importance of the present is denoted by $1-\theta$, whereas of the past by θ, where $0 \leq \theta < 1$. The θ parameter affects the system's stability. More specifically, a high θ (e.g., larger than 0.95) corresponds to a greater emphasis on the historical observations of benefit (i.e. present), leading to more robustness to noise, making the quantification of benefit more stable, but slower at adapting to changes. Conversely, smaller values give more emphasis to the more recent observations of benefit and swifter adaptation to changes. However, too low values can lead to unstable quantification of benefit (due to higher sensitivity to noise). To this regard, the architect has the freedom to adjust the θ parameter,

with respect to the required system stability (Fig. 4).

$$\mu_{dao_i}(t) = \theta\mu_{dao_i}(t-1) + (1-\theta)B_{dao_i}(t) \quad (6)$$

6. *Determine the variance and standard deviation of dao over time:* The time-decayed variance $\sigma^2_{dao_i}(t)$ of the benefit of this option is computed using Eq. (7) (line 14). After that, the standard deviation $\sigma_{dao_i}(t)$ is computed as the square root of variance as depicted in Eq. (8) (line 15).

$$\sigma^2_{dao_i}(t) = \theta\sigma^2_{dao_i}(t-1) + (1-\theta) * (B_{dao_i}(t) - \mu_{dao_i}(t))^2 \quad (7)$$

$$\sigma_{dao_i}(t) = \sqrt{\sigma^2_{dao_i}(t)} \quad (8)$$

4.4. Detecting significant changes over time

At every timestep t, the detect module triggers an alert if there is a significant change for the worse on the benefit $B_{dao_i}(t)$. If such a detrimental change is detected, then it may be necessary to switch from the current *dao* to another better *dao* for the current circumstances of the software system. The steps for detecting significant detrimental changes are shown in Algorithm 2 and are explained below.

ALGORITHM 2: Detect().

Input: confidence intervals (α_1, α_2), exponential benefit $\mu_{dao_i}(t)$, standard deviation $\sigma_{dao_i}(t)$
Output: change/warning/no change is detected

1 $\mu^{\max}_{dao_i} = \mu_{dao_i}(0)$, $\sigma^{\max}_{dao_i} = \sigma_{dao_i}(0)$
{Update the maximum exponential benefit and its corresponding exponential standard deviation}
2 **if** $\mu_{dao_i}(t) > \mu^{\max}_{dao_i}$ **then**
3 $\quad \mu^{\max}_{dao_i} = \mu_{dao_i}(t)$
4 $\quad \sigma^{\max}_{dao_i} = \sigma_{dao_i}(t)$
end
{Check if a change is detected}
5 **if** $\mu_{dao_i}(t) - \sigma_{dao_i}(t) \leq \mu^{\max}_{dao_i} - \alpha_2 * \sigma^{\max}_{dao_i}$ **then**
6 $\quad$ a change is confirmed
7 $\quad$ reset $\mu^{\max}_{dao_i}$ and $\sigma^{\max}_{dao_i}$
{Check if a warning is triggered}
8 **else if** $\mu_{dao_i}(t) - \sigma_{dao_i}(t) \leq \mu^{\max}_{dao_i} - \alpha_1 * \sigma^{\max}_{dao_i}$ **then**
9 $\quad$ a warning is triggered
else
10 $\quad$ no change/warning is detected

To detect changes, we use the confidence interval of the maximum exponential benefit seen so far and its corresponding exponential standard deviation, as shown in Eq. (9). In particular, two variables computed based on the evaluate phase are used: $\mu^{\max}_{dao_i}$ is the maximum exponential benefit seen so far, and $\sigma^{\max}_{dao_i}$ is its corresponding standard deviation. Whenever a new reading arrives at time t, those values are updated if $\mu_{dao_i}(t) > \mu^{\max}_{dao_i}$ (line 2–4). The parameter α is a parameter that affects the confidence level (Gama et al., 2004). In particular, confidence levels 95% and 99% correspond to $\alpha = 1.96$ and 2.58, respectively.

$$\left[\mu^{\max}_{dao_i} - \alpha\sigma^{\max}_{dao_i}, \mu^{\max}_{dao_i} + \alpha\sigma^{\max}_{dao_i}\right] \quad (9)$$

If the current exponential benefit $\mu_{dao_i}(t)$ is outside the left boundary of the confidence interval (line 5), a significantly detrimental change is detected (line 6). This leads to the insight that the current *dao's* benefit is getting worse and may need to be replaced.

Replacements may affect the system's stability. The parameter α enables us to tune the sensitivity of the approach to changes, and therefore how stable/unstable it will be over time. For instance, consider that the architect has chosen the confidence level of 95%. Then, if the current exponential benefit is outside the left boundary of the 95% confidence interval ($\mu_{dao_i}(t) - \sigma_{dao_i}(t) \leq \mu^{\max}_{dao_i} - 1.96 * \sigma^{\max}_{dao_i}$), a significantly detrimental change is informed to the software architects. However, if a 99% confidence interval been chosen, a significantly detrimental change would only be informed when $\mu_{dao_i}(t) - \sigma_{dao_i}(t) \leq \mu^{\max}_{dao_i} - 2.58 * \sigma^{\max}_{dao_i}$. In this context, the detect step uses $\mu_{dao_i}(t)$ and $\sigma_{dao_i}(t)$ along with α parameters (set by the architect) to detect changes, as shown in Fig. 4.

When a significantly detrimental change is detected, $\mu^{\max}_{dao_i}$ and $\sigma^{\max}_{dao_i}$ are reset and recomputed from scratch using the values of the exponential benefit accumulated since a warning was triggered (line 7). A warning is triggered based on a more relaxed confidence interval (line 8 and 9). For example, if using 99% as the confidence interval to detect significant detrimental changes, a confidence level of 95% could be used to issue a warning. Otherwise, no change/warning is detected (line 10).

To summarize, a change detection method is necessary to check whether the current *dao* is getting worse based on its accumulated exponential benefit $\mu_{dao_i}(t)$. In this context, the change detection test used could be adjusted to detect only significant changes and hence improves the stability of the system (i.e. avoid alerting the software architects of insignificant triggers).

4.5. Selecting the architecture option with the optimal trade-offs

If a significant detrimental change is detected in the *dao* currently being used (dao_{curr}), this means that it may be beneficial to replace this *dao* by another one from *DAO* elicited in step 1 of Section 4.3. In such a situation, it is desirable to know which *dao* among *DAO* has recently been providing the optimal trade-off between cost and benefit (Fig. 4). Such a *dao* is assumed to be the best one to use now and hence the best one to switch to.

To determine the *dao* with the optimal trade-offs, we were inspired by the literature on MOP, NSGA-II, and a knee selection method (Branke et al., 2004; Deb et al., 2002). Our problem has two objectives: maximize benefit (Eq. (6)) and minimize cost (Eq. (5)). It is therefore a multi-objective problem with two objectives. As such, we calculate the *expected* marginal loss in order to identify the *dao* with the likely most balanced trade-off between benefit and cost. The process of determining the *dao* with the optimal trade-offs between cost and benefit at time t is divided into three sub-processes as explained below and summarized in Algorithm 3.

1. *Determine the non-dominated DAO:* This includes determining which diversified architecture options are non-dominated by any other *dao* (Branke et al., 2004; Deb et al., 2002) (line 2–9). A given dao_i dominates dao_j iff:($c_{dao_i}(t) \leq c_{dao_j}(t)$ and $\mu_{dao_i}(t) \geq \mu_{dao_j}(t)$) and ($c_{dao_i}(t) < c_{dao_j}(t)$ or $\mu_{dao_i}(t) > \mu_{dao_j}(t)$). According to the definition above, non-dominated architecture options (i.e. knee point solutions) can be considered as better than dominated ones.
2. *Calculate the marginal loss of the non-dominated DAO over time:* The marginal loss is used for the purpose of computing the *expected* marginal loss (Step 3). In particular, we use a loss function that aggregates cost and benefit into a single value as follows (line 12):

$$L_\lambda(dao_i, t) = \lambda c_{dao_i}(t) - (1-\lambda)\mu_{dao_i}(t), \quad (10)$$

where $\lambda \in [0, 1]$ is a pre-defined parameter that controls the relative importance between cost and exponential benefit. The loss describes how (un)desirable a certain *dao* is.

ALGORITHM 3: Select().

Input: change detected in dao_{curr}, a pre-defined parameter λ, number of λ ($|\lambda|$), number of *DAO* ($|DAO|$), exponential benefit $\mu_{dao_i}(t)$, cost $c_{dao_i}(t)$
Output: optimal dao_i

{A change is detected in dao_{curr}, then do:}
{Initialization:}
1 $\lambda : \{0.1, 0.2, \cdots, 0.9\}$, $|\lambda| = 9$
{Determine the Non-Dominated *DAO*:}
2 **for** *i = 1 to* $|DAO|$ **do**
3 $dominant = 0$
4 **for** *j = 1 to* $|DAO|$ **do**
5 **if** $(c_{dao_i}(t) \leq c_{dao_j}(t) \& \mu_{dao_i}(t) \geq \mu_{dao_j}(t)) \& (c_{dao_i}(t) < c_{dao_j}(t) | \mu_{dao_i}(t) > \mu_{dao_j}(t))$ **then**
6 dao_i dominates dao_j
7 $dominant = 1$
 end
 end
8 **if** $dominant = 0$ **then**
9 Add dao_i to list of non-dominant *DAO*
 end
end
{Calculate marginal loss for the non-dominant *DAO*:}
10 **for** *i = 1 to Length (list of non-dominant* $|DAO|$*)* **do**
11 **foreach** $\lambda \in \{0.1, 0.2, \cdots, 0.9\}$ **do**
12 $L_\lambda(dao_i, t) = \lambda c_{dao_i}(t) - (1-\lambda)\mu_{dao_i}(t)$
13 **if** $i = argmin_i L_\lambda(dao_{i'}, t)$ **then**
14 $L'_\lambda(dao_i, t) = \min_{(i \neq ii)} L_\lambda(dao_{ii}, t) - L_\lambda(dao_i, t)$
 else
15 $L'_\lambda(dao_i, t) = 0$
 end
 end
end
{Determine the expected marginal loss for the non-dominant *DAO*:}
16 **for** *i=1 to Length (list of non-dominant* $|DAO|$*)* **do**
17 $approxM[L'_\lambda(dao_i, t)] = \frac{\sum_{\lambda \in \{0.1, 0.2, \ldots, 0.9\}} L'_\lambda(dao_i, t)}{|\lambda|}$
end
18 Return optimal $dao_i = dao_i$ with $\max(approxM[L'_\lambda(dao_i, t)])$

The marginal loss $L'_\lambda(dao_i, t)$ represents how much worse the loss would be if dao_i was not available and we had to use the one with the second optimal trade-off, given a certain λ (line 13–15). It can be calculated based on the following equation (Branke et al., 2004):

$$L'_\lambda(dao_i, t) = \begin{cases} min_{(i \neq ii)} L_\lambda(dao_{ii}, t) - L_\lambda(dao_i, t) \\ \quad : if \quad i = argmin_i L_\lambda(dao_{i'}, t) \\ 0 \quad : otherwise \end{cases} \tag{11}$$

where *argmin* function is used to check if dao_i has the minimum loss. If this is true, then we will iterate over all *DAO* to find the minimum difference between losses of a particular dao_{ii} and dao_i. This is set as the marginal loss. If dao_i is not the one with the minimum loss, we set $L'_\lambda(dao_i, t)$ to 0.

3. *Determine the expected marginal loss of diversified architecture options over time:* The *dao* with the optimal trade-off between cost and benefit at time t is the *dao* with the maximum expected marginal loss at time t. This option can be suggested to the software architect as the optimal *dao* to be adopted at time t. This *dao* may or may not be the same as current option dao_{curr}.
As in Branke et al. (2004), we use an approximation of the expected marginal loss rather than the true marginal loss. The expected marginal loss has been proposed and used in the MOP literature (Branke et al., 2004) to facilitate the choice of which solution from the Pareto front to adopt in practice. In contrast, a single-objective problem would use a single fixed value for λ. However, we need to compute the marginal loss with a sample of different λ values. In our work, we used equally spaced values λ: $\{0.1, 0.2, \ldots, 0.9\}$, where $|\lambda| = 9$ is the number of λ values used. The expected marginal loss ($approxM[L'_\lambda(dao_i, t)]$) can be approximated by taking the average of the marginal losses computed using different sampled values for λ (Branke et al., 2004) (line 16 and 17), as shown in Eq. (12). The optimal dao_i is the one with maximum expected marginal loss (line 18).

$$approx\,M\left[L'_\lambda(dao_i, t)\right] = \frac{\sum_{\lambda \in \{0.1, 0.2, \ldots, 0.9\}} L'_\lambda(dao_i, t)}{|\lambda|} \tag{12}$$

As explained above, to determine the *dao* with the optimal trade-off between cost and benefit at time t, we need to know the exponential benefit and standard deviation of the *DAO* at this timestep, including the *DAO* that are not currently in-use by the software system. This information can be obtained in a number of different ways:

- If a given dao_i uses the same D as the current dao_{curr}, but connects them in a different way, the exponential benefit and standard deviation can be tracked over time even if dao_i is not currently being used. This is because the qualities of interest of the D are being monitored via dao_{curr} and just need to be aggregated using a different function to compute dao_i's exponential benefit and standard deviation.
- A given $dao_i \neq dao_{curr}$ can be activated for use in parallel with dao_{curr} at pre-defined time intervals T' to track their qualities of interest. In this case, dao_j's exponential benefit and standard deviation are updated at every $T' > 1$ units of time, saving the overhead of having to use more than one *dao* at every timestep. The accuracy of dao_j's exponential benefit and standard deviation will depend on how large T' is.
- A simulation based on what-if test scenarios can be used to estimate the exponential benefit and standard deviation of $dao_i \neq dao_{curr}$. Simulators are typically used during the architecture prototyping, analysis, and refinement stages to evaluate the response and sensitivity of the architecture for these tests. In this case, dao_i's exponential benefit and standard deviation could be updated frequently, possibly at every unit of time. However, their accuracy depends on the simulator. Due to the exponential time-decay factor used for calculating exponential benefit, inaccuracies on exponential benefit of architecture options not in-use at time t can be quickly found if and when we switch from a *dao* to another. If the newly adopted one is found to have significantly poorer benefit than previously estimated, the change detection mechanism will detect this and trigger the procedure to determine whether we should switch to another one. For instance, if some of the IoT devices forming the selected *dao* become unavailable after selecting it due to the highly dynamic application environment, making it not possible to execute the selected *dao*, this will lead to a decrease in exponential benefit and hence cause a change to an alternative architecture.

In summary, our proposed approach monitors the extent to which the concerned architecture design decisions satisfy the qual-

ity attributes over time; it provides feedback on their performance against the said qualities. The feedback is used to adapt the architecture as seen fit. In particular, the feedback determines the benefit of diversified architecture options and hence can aid the architect in demonstrating the situations where the *dao* would work and the others which is not suitable for that context. As an example, if a particular *dao* does not perform well in any of the provided scenarios (e.g. when the energy consumption is a priority) over a prolonged period of time, this is a strong indicator of a wrong choice of the design decisions and choices. The approach incorporates human expertise in the decision of whether or not to change or keep this architecture as an option, as the modelling of diversified architecture options is the fundamental domain knowledge from the engineers, as for every software system.

5. IoT case study design

In this section, we introduce the IoT case, how diversity can be embedded in the architecture, and how the data is collected through iFogSim.

5.1. Introducing the IoT case

The basic concept of IoT is the interaction between a group of devices— "things"—such as sensors and actuators, over the Internet. Key challenges for IoT (Gubbi et al., 2013; Da Xu et al., 2014; Issarny et al., 2016; Nahrstedt et al., 2016) include:

- *heterogeneity*, having different types of things, such as static sensing (e.g. fixed sensors), mobile crowd-sensing (e.g. cellular-based, vehicle-based, and bike-based sensors), virtual (e.g. web services), and social sensing (e.g. share data across social networks like facebook);
- *high dynamism* due to the presence of mobile things and uncertainty of their resource demand and QoS provisions over time (i.e. service level objectives), varying energy consumption per thing and their varied availability;
- *scale*, where their ubiquitous, light, and mobile nature has led to the presence of, in some cases, millions of things.

To address these challenges, prior approaches have provided some solutions for IoT, such as Da Xu et al. (2014), Hachem (2014), Issarny et al. (2016), and Gupta et al. (2017). We draw on an IoT application, documented in Gupta et al. (2017), to demonstrate the applicability and effectiveness of the approach. Nevertheless, our approach has the potential to be applied to other systems exhibiting high dynamism and uncertainty in their operations. Our IoT application extends Gupta et al. (2017)'s application – an urban traffic monitoring system, named *iTransport*. Gupta et al. (2017) used a video surveillance application to demonstrate the usefulness of their proposed cloud/fog simulator tool *iFogSim*. However, in their case study, the context of run-time architecture evaluation and adaptability under time-varying environment have not been considered, even though iFogSim is capable of simulating the dynamics and uncertainty of cloud/fog environments. This has motivated us to extend their case study. In a nutshell, iFogSim is a cloud/fog simulation environment; it can aid developers to simulate the impact of their application on qualities of interest. It forms the basis in our work to mimic the dynamics and uncertainty of cloud/fog environments, and their impact on qualities of interest in our case study. Further explanation related to our use of iFogSim, which differs from that of Gupta et al., can be found in Section 5.3. In addition,Gupta et al. (2017) assume the presence of one application architecture (i.e. configuration). We have designed the multiple configurations with respect to the common architecture decisions of a video surveillance application.

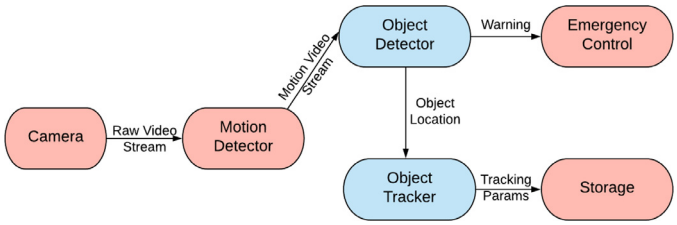


Fig. 5. The flow diagram of *iTransport* application Bittencourt et al. (2017).

The *iTransport* application provides online (for emergencies) and offline (for long-term prediction) analytics. It uses smart cameras, which are either fixed (attached to street lights and buildings) or mobile (attached to vehicles and bikes) to capture the traffic for accident avoidance and traffic management. The application has 6 modules as seen in Fig. 5: camera, motion detector, object detector, object tracker, accident storage, and emergency control. Smart cameras transmit raw video streams to the motion detector module, which then forwards the video in which motion was detected to the object detector module. The object detector module analyzes the objects and detects any abnormal actions (i.e. car accidents). If it observes an accident, the emergency control searches for a nearby ambulance for notification. The data is then sent to the accident storage cloud to profile the accidents with respect to areas. The application automatically provides either "online analytic" functionality or "offline analytic" functionality, every 10 minutes based on user requirements. For instance, if "online analytic" functionality is invoked, then minimized response time and network usage are necessary, whereas if "offline analytic" is called, then the energy consumption is the main goal for optimization.

5.2. Diversified architecture options in the context of IoT case

When architecting the *iTransport* application, the architects must address uncertainties due to heterogeneity of the things; the dynamicity of the things' behaviors and the dynamism of their composition. Design diversity can be employed to handle these uncertainties: the greater the uncertainty, the more diversity is applied Avizienis and Kelly (1984) in an attempt to improve performance and availability (Avižienis et al., 2004; Baudry et al., 2014b; Sobhy et al., 2016). We contend that diversification means embedding in flexibility. Since there is a variety of ways to diversify, each diversified architecture can be treated as an option, which we denote by *dao* (Sobhy et al., 2016). A software architecture encompasses a set of architecture decisions D, where a decision $d_{ka} \in D$. A *dao* implements a set of diversified decisions to meet some quality goals and trade-offs. d_k denotes a particular capability, including connectivity, data collection, data management, *etc.* d_{ka} indicates the software architecture components/connections that implement this capability. For example, the architects decided to diversify the *data collection* capability (d_1), where video could be captured using fixed cameras (d_{11}), mobile cameras (d_{12}), or both (d_{13}). Another diversification decision is concerned with the *connectivity and processing* capability (d_2), where the things can connect, track, and process the captured video on the cloud (d_{21}) or both cloud and fog (d_{22}). So dao_1 comprises d_{11} and d_{21}, whereas dao_2 consists of d_{12} and d_{22} and so forth. Table 2 depicts selected options, with decisions designed for cloud, fog, mobile, or fixed.

In *iTransport*, there are several design trade-offs concerning the critical QoS attributes (e.g., response time, energy consumption, network usage, *etc*) and cost, subject to constraints such as the pre-defined coverage and availability of the things. In the context of iTransport, we consider deployment cost (the expenses related to the infrastructure deployment in cloud/fog environment),

Table 2
Possible Diversified Architecture Options for *iTransport* application. The diversity in each *dao* can refer to using fixed and/or mobile fog devices for data collection capability; using different cloud providers and heterogeneous fog devices for data processing capability.

Option dao_i/Decision d_k	Data Collection Capability (d_1)	Data Processing Capability (d_2)
1	Fixed (d_{11})	Cloud (d_{21})
2	Mobile (d_{12})	Cloud (d_{21})
3	Fixed and Mobile (d_{13})	Cloud (d_{21})
4	Fixed (d_{11})	Fog and Cloud (d_{22})
5	Mobile (d_{12})	Fog and Cloud (d_{22})
6	Fixed and Mobile (d_{13})	Fog and Cloud (d_{22})

execution cost (the computational costs of running the processing tasks on cloud/fog devices), and networking costs (related to the bandwidth requirements and associated expenses. For instance, data uploading cost from end devices/sensors and inter-nodal data sharing cost) (Mahmud et al., 2018). Further, the switching costs in *iTransport* embrace the migration costs to/from the cloud/fog, thing's connectivity and other costs (if any). Nevertheless, the approach is flexible enough to include other costs. The design trade-offs can inform the diversification design decisions and the deployment of *dao*. Addressing the following scenarios require us to consider trade-offs when deciding on *dao*:

- dao_1 uses fixed camera sensors to provide more stable and better response time to fulfill the pre-defined coverage. However, achieving the coverage for the scale of city highways using fixed camera may incur much higher cost and static coverage. In contrast, the use of mobile crowdsensing (using smart vehicles) (Issarny et al., 2016), as in dao_2, could be an alternative solution due to its low cost. But the mobile crowdsensing in dao_2 may be unstable in terms of response time and it can consume much more power at this scale due to the simultaneous transmission, processing, and remote execution of the images on the cloud. Further, availability in dao_2 is much more restricted than that of dao_1.
- When comparing with the case where cloud is used as the sole computation paradigm (in dao_1 to dao_3), the partial use of fog (in dao_4 to dao_6) could provide faster response time (e.g., for scenarios such as emergency notification and online analytics) and lower network usage, due to the offloading of computational load on the near by fog devices. But it may incur more energy consumption, given the large number of required fog things and the additional overhead that may be required to synchronize and store the processed information on the cloud (if any). In addition, the fog option needs to fulfill the constraints on the proximity of the thing to the fog and the availability of the fog.

There are some scenarios where design-time decisions may fail to select the "right" options due to the run-time uncertainties and dynamics caused by various environmental factors, which emergently affects the benefit of the options. In particular, it is possible that, at run-time, their benefit deviates more than the expected value at design-time, for example:

- The design-time evaluation suggests dao_6 to be continuously deployed when response time and network usage are the stakeholders' concerns, due to its high expected benefit. Conversly, at run-time (i.e. output of simulator), the hyper-connectivity of mobile things and the high network latency affect its actual benefit in terms of response time and network usage, which was significantly lower than expected.
- The architect has decided to implement dao_1 in cases where response time and network usage is not a concern, aiming to improve the energy consumption in fog devices. However, the actual overall benefit (e.g. response time, network usage and energy consumption) was much worse than expected during run-time. This is because the sensors are still performing some processing to transmit the data to the cloud (i.e. there is high power load on the fog devices).
- The architect has prioritized the response time concern over the network usage and energy consumption. S/he selected dao_2 for deployment due to the use of mobile things, which may have lower impact on network congestion as compared with fixed ones. On the contrary, the architect has discovered that the actual benefit of the selected *dao* performed much worse than expected, due to the presence of very large number of mobile things (i.e equivalent to the deployment of lower fixed number of sensors), which resulted in high network usage. Therefore, dao_2 was almost violating the quality constraints in most of the cases.

The prior scenarios motivate the need for the run-time evaluation to capture conditions that may not be discovered at design-time. The approach can accumulate run-time information (i) discovering patterns related to the availability of mobile nodes and their connection to benefit improvement/degradation and value; (ii) the added value of switching to a diversified option and when that value will be realized, become optimal or cease to exist, considering various costs and load. The dynamism of the above cases make it difficult for designers to solely evaluate the architecture based on design-time knowledge. Run-time knowledge can be particularly useful to suggest refinements for the diversified architecture options; informing when a *dao* should (not) be invoked; phasing out a *dao* or suggesting a replacement, *etc.*

5.3. Experimental data collection

Our experiments focus on architecting the sensing-actuating functionality of *iTransport* to show how run-time evaluation can complement design-time evaluation. At design-time, the architect has followed the procedure of Section 5.2 to preliminary decide on the *DAO* (and their composing *D*) for implementing this functionality. The experiments were executed on 3.1 GHz Intel Core i7 processor machine with 16GB of RAM and Mac OSX. The data synthesis process is performed using iFogSim (Gupta et al., 2017), whereas Matlab is exploited for data analysis. To simulate the qualities of interest of each architecture decision over time, we adopt the *iFogSim* (Gupta et al., 2017) tool. This tool builds on Cloudsim (Calheiros et al., 2011); it provides the architect with the freedom of hierarchically composing the fog devices, clouds, and data streams. In iFogSim, we have hierarchically composed the application as shown in Fig. 5. The candidate *DAO* used in this study are shown in Table 2. In particular, each *dao* is composed of different types of data collection (type of sensors) and connectivity (computation locations) as architecture decisions, meaning that the processing performed by each *dao* is executed differently. Connectivity was simulated by either executing the *object detector* and *tracker*

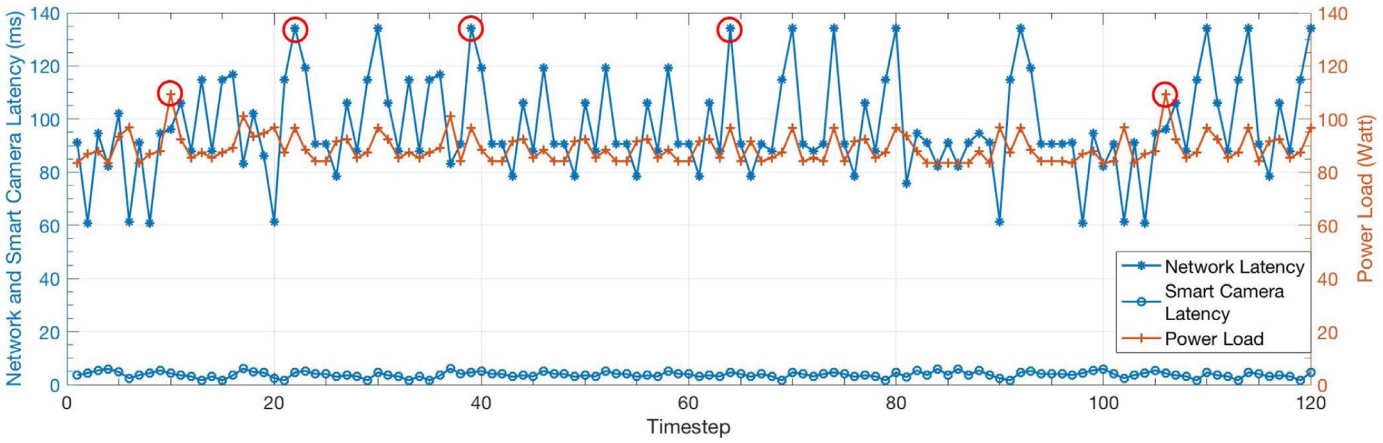


Fig. 6. Sample of changing environmental conditions facing *iTransport* (i.e. input for one of the *DAO*). *The red circles denote some examples of accidental spikes.* (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

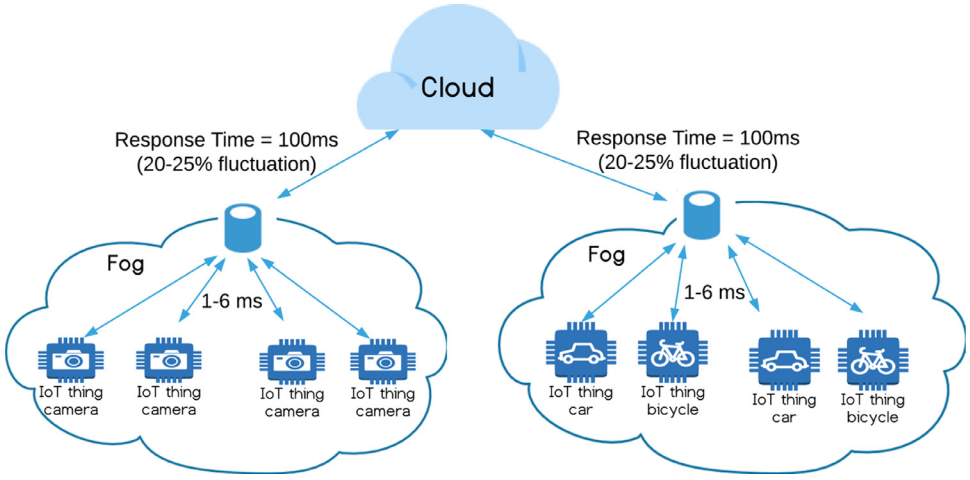


Fig. 7. The Initial Experiment Configuration for iFogsim.

modules (shown in Fig. 5) in the cloud and/or fog. For data collection, two gateways were used, where each is connected to an average of 50 smart cameras (Fig. 7): fixed, mobile, as well as fixed and mobile, having a total of 100 fog devices. The fog devices and cloud are configured based on Gorbold (April 2010),Guérout et al. (2013) and Gupta et al. (2017).

The goal is to continually optimize the following two conflicting requirements:

- *Benefit:* It needs to be maximized and is based on three quality attributes of interest:
  1. *Response Time: iTransport* is time-critical application, so it should respond as early as possible. In *iTransport*, the response time (RT) of an application is the application's end to end delay (in milliseconds *ms*) and measured using iFogSim.
  2. *Network Usage:* High network usage would cause network congestion, so it should be as low as possible. The network usage (NU in mega bytes *MB*) is also measured using iFogSim.
  3. *Energy consumption:* IoT will lead to unlimited energy consumption if not controlled (Reporter, 2016). Therefore, energy consumption needs to be minimized. In this context, when designing IoT architectures, architects have to consider energy consumption as a major concern (Kazman et al., 2018). In *iTransport*, the energy consumption (EC in mega joules *MJ*) is the total energy consumption by all the devices in the application, which is also determined through iFogSim.
- *Cost:* It needs to be minimized. It is a composite of operating cost, and switching cost that is added once we switch. In the context of *iTransport*, the average cost encompasses a thing's connectivity (includes switching), execution in the cloud and/or fog (i.e. leasing cost of processing services), and other costs mentioned in Section 5.2. The mean overall costs have been collected from iFogSim.

The iFogSim tool takes as input: the network latency, power load, smart camera's latency (i.e. fog devices), number of cameras, number of gateways, tuple configurations, cloud and fog devices configurations. The sources of uncertainty in *iTransport* come from the varying network delays due to network congestion. There are other factors, which as well impact the environmental conditions, such as uncertainty in QoS of cloud service providers (e.g. Amazon, Google Cloud, *etc*) and software-defined capabilities of fog service providers in terms of their processing power, as well as the hyperconnectivity of the nodes. In this context, we have generated data corresponding to each *dao* including typical as well as worst case scenarios. For instance, we varied the power load as 80–110 W with a fluctuation of 5–10%, following the typical power load values from Guérout et al. (2013). The latency was varied in the range of low to high, i.e. 1–6 ms with a fluctuation of 20–30% on the smart camera. We also varied the network latency with an average of 100 ms and fluctuation of 20–25%, as exemplified in Fig. 6. The choice of this fluctuation was based on Chen and Kunz (2016),

Table 3
Simulation Parameters for *iTransport*.

Parameter	Value
Device Configurations:	
Cloud Datacenter	3 GHz CPU, 40GB RAM, $0.1-0.3/day
Wifi and ISP Gateways	3
ins,GHz CPU, 4GB RAM, $0.0053-0.0056 /day	
Smart Camera	[1.6, 1.867, and 2.113] GHz CPU, 4 GB RAM,$0.0053-0.0056 /day
Number of Smart Cameras/Devices' OS	80–120 cameras/Linux OS
Network Configurations (Average Latency):	
From ISP Gateway to Cloud Datacenter	80–120 ms
From Wifi Gateway to ISP Gateway	1–6 ms
From Smart Camera to Wifi Gateway	1–6 ms
Tuple Configurations (Message size):	CPU Length (MIPS), Network Length (Bytes)
Raw Video Stream:	1000, 20,000
Motion Video Stream:	2000, 2000
Object Location:	500, 2000
Warning:	1000, 100
Tracking parameters	28,100
Average QoS:	
Energy Consumption of devices	80–120 MJ
Applications Response time	300–4000 ms
Network Usage	500 KBytes- 2 MBytes
Evaluation Settings:	
θ	{0.7, 0.9, 0.99}
α_1, α_2	{80, 92%}, {90, 95%}, {99, 99.9%}
{Normal;Strict} Quality Constraint	{[400ms, 130MJ, 2MB];[350ms, 130MJ, 500KB]}
Normalized {Operating;Switching} Cost	{0.3 – 0.5;0.2 – 0.4}
Weights for {normal; strict} Quality constraints	{[0.4,0.3,0.3]; [0.4,0.2,0.4]}

as it normally provides acceptable throughput across various networking protocols, but also causes accidental spikes that represent worst case scenarios.

We also simulated changes by using diverse smart cameras' configuration (Guérout et al., 2013; Gupta et al., 2017), as depicted in Table 3. Based on that, it outputs the energy consumption of devices, application's response time, and network usage. This setting was intentionally designed as a worst case that goes beyond a stable setting. Further, the iFogSim takes the pricing configurations for IoT devices (i.e. fog devices) and cloud to generate the mean costs of each *dao* (i.e. application architecture). All the pricing configurations are used with respect to AWS IoT services (AWS, 2018). We have run the simulations for each *dao* for 120 timesteps.

6. Experimental evaluation

In the next series of experiments, we aim to show the usefulness of the approach by evaluating how well it addresses the research questions introduced in Section 3. We will also evaluate the scalability of the approach.

As aforementioned, the system that uses the *dao* evaluated by our approach as having the optimal trade-offs is called *Informed-Selection System*. We compare our approach against the following baseline selection systems:

1. *Static-selection system:* This is a typical type of system used in practice (Taylor et al., 2009; Sobhy et al., 2016), where the expert implements a single *dao* based on its assessed value at design-time. For our work, the value is determined using the Binomial option pricing model (Brandão et al., 2005; Ozkaya et al., 2007; Sobhy et al., 2016), which estimates the future benefits and costs of options based on a binomial decision tree.
2. *Predefined-selection system:* This is inspired by Gokhale (2007) and Meedeniya et al. (2011), where the architect will choose the *dao* that is likely to perform the best for a given context. This selection is typically based on experience, backed up by back-of-the-envelope calculations for the cost, benefits, and technical potential. However, the selection may fail to predict potential fluctuations in value, quality potentials, and costs. As an example, our case used dao_6 during week days (because of peak hours) and dao_3 during weekends (because of less demand).
3. *Random-selection system:* Our design for the baseline system follows the argument of Nahrstedt et al. (2016) and Wagner et al. (2016) but for the context of services. When a significant change is detected, it selects a *dao* randomly independent of its QoS over time. This is because the *DAO* are deemed to be functionally equivalent but deployed in different environments and geographical location (i.e., distributed Fogs/clouds). All the results related to the Random-selection approach are based on the average of 30 runs (the choice of 30 runs is recommended by Arcuri and Briand, 2011).

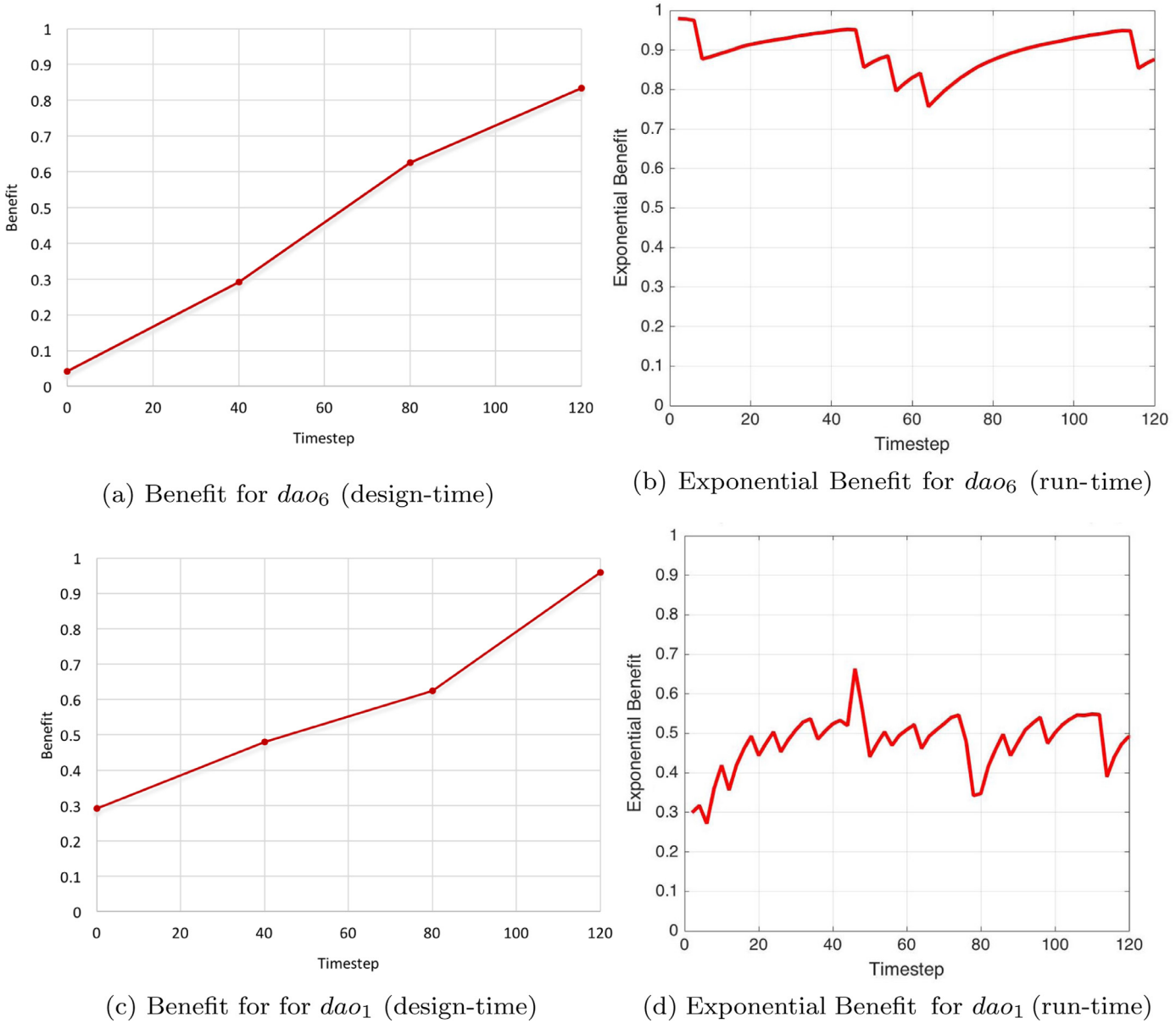
6.1. Evaluation of the usefulness of the approach

Next, we will provide answers to RQ1 and RQ2.

RQ1: How to evaluate the benefit of each dao over time?

Motivation: This experiment aims to show the usefulness of run-time evaluation over design-time evaluation. More specifically, it conveys that the run-time evaluation visualizes scenarios and dynamics, which can hardly be captured at design-time. It is also important to confirm the design-time choices. For that, we compare our approach against the design-time architecture evaluation approach proposed in Sobhy et al. (2016). The latter uses *options theory* (Hull, 2006) to evaluate and justify the employment of architecturally diversified decisions and their augmentation to long-term value creation under uncertainty. The architect has the freedom to estimate the increases and decreases in the value potentials for the candidate architecture options over time, backed up by their experience.

Experimental setup: The design-time architecture evaluation approach uses experts' (e.g. architects and other stakeholders) assumptions on the likely utilities of a *dao* over a pre-defined pe-



(a) Benefit for dao_6 (design-time)

(b) Exponential Benefit for dao_6 (run-time)

(c) Benefit for for dao_1 (design-time)

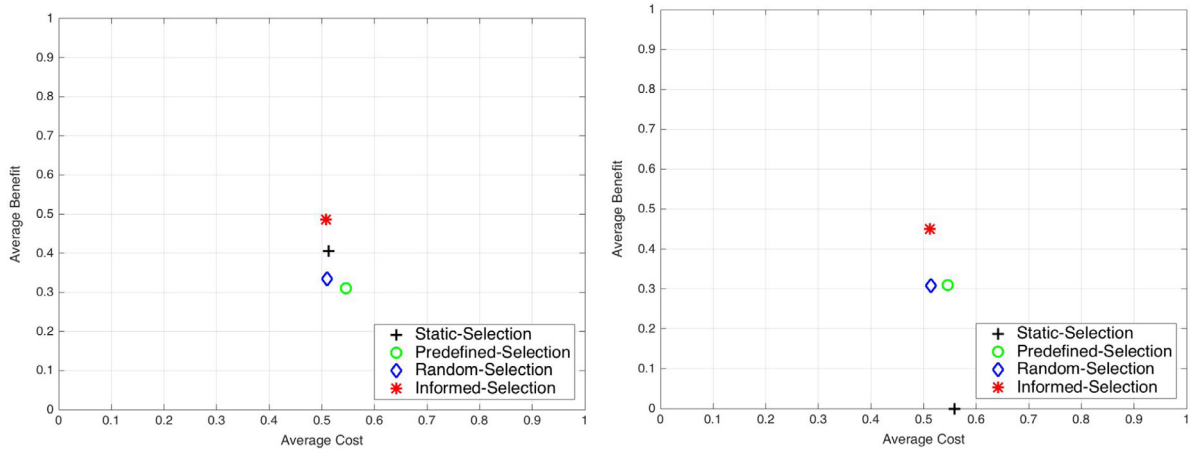
(d) Exponential Benefit for dao_1 (run-time)

Fig. 8. The results of assessing design-time and run-time evaluation.

riod of time (Sobhy et al., 2016). In our experiments, the pre-defined period of time corresponded to 120 timesteps. The expert's opinion is depicted by a utility tree that is provided at design-time, without making use of any run-time information. The utility tree created for the design-time approach used in the experiments is shown in Fig. A.1a and A.1b. The design-time approach (Sobhy et al., 2016) then uses this utility tree to compute the likely benefit of a *dao* over the pre-defined period time based on binomial real option analysis. This benefit is the one depicted in Fig. 8a and 8c. Therefore, even though this is a design-time evaluation approach, it provides information on the expected run-time benefit of the *DAO*, being a meaningful design-time approach to compare against.

We investigate how the design-time architecture evaluation approach and our proposed run-time approach evaluate two *DAO*: the *dao* with the best benefit (dao_6) over time and the *dao* with the worst benefit (dao_1). Fig. 8a depicts the *dao* with best benefit (i.e. dao_6) over time computed using the design-time approach (Sobhy et al., 2016), whereas Fig. 8b shows the exponential benefit quantified using our run-time architecture evaluation approach. Further, the benefit of dao_1 (i.e. worst) is shown in Fig. 8c, whereas its exponential benefit is plotted in Fig. 8d. Note that we have normalized the design-time benefit to ensure fair comparison with the run-time benefit, as the design-time evaluation approach provides a monetary value for the benefit of *DAO*. We assume that the cost of both options is constant over time, whereas the benefit is varying over time.

Analysis: As we can see, the design-time approach was conservative and estimated that dao_6 had initially low benefit and then improved over time. Our run-time approach, on the other hand, is able to show to the software architect that dao_6 has high benefit from the beginning. Fig. 8c shows the benefit value of dao_1 over time computed using the design-time approach (Sobhy et al., 2016). In this scenario, the design-time approach incorrectly estimated that the value of dao_1 was going to increase over time. Our approach was able to discern at run-time that this was not really



(a) Average Benefit and Cost for the best case scenario across 120 timesteps.

(b) Average Benefit and Cost for the worst case scenario across 120 timesteps.

Fig. 9. The evaluation of the four approaches' decision-making process under strict quality constraints.

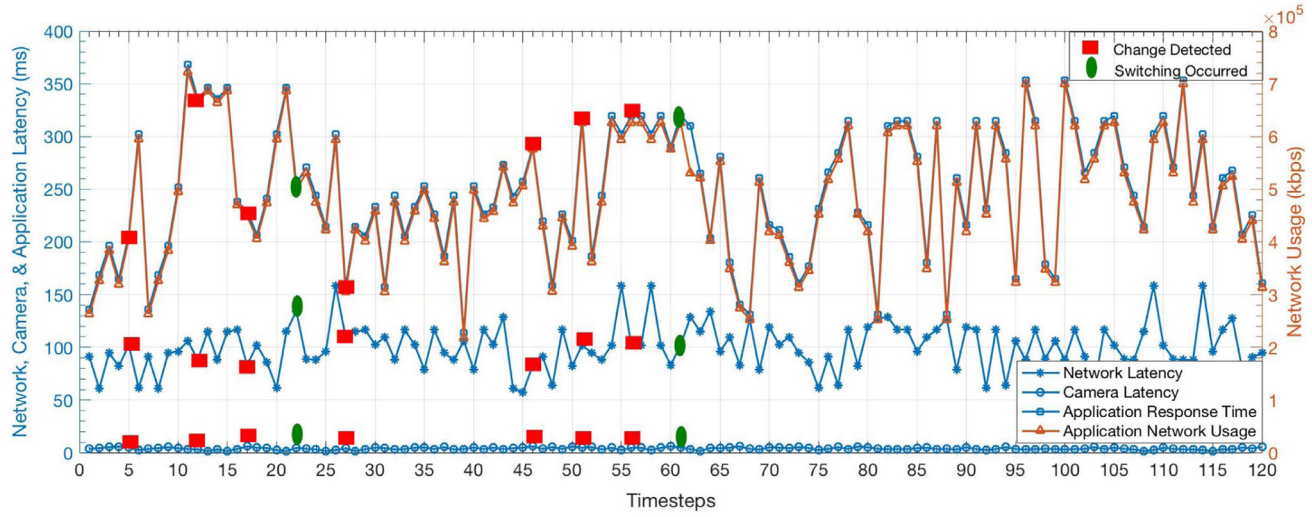
the case, as shown in Fig. 8d. Even though the exponential benefit ascended from 0.3–0.7 until timestep 25, this was followed by a deterioration to an average of 0.4 (Fig. 8d).

RQ2: How can run-time evaluation determine changes in dao's value over time and inform subsequent decisions?

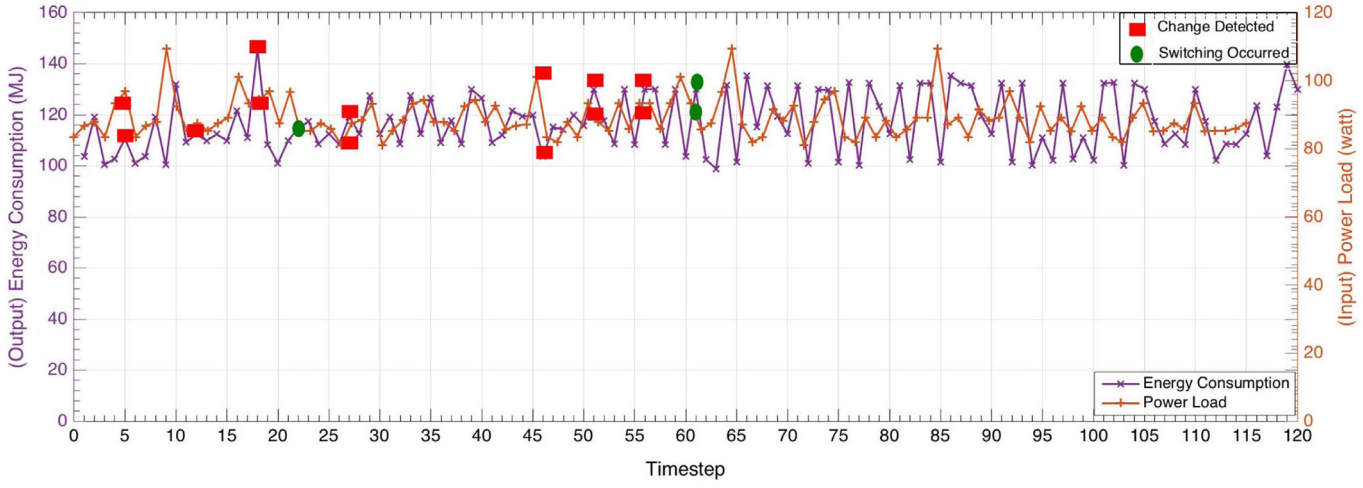
Motivation: This experiment aids the architect to evaluate the *DAO* and suggest the ones with most balanced cost-benefit trade-offs using the informed-selection approach as compared with other approaches introduced earlier in Section 6. For that, we consider that when our approach detects a significant detrimental change, the software architect decides to implement the *dao* that our approach recommends as the one with the optimal trade-off between cost and benefit, based on *strict* quality constraints. This experiment will also aid the architect in refining the selection of design-time *DAO* by checking their cost-benefit at run-time. It can also indicate which *DAO* were not performing well, and hence require phasing-out. Back to IoT context, the use of fog computing and cloud computing in mobile crowd sensing applications is highly affected by the QoS requirements. Consider a scenario where the stakeholders require the *iTransport* application to quickly track the accident in city center, especially in rush hours. In this context, low response time and network usage is higher concern rather than energy consumption. Therefore, the ranking score (i.e. weights) for response time and network usage qualities are higher than energy consumption. Also the use of fog-cloud computing is advisable over cloud computing. So object detector and tracker modules will be executed in the fog. We are uncertain about the network latency (due to dynamic traffic and variable load) and mobility of devices (nodes join/leave the network). This will cause instability, which may require a switch to another architecture option.

Experimental setup: The environmental conditions that keep changing at run-time are network latency, camera latency and power load. These potentially affect the aggregated benefit, which is composed of application response time, energy consumption in devices and cloud, and network usage. Therefore, we might not get a linear effect from input to output – this is due to the use of aggregated exponential benefit, where the change detection and selection is based on it. Next, we will demonstrate strict quality constraint scenario to show whether the changes detected by the approach are aligned with the input environmental conditions. Consider the case where the architect can adjust the quality weights to reflect priorities from stakeholders. For example, when the application response time and network usage become a priority, energy consumption priorities can be downgraded. In our case study, we assume a w_{RT} of 0.4 for response time, w_{EC} of 0.2 for energy consumption, and w_{NU} for network usage of 0.4. We also consider that the architect has constrained the application to handle the request in less than 350 ms and network usage does not exceed 500 KB; whereas, the energy consumption should not exceed 130 *MJ*. Historical performance and experts' judgment can inform the adjustment of the prior constraints (Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are, 2016; Gupta et al., 2017). In this experiment, the focus is on the application's response time and network usage concerns. Nevertheless, the same experiment could be applied on other stakeholders' concerns, such as the ones discussed in Section 5.2. Fig. 9 shows the average benefit and cost of the system as a whole across 120 timesteps, i.e., calculated based on the *DAO* currently being used for two scenarios: *best case* and *worst case*. We have also plotted the environmental changing conditions (the power load, network latency, and camera latency) for best case scenario, in Fig. 10a and 10b, along with their impact on response time, network usage, and energy consumption.

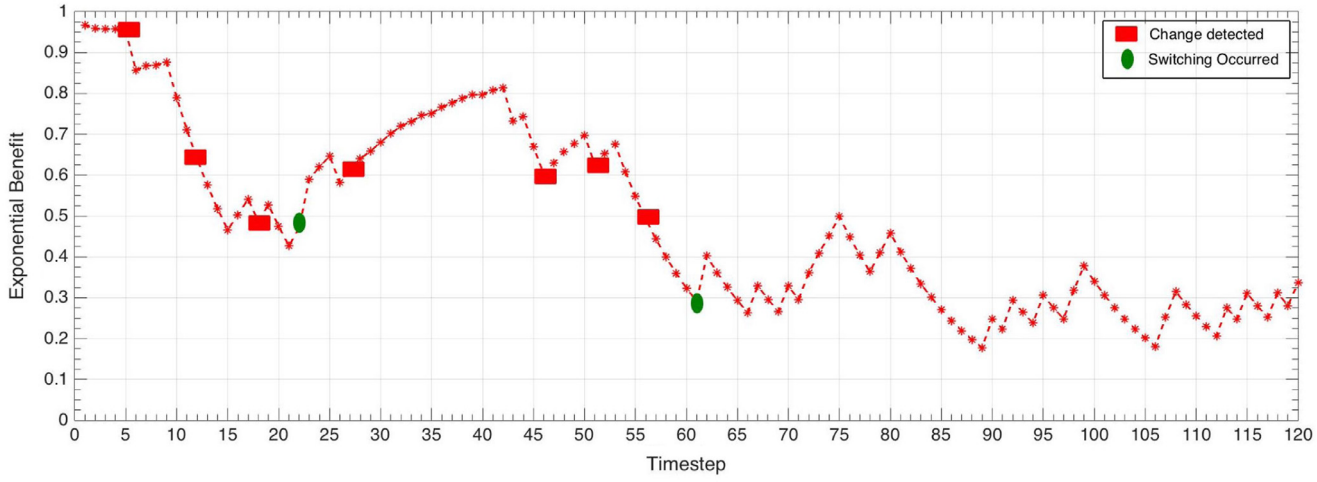
Analysis (best case scenario): The design-time evaluation approach suggests the systems to start operation using the *dao* that are believed to provide the most balanced cost-benefit trade-off at run-time (dao_6). Different changes were observed after 5 timesteps (Fig. 9a). The mobility of devices has caused a decrease in the overall benefit, due to highly changing response time causing a violation in response time constraint. The random-selection approach selects a random *dao*, which is not always the best. In addition, throughout the experiment, this approach suffered from too many switches (9 switches), causing instability and lowering the benefit to about 0.35. The pre-defined selection performs a bit worse than the random one; this is because the constraint was too strict for options recommended by that approach, causing various violations. The static-selection is the second best one; this is because the selection is geared towards selecting *dao* with better potentials for the selected scenarios. This strategy considers design-time knowledge, which can be challenged or confirmed by future



(a) The impact of network and camera latency on application's response time and network usage.



(b) The impact of power load on application's energy consumption.



(c) Exponential Benefit of informed-selection with change detection and switches.

Fig. 10. An illustration of the input environmental conditions for the 120 timesteps including network latency, smart camera's latency, power load on the devices, change detection, and switching occurrence, as well as the output exponential benefit of informed-selection approach for strict quality constraints and prioritized ranking score for the best case scenario as an example. *The red squares represent change detections that did not lead to switching DAO, and green circles represent change detections followed by dao switching.* (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

runs of the system. The informed-selection approach provides the most appealing results compared with other approaches: this is because the informed-selection approach can continually recommend the *dao* with the most balanced cost-benefit trade-offs. For example, we observed that when the first change was triggered, the informed-selection figured out that the initially selected *dao* was still the optimal choice to keep. Yet, after 22 timesteps when another change was detected, dao_4 was recommended due to its large improvements for response time. However, this requires cost of leasing these services and maintaining their devices (switching cost).

By mapping the environmental conditions to the evaluation of *DAO* based on the informed-selection approach (Fig. 10a and 10b), the approach has detected significant deviations which are consistent with input environmental conditions. In particular, the change could be triggered either from high fluctuation and/or a deterioration in one/all of QoS. For instance, from timestep 1–5, we see an increase in the values of application response time and network usage, caused by a (smaller) increase in network and camera latency (Fig. 10a). These result in a decrease in the exponential benefit, which is considered as significant when reaching timestep 5 (Fig. 10c).

Since the approach is still building knowledge about the current *dao* and having highly dynamic changing conditions during the initial observation period, this caused a change detection at almost every consecutive 5 timesteps until timestep 27 (Fig. 10). However, these only led to switches when another better *dao* was available (at timestep 22). This led to improvements in the exponential benefit of the proposed informed-selection approach over the following timesteps (Fig. 10c).

Further, at $t = 46$ (Fig. 10a), an increase in network camera latency has caused a noticeable rise in application's response time and network usage, which accordingly resulted in the detection of a significant change in environmental conditions. However, the approach found that the current *dao* still provided the most balanced cost-benefit trade-off and hence the approach kept using this *dao* for the next 15 timesteps (though other change detections were triggered). After that, at $t = 61$, the approach detected a significant change (i.e. a high power load, network and camera latency) and has then selected another *dao* which provided the most balanced cost-benefit trade-off. After $t = 61$, the exponential benefit was just oscillating resulting in no significant changes (i.e. no changes were detected), as reflected by the exponential benefit (Fig. 10c).

Analysis (worst case scenario): The design-time evaluation approach suggests the systems to start operation using the *dao* that turns out to provide the worst balanced cost-benefit trade-off at run-time (i.e. dao_2). In this respect, a replacement is advocated by our approach. Fig. 9b shows that the most balanced trade-off between benefit and cost is achieved by the informed-selection approach, followed by the random- and predefined-selection approaches. Here, informed-selection achieved much higher benefit than other approaches. Since the response time constraint is violated from the beginning, the static-selection approach experiences a zero benefit. In this context, high application response time is not recommended, because this is a safety-critical application. From this experiment, dao_1 has never been recommended by the approach because of its low response time and high energy consumption, which may inform the architect to phase-out. For the best and worst case scenarios, the architect can use the run-time evaluation approach to visualize the cost-benefit trade-offs of the suggested *DAO* over time (Fig. 9) for informed-selection approach against other approaches. It can also show how the adoption of optimal *DAO* provided an improved benefit over time (i.e. added values).

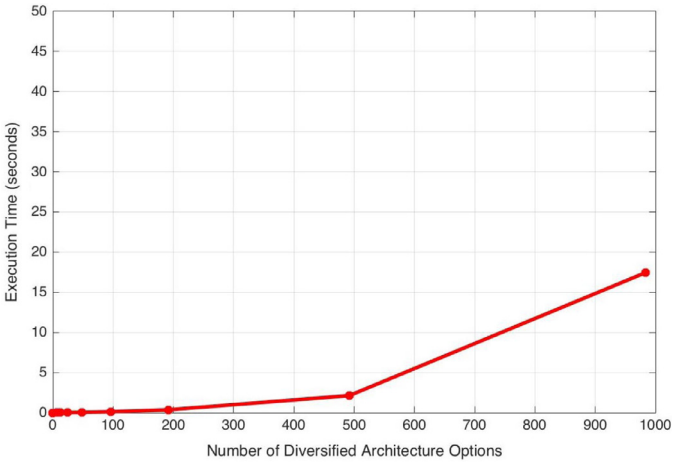


Fig. 11. The execution time for large number of *DAO* for every timestep.

6.2. Scalability of the proposed approach

Motivation: This experiment will answer the following: *How scalable is the proposed approach to larger numbers of dao?*. The scalability of the proposed approach (informed-selection) is an important indicator to show to what extent our run-time evaluation can be applied in practice. There is a trade-off between the candidate options and the execution time of the algorithm.

Experimental setup: In this experiment, the mean execution of the informed-selection algorithm is computed over 120 timesteps for varying number of *dao*, as illustrated in Fig. 11.

Analysis: The informed-selection approach performs very well until reaching 500 *DAO*, it takes less than 3 s. This is applicable for safety-critical IoT applications, which in case a change is detected, the approach will be able to search for the optimal solution in few seconds. However, after 500 architecture options, the execution time starts to increase reaching 17 s for 1000 options. Therefore, for an application, which requires more than 1000 options, our approach will take further time for decision-making.

7. Further analysis of the proposed approach

This set of the experiments aims at evaluating the robustness and scalability of the proposed approach. It also aims at providing a better understanding of the influence of the stability parameters (θ, α) on the proposed approach. This better understanding can guide software architects in the decision of which values to use for these parameters.

7.1. Impact of frequency of monitoring

Motivation: The proposed approach depends on the possibility of either monitoring the *DAO* that are not currently in-use, or using a simulator to get an idea of their likely behavior. If we opt for monitoring the *DAO* that are not currently being used, this will lead to an overhead in terms of time taken for the informed-selection approach to do the analysis. Therefore, one may opt for not monitoring them very often. Practically, its hard to monitor the architectural options (e.g. virtual sensors, physical sensors) every timestep, due to their availability. There is a trade-off between the execution time and % error, which we aim to measure in this experiment. To this regard, this could guide the architect on deciding the monitoring intervals. In this context, this experiment aims to provide answers for the following question: *What is the impact of the frequency of monitoring on the accuracy of the approach?*.

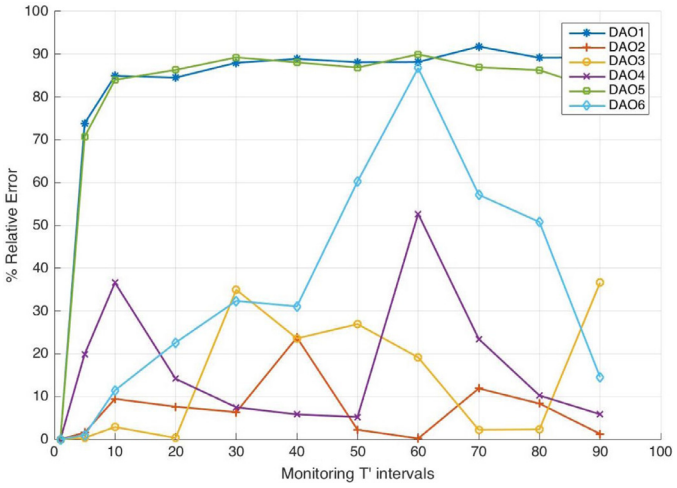


Fig. 12. The % Relative Error of monitoring every T' intervals for each *dao*.

Experimental setup: For that, we use an error metric *% Relative Error* to measure the error in decision-making between monitoring every T' intervals and every timestep. In this context, we will measure the error *for each dao separately* and *mean exponential benefit for the whole process*. For the former, the $\%\,Relative\,Error = \left|\frac{Actual\mu_{dao_i}(t) - Monitored\mu_{dao_i}(t)}{Actual\mu_{dao_i}(t)}\right| * 100$. The actual exponential benefit is the one monitored every timestep till T', where $T' = \{5, 10, 20, 30, 40, 50, 60, 70, 80\}$. The monitored exponential benefit is the one monitored every T' intervals. For example, if T'=5, then from t=1 to 4, the exponential benefit will be the same and then at t=5, it is updated. Fig. 12 depicts the % relative error versus T' intervals for each *dao*. As for the *mean exponential benefit for the whole process*, the $\%\,Relative\,Error = \left|\frac{ActualMean\mu - MonitoredMean\mu}{ActualMean\mu}\right| * 100$. The actual mean exponential benefit is the average benefit if monitoring occurs at every timestep, so it is constant for all. The monitored mean exponential benefit is the average benefit if monitoring occurs at every T' intervals. We consider the error in the whole process to evaluate how much worse the mean benefit (based on informed-selection approach) would be if monitoring happened every T' intervals rather than every timestep. This includes the monitoring, change detection, and decision-making processes. We have developed two scenarios: *best case* (Case 1 and 3), where the optimal *dao* is initially chosen; *worst case* (Case 2 and 4), where the worst *dao* is initially selected. This is illustrated in Fig. 13, where each point in the plot has 3 coordinates (X: execution time; Y: monitoring interval; Z: % Error)

Analysis (for each dao): In Fig. 12, the relative error for dao_2 and dao_3 ranges from 0–30%. This is due to the low fluctuation in these options. This in turn lowers their impact on the accuracy. However, dao_1 and dao_5 violated the constraint at $t = 1$, which resulted in an error of about 70%. This is because dao_1 and dao_5 had good exponential benefit after the first timestep (i.e. not violating constraint), but the monitoring was based on the first timestep only.

Analysis (for whole process): Case 3 presents the smallest error (Fig. 13c), which was about 0.27%. The error was small because the initially selected *dao* was optimal and fluctuating less than the other *DAO*. As a result, the informed-selection approach managed to keep using it for different monitoring intervals (i.e. no switching). In this context, the architect could use higher monitoring intervals to save execution time. This is due to the decrease in execution time from 0.45 s (T' = 5) to 0.22 s (T'= 50).

Further, Case 1 experienced 10% increase in error (constant for all intervals), as shown in Fig. 13a. This rise is due to the fact that when monitoring happened at every timestep, three switches were recommended by our approach. However, when the monitoring interval increased, the approach had quite outdated information about the current *dao*. Therefore, it did not advocate any switches, which resulted in slight degradation in benefit. To this regard, if time is the concern, the architect could choose higher monitoring intervals to benefit from lower overhead in terms of time and cost.

On the contrary, the selection of the worst *dao* in cases 2 and 4 caused a significant rise of 65% in relative error (Fig. 13b and 13d), because the approach had outdated knowledge about the initial *dao*. To exemplify in case 2 at $T' = 50$, the approach updates the current benefit with respect to the first timestep. Therefore, for the first 50 timesteps the benefit was very low and a switch was recommended after $T' = 50$. On the other hand, if monitoring occurred at every timestep, then the approach will advocate switching after 5 timesteps to an optimal *dao*. This explains the rise in the relative error to 40%. For the latter scenarios, it is recommended for the architect to use lower monitoring intervals with higher overhead, rather than experiencing a reduction in benefit. To this end, our approach could aid the architect in tuning the monitoring interval with respect to execution time overhead and relative error.

7.2. Robustness to noise

Motivation: An alternative to monitoring each *dao* at regular T' intervals would be to use a simulator to monitor the likely benefit of the *DAO* that are not currently in-use. However, simulators are likely to produce noisy quality indicators, which differ from the actual qualities that these *DAO* would have if they were currently being used. In order to evaluate the impact of the noise potentially produced by simulators, we investigate how the proposed approach reacts to different levels of noise i.e. *To what extent the informed-selection system can deal with noise data as compared with other systems?*.

Experimental setup: In this experiment, we have generated *Gaussian Noise* (Zhu and Wu, 2004) on the QoS data. A given quality attribute $q'_{dao_i}(t)$ is replaced by a value drawn from a Gaussian distribution $\mathcal{N}(q'_{dao_i}(t), s)$, where $q'_{dao_i}(t)$ is the mean and $s = \{0.05(low), 0.1(mid), 0.5(high)\}$ is the standard deviation. The smaller/larger s represent the cases where there is less/more noise. This reflects the cases in which simulators are more/less reliable. It enables us to check how robust the approach is to wrong measurements provided by a simulator. In particular, we expect the proposed approach to be affected by such erroneous quality information, but to quickly react and recover from it if a poor switch occurs. This is because, once the switch occurs, the true benefit of the *dao* can be determined. If this benefit is worse than expected, a change will be detected, leading to a switch to another potentially better *dao*. The results are based on the average of 30 runs, due to the randomness of noise (the choice of 30 runs is recommended by Arcuri and Briand, 2011). In this experiment, we have compared the proposed approach against the state-of-the-art and baseline approaches introduced in Section 6. For this experiment, the design-time evaluation approach suggests the systems to start operation using the *dao* that are believed to provide the worst cost-benefit trade-off at run-time (dao_2). This choice is advocated to show how our approach can deal with different noise levels, in even worst case scenarios. Fig. 14a–14c show the exponential benefit of four approaches for the selected *DAO* across 120 timesteps. Whereas, Fig. 14d–14f depict the mean exponential benefit and cost of four approaches for the selected *DAO* across 120 timesteps.

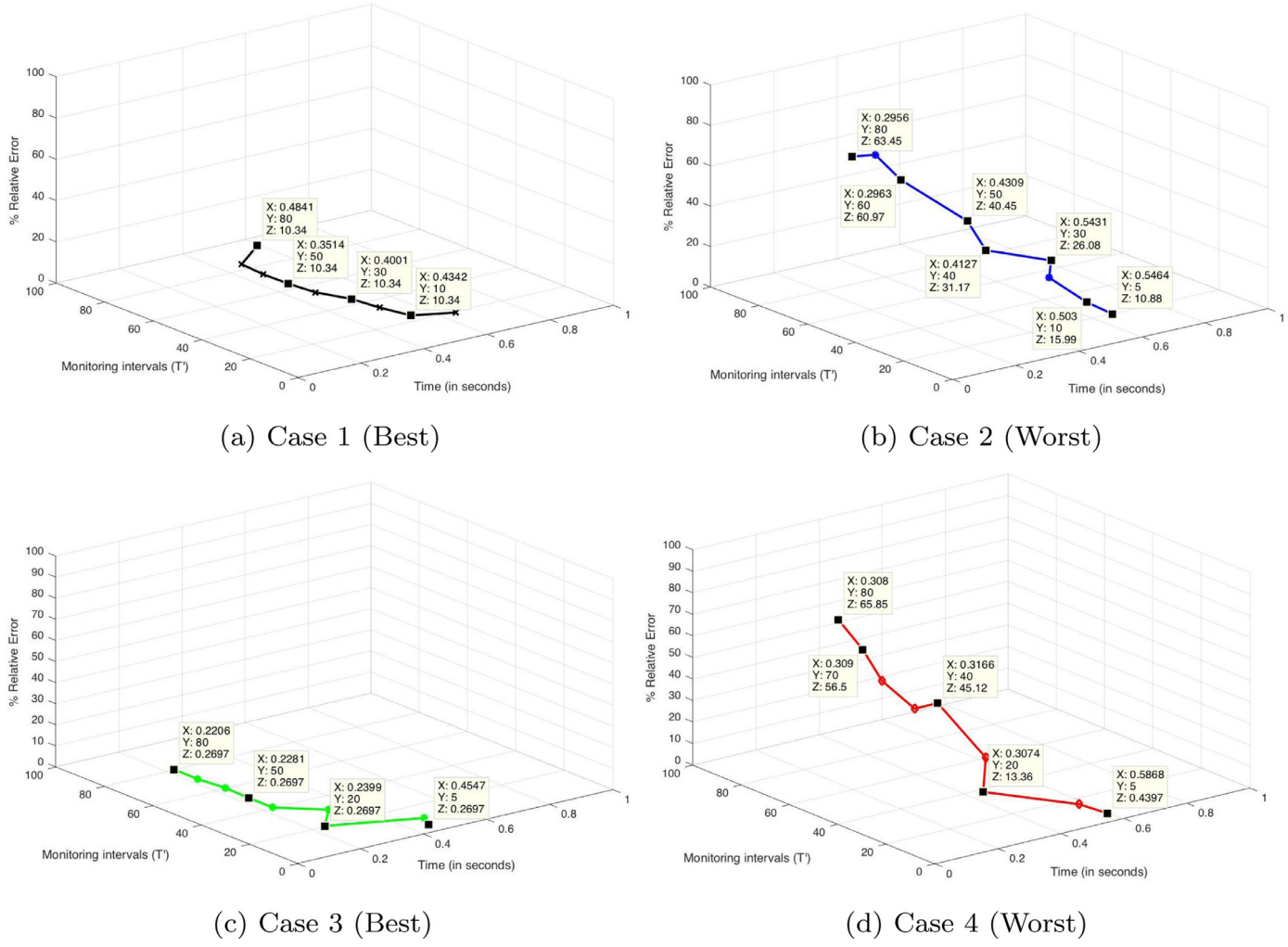


Fig. 13. The % relative error and execution time of monitoring every T' intervals for four scenarios.

Analysis: As clearly illustrated in Fig. 14, the informed-selection approach has managed to select optimal options, although by introducing higher noise levels, the choice could have become much worse. Followed by the random-selection, which benefited from the change detection test. However, it selected *DAO*, which were not always the best. The static-selection and predefined-selection approaches are based on design-time knowledge, thus they are not affected by noise. However, the static-selection produced zero benefit across 120 timesteps, due constraints' violation. The predefined-selection was highly fluctuating because of constraints' violation in some timesteps. Therefore, the informed-selection approach produced the best results overall (Fig. 14d–14f). Further, the informed-selection has quickly recovered from wrong choices due to noise. For instance in Fig. 14c, dao_2 was initially selected for deployment (in one of the runs), which turned out to be the worst *dao*. After 5 timesteps, the approach detected deterioration in exponential benefit, and hence recommended dao_4 to switch to. Ten timesteps later, another change is triggered and the informed-selection chose dao_5 to replace the current *dao*. We have found that dao_5 was not the optimal one and dao_6 seemed to be better (i.e. this choice was affected by noise). Though, the informed-selection approach has quickly recovered from the poor switch and suggested dao_6 after 5 timesteps. To this extent, our approach managed to self-repair from incorrect choices (due to varying noise levels).

7.3. Impact of the parameter θ on the system's stability

Motivation: A low θ value corresponds to a greater emphasis on the most recent observations of benefit and swifter adaptation to changes in the benefit. However, very low values can lead to unstable quantification of benefit, causing too much switching over time. A high value (e.g., larger than 0.95) places more importance to the past observations of benefit, leading to more stability, but potentially failing to track changes in benefit. This experiment will answer the following: *What is the impact of the parameter θ on the system's stability?*. In this context, this experiment is performed to indicate the most applicable value for the relative importance under typical environment settings.

Experimental setup: This experiment considers the values of $\theta \in \{0.7, 0.9, 0.99\}$. In this experiment, we started with dao_6, which is advocated by the architect to be the optimal *dao* for the normal quality constraints as depicted in Table 3. If a change is detected, the approach recommends another *dao* to be implemented. In this context, the exponential benefit of the selected *DAO* by the approach over 120 timesteps is computed and plotted in Fig. 15.

Analysis: As seen in Fig. 15, $\theta = 0.7$ leads to highly fluctuating benefit over time, which may cause the system to recommend a lot of switches throughout time, as compared with $\theta = 0.9$ (the moderate fluctuation) and $\theta = 0.99$ (too stable, which may be not be realistic). For instance, $\theta = \{0.7, 0.9, 0.99\}$ has recommended the

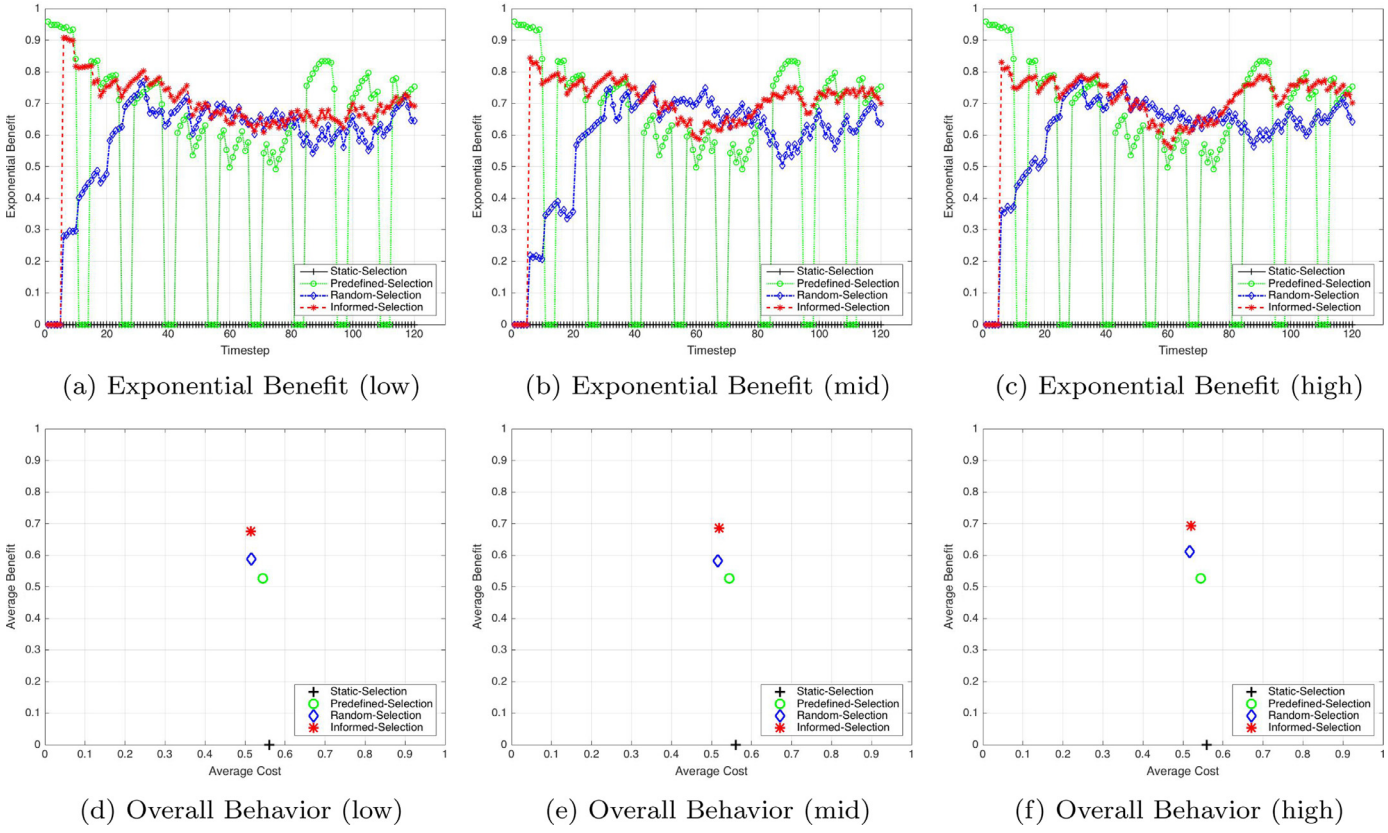


Fig. 14. The behavior of all the approaches after the application of varying noise levels on the data.

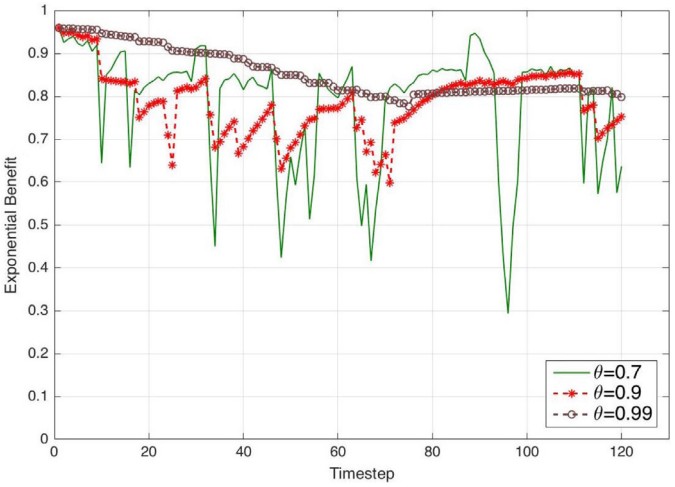


Fig. 15. Exponential Benefit of the informed-selection approach using different values for the stability parameter θ.

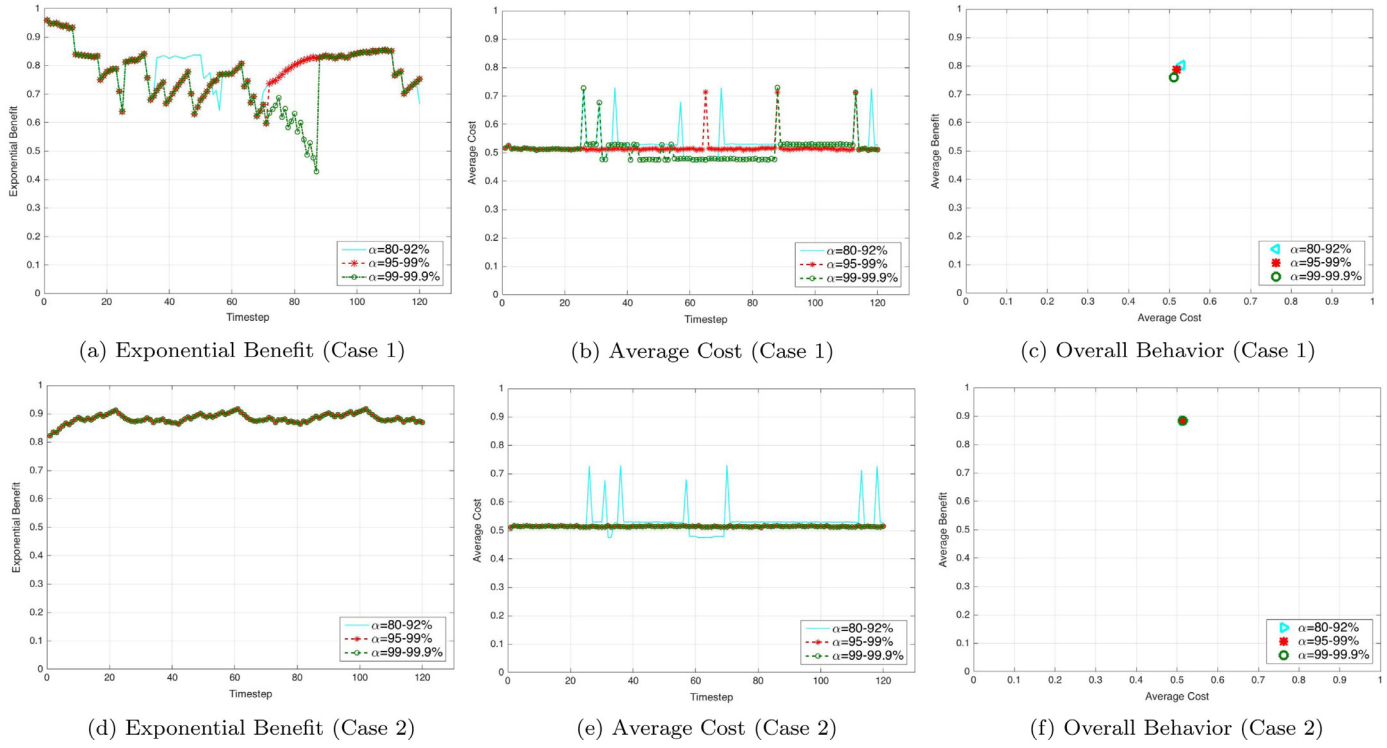
following number of switches {13, 4, 1}, respectively. Therefore, based on this experiment, we recommend for the architect to evaluate the four approaches for the next experiments, with respect to $\theta = 0.9$, as it indicates the most realistic forgetting factor.

7.4. Impact of the parameter α on the system's stability

Motivation: The confidence interval is an interval with two boundaries (α_1 and α_2), where the architect is not confident about the exponential benefit values outside this interval. In other words, if the current exponential benefit $\mu_{dao_i}(t)$ is outside the left boundary of the confidence interval, this is an indicator that the current option is getting worse and requires replacement. For instance, If a system is highly sensitive to changes, then we can enlarge the α values used ($\alpha \geq 95\%$), whereas for the opposite case ($\alpha \leq 92\%$) is more applicable and so forth. Based on that, the architect can learn what α values to use and for which scenarios. In order to adjust the α values, there is a trade-off between the number of switches and overall benefit (i.e. stability versus improved benefit), which will be measured in this experiment. To this end, the experiment aims at answering the following: *What is the impact of the parameter α on the system's stability?*.

Experimental setup: We have developed two cases for analysis: *case 1* where the best *dao* changed over time, which caused several switches; *case 2:* the best *dao* performed well over time, resulting in no switches. This experiment considers the values of α_1, $\alpha_2 \in \{80, 92\%\}$; {90, 95%}; {99, 99.9%}. Fig. 16 shows the impact of varying α values on the approach's stability, whereas Table 4 summarizes the corresponding number of change detection and switches. Fig. 16a and 16d show the exponential benefit of *DAO* recommended by the informed-selection approach, whereas Fig. 16b and 16e depict their corresponding cost. The mean exponential benefit and cost of 120 timesteps for varying α values are depicted in Fig. 16c and 16f to show the overall behavior of four approaches.

Analysis (case 1): The use of low confidence interval {80,92%} caused higher change detection than other confidence intervals. This is due to the detection of unnecessary changes. So in a case where the *DAO* are highly changing, this caused higher number of switches (7), as depicted in Table 4 and Fig. 16a. A high confidence interval {99,99.9%} can lead to neglecting significant changes, be-



(a) Exponential Benefit (Case 1) (b) Average Cost (Case 1) (c) Overall Behavior (Case 1)

(d) Exponential Benefit (Case 2) (e) Average Cost (Case 2) (f) Overall Behavior (Case 2)

Fig. 16. The impact of varying α values on the informed-selection approach.

Table 4
The number of change detection and switching for informed-selection approach for varying α values.

Parameter/Case	1	2
# of switches ($\alpha_1 = 80\%, \alpha_2 = 92\%$)	7	0
# of detection ($\alpha_1 = 80\%, \alpha_2 = 92\%$)	13	9
# of switches ($\alpha_1 = 95\%, \alpha_2 = 99\%$)	4	0
# of detection ($\alpha_1 = 95\%, \alpha_2 = 99\%$)	10	0
# of switches ($\alpha_1 = 99\%, \alpha_2 = 99.9\%$)	4	0
# of detection ($\alpha_1 = 99\%, \alpha_2 = 99.9\%$)	8	0

cause the system is confident enough about the data. There is a trade-off between the number of switches and improvement in benefit. For case 1, the informed-selection recommended 3 additional switches (when $\alpha = \{80, 92\%\}$ over $\alpha = \{99, 99.9\%\}$), with 5% increase in exponential benefit and 0.5% increase in cost over {99,99.9%}. Although {95,99%} and {99,99.9%} advocated the same number of switches, yet {95,99%} produced a 3.6% increase in exponential benefit and 1% increase in cost over {99,99.9%}. This is because {99,99.9%} neglected significant changes, as stated before.

Analysis (case 2): Though for {80,92%} the informed-selection detected 9 changes (Table 4), yet it managed to continue with the optimal option without any recommendation for switching. This is because it has found that there is no other *dao* better than the current one. Therefore, the informed-selection approach has a safety mechanism, which recommends a switch only if there is another better *dao*, otherwise it will keep using the same *dao*. Further, no change was detected for {95–99%} and {99–99.9%}, this explains why all the confidence intervals produced the same overall behavior as seen in Fig. 16f. Besides, the low confidence interval also caused higher overhead in terms of searching for another optimal option (informed-selection). In all cases, the informed-selection approach managed to recommend the optimal *dao* in terms of balancing between cost and benefit over time. To this regard, a confidence interval of {95–99%} is more applicable in most of the cases, because it detects significant changes and neglect unnecessary ones. We recommend the architect to use {95–99%} in the evaluation of the four approaches for the next experiments.

8. Application of our approach to analyse the usefulness of diversification

This section discusses and demonstrates the usefulness of diversification and whether it can always add value to the decision-making based on what-if scenarios.

Motivation: Design diversification has the potential to mitigate risks and improve the dependability in design in situation exhibiting uncertainty in operation and usage (Avižienis et al., 2004; Baudry et al., 2014b). Design diversity has raised the potential awareness to apply diversification in the decision-making process, which in turn may cause a noticeable improvement in the way we design dependable and evolvable software. However, do diversification continuously deliver value over time? In this experiment, we aim to evaluate the added value of diversification to the decision-making process.

Experimental setup: In this experiment, we are trying to show whether the inclusion of new *DAO* will benefit the decision-making. To demonstrate that, we have tested our approach with respect to two cases: (1) Add new six *DAO* where the approach benefits from them and a noticeable improvement in the overall behavior occurred (Case 1); (2) Add new six *DAO* where the approach does not benefit from them (Case 2). We used the original six *DAO* recommended by the approach (Table 2) and created new six *DAO* adhering to the same topology of original six *DAO*, but with different QoS fluctuations to generate Cases 1 and 2.

For Case 1, we have randomly generated their QoS over time using the mean of original *DAO* and fluctuation of $5\% - 25\%$ for response time, $0\% - 15\%$ for energy consumption, and $20\% - 29\%$

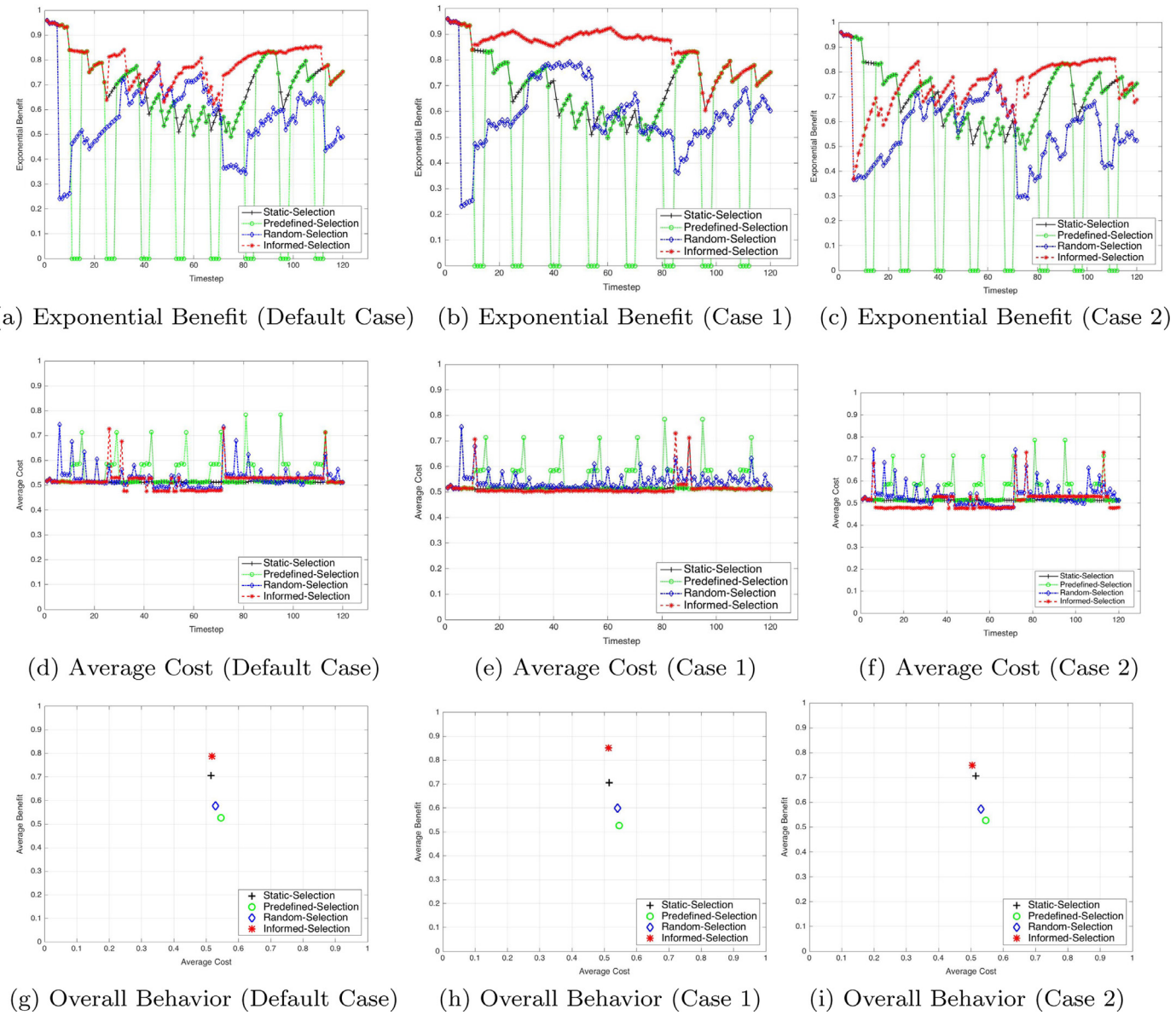


Fig. 17. The evaluation of embedding diversification to the architecture on the decision-making (positive impact (Case 1) and negative impact (Case 2)) as compared to the default case (6 original *DAO* in Table 2).

for network usage. As for Case 2, the QoS fluctuation is 30% – 40% for response time, 25 – 40% energy consumption, and 30% – 40% for network usage. These volatility values were chosen with respect to Balasubramanian et al. (2009) and Dejun et al. (2010). In Case 1, the low fluctuation has resulted in additional *DAO* with improved aggregated benefit, whereas the newly created *DAO* in Case 2 has generated *DAO* with highly fluctuating benefit (i.e. seems good at the beginning but then turns out to be worse after being selected).

Analysis: In *Case 1*, we have added 6 *DAO*, which seemed to add value to the decision-making process. The overall average benefit and cost for the informed-selection approach was better than the other approaches (Fig. 17g and 17h), with noticeable rise in exponential benefit over time (Fig. 17a and 17b), and slight decrease in cost (Fig. 17d and 17e). This is because the informed-selection approach has benefited from the inclusion of new *DAO*, by providing more stable behavior in terms of benefit. For instance, after 10 timesteps the approach has detected a significant change in current option (i.e. dao_6). It has then selected dao_{12} instead, which seemed to provide more stable benefit (Fig. 17b) with almost similar cost (Fig. 17e) to the one chosen in default case (i.e. 6 *DAO* instead of 12 *DAO*).

In *Case 2*, the additional 6 *DAO* were highly fluctuating over time, which explains the slight degradation in benefit (Fig. 17c) as compared to the original 6 *DAO* only (Fig. 17a). However, the addition of new options has introduced further instability to the random-selection approach by selecting the worse *DAO* (4 more switches than the original case), as depicted in Table 5. For example, after 5 timesteps, the approach has detected a significant deviation in current option (i.e. dao_6). After that, it chose dao_{11}, which provided the most balanced trade-off between benefit and cost. The exponential benefit of dao_{11} was increasing, which explains why the approach did not detect any change. However, this has caused the application to not benefit that much from diversification. At $t = 71$, the approach detected a change and till $t = 120$, it has chosen similar *DAO* to the default case.

Table 5
The evaluation of embedding diversification to the architecture with respect to the number of change detection and switches.

Parameter/Approach	Static -selection	Predefined -selection	Random -selection	Informed -selection
# of switches (6 *DAO*)	0	8	8	4
# of detection (6 *DAO*)	0	0	8	10
# of switches (12 *DAO*)[Case 1]	0	8	10	3
# of detection (12 *DAO*)[Case 1]	0	0	10	4
# of switches (12 *DAO*)[Case 2]	0	8	12	4
# of detection (12 *DAO*)[Case 2]	0	0	12	5

To this regard, the proposed approach successfully showed that diversification was helpful in Case 1, and was not helpful in Case 2. This concludes that diversification will not always add value and run-time evaluation could aid in assessing the worthiness of this exercise.

9. Discussion and threats to validity

In this section, we will discuss the usefulness and applicability of proposed approach with respect to experiments introduced in Sections 6 and 7 and threats to validity.

9.1. Discussion

Our approach allows reasoning about value added under uncertainty, facilitated by the use of reinforcement learning to profile the fitness values of options rather than the traditional (static) predictions of design-time decisions. The exponential decay factor provides architects with a visual demonstration of the benefit of options over time and aids them in complementing design-time decisions (Fig. 8). Our approach also alerts architects of significant detrimental changes in the benefit of the option being employed, and highlights which of the candidate options provides the optimal cost-benefit trade-off when changes occur for normal and strict constraints (Fig. 9). This could assist architects in the process of eliminating options with poor balance between benefits and costs over time. For instance, in the best and worst case scenarios in the experiment addressing *RQ2*, we found that one out of six options could potentially be eliminated because it was always worse than the others (i.e. the cloud-based option). In summary, the results shown in Section 6 confirm the effectiveness of our approach, demonstrating that self-adaptive systems can benefit from our approach, even though our approach itself is an architecture evaluation approach, and not a self-adaptive system.

There is a trade-off between monitoring the architecture options at every timestep and every T' intervals; our approach was able to show this trade-off to the architect (Figs. 12 and 13). Our approach is also robust when dealing with noise (Fig. 14) and scalable to be applied in practical settings (Fig. 11). Further, the exponential decay factor that we use weights recent values more heavily, which yields more practical results (Fig. 15). The approach uses the confidence interval to automatically tune the sensitivity of the approach to changes, which improves the system's stability (Fig. 16). In sum, our approach allows the architect to determine the added value of embedding diversification in the architecture (Fig. 17).

Architecture design decisions could be *static* or *dynamic* in nature. Several structural design decisions are static in nature; this implies that these decisions can be expensive to change and cannot be altered very frequently at run-time. Henceforth, the architect should evaluate them cautiously at design-time. Example of these decisions include network-related decisions such as the physical connectivity between devices (e.g. how data bits are moving in/out of the IoT device), logical connectivity (e.g. what protocols the software uses to transport these bits, such as MQTT), and also the network topology. These decisions are affected by the expected incoming data volumes, cost, memory requirements, *etc*. Therefore, they are quite difficult to change.

However, there are other decisions, which are dynamic in nature and could be customized at run-time (e.g., predefined decisions that could be tailored to fit the run-time context; strategies and tactics to address behavioural requirements). For instance, different deployment strategies, such as the use of cloud, fog-cloud, *etc*, are an example of a decision that can be best evaluated dynamically. When deemed to be necessary, diversification was also employed to provide "malleability" to alter the structure through inclusion of limited number of tactics that can better meet the behavioural requirements. In this context, our work is particularly interested in investigating and evaluating dynamic design decisions.

We have found that the pricing in most of IoT service providers (e.g. amazon) is based on the following AWS (2018): (1) Fixed charge to reserve the required resources for a particular time; (2) Pay-per-use, where the consumer pays for CPU per hour or per GB/TB of data; and (3) Auction-based, where a consumer could book resources if s/he pays the highest price for these resources. For these types of pricing methodologies, a noticeable reduction in the price volatility of resources has begun from December 2017 (Opel, 2018). In this context, when the price volatility is so low, the use of exponential decay factor for cost is not necessary. Therefore, we focused on providing a continuous measure for the benefit of each *dao* (i.e. exponential benefit). We plan to investigate cases where price volatility might be higher (using exponential decay for the cost) as future work.

Models@run.time may benefit from our approach to evaluate abstract reference models and their run-time instantiations. Further, architects of self-adaptive systems can use our method to systematically evaluate the adaptive design decisions, justify their inclusion, and model their potential behavior at run-time and before the system is deployed in the next release cycle. It can also value and profile the overall behavior and cost-benefit of these decisions over time as the software evolves, which could aid in determining which options could be best deployed at run-time. Cloud-based architectures, which are essentially based on service-oriented architectures, can use our approach to justify the choice of abstract architecture model and its possible concrete instantiations over different releases. Inputs from the evaluation can help architects in refining the abstract model; adjust, limit or rethink modes for dynamic composition of concrete ones prior to future deployments.

This approach is also generic so it could be integrated with any of existing tools for better informed evaluation. For instance, an application built using AWS IoT and Greengrass suites (AWS, 2018) could benefit from our approach in the context of evaluating the benefit of each option at run-time using the CloudWatch monitoring tool. It could aid the architect on deciding and planning which *dao* to currently use. The actual execution of the chosen *dao* is outside the scope of our approach. Instead, our contribution is in pro-

viding a framework for systematic support, informing the design and evaluation of IoT architectures.

The current approach accumulates the run-time knowledge of the benefit of the architecture decisions. It then assumes that the best *dao* to switch to is the one which has recently been obtaining the most balanced cost-benefit. This is effective in some contexts, e.g., when the deployed architecture option is not violating the QoS constraints that much, or when its performance level is not significantly changing over time. However, in other contexts, this type of learning may suggest wrong decisions and recommend unnecessary switches due to the lack of knowledge about the future benefit of candidate architecture decisions. Taking into account the future potential of diversified architecture option may provide a more informative evaluation. This calls for extending the following run-time approach with additional machine learning algorithms to anticipate the benefit of architecture decisions over time. The validity of these algorithms and effectiveness of the decision-making process are subject for future work.

9.2. Threats to validity

Threats to Internal Validity are concerned with the impact of evaluation parameters on the proposed approach. For that, we have analyzed our approach with varying input parameters (e.g. the relative importance of present/past, the confidence interval, *etc*) to ensure acceptable accuracy and stability. On the contrary, the values of these parameters might vary depending on some characteristics (e.g., the environmental conditions and more complex dependencies between architecture decisions), which could be investigated for future research. Further, one of threats to internal validity that the real environment may differ from the simulated data due to uncertainties at run-time. Therefore, we have tested the sensitivity of approach to noise, to check how well our proposed approach can handle this issue. We have also demonstrated the applicability of our approach to monitor the environment at every T' intervals if the real-time evaluation was expensive and the architects did not wish to use simulation.

Additionally, we have built on the iFogSim's (Gupta et al., 2017) initial architecture and extended it as well. In particular, the alternative options were realized taking into consideration the available technological solutions in satisfying concerns of interest (Mahmud et al., 2018; Nahrstedt et al., 2016). The attempt was conducted by first author and reviewed independently by coauthors to mitigate risks that relate to bias in the design decisions inception and their evaluation. Additionally, the simulation component of the contribution has provided flexibility to experiment in ranges of value; i.e. eliminating experimentation bias in fixating the value.

Threats to external validity are linked to the run-time data synthesis and analysis used in the experiments. In particular, the notion of run-time evaluation is meant to contextualize dynamic and behavioural evaluation; such evaluation needs to be conducted by monitoring a running system; analyzing historic data from a running system or using a simulated environment that mimic the behaviour of the running system to perform stress and what-if analysis for the performance of the architecture design decisions under changing environment and/or extreme scenarios. The results of such evaluation are time-dependent in non-stationary environments as it is the norm for systems such as IoT. Evaluation that is based on simulation can still be considered as design-time if the evaluation is performed at design-time and before deployment. However, we also see potentials for the same simulated approach to work in parallel with the running system, with symbiotic feedback between the simulator and the running system to perform anticipatory evaluation of key design decisions and their possible variants based on the run-time context, which may be difficult without the aid of simulation. Additionally, there will always be a trade-off between using the simulators and physical IoT devices in experimentation and data generation. This is due to the high cost of the actual deployment of IoT devices as compared to simulators. However, some companies, such as Amazon, IBM, and Intel, are motivating the need for having IoT simulation instrumenting what-if test scenarios, typically used during the architecture analysis and refinement stages to evaluate the response and sensitivity of the architecture to these tests. To generalize, our approach can steer the evaluation process using simulated data; partially simulated data and/or using monitored run-time data. The simulation can be used to simulate what-if an option would be deployed. This is particularly useful if the architect would like to solicit an early assessment on the option, where the cost of deployment would be expensive and the outcome can not be verified with high confidence. Simulation can be also used to simulate the performance of some design decisions under worst and stress scenarios. Henceforth, simulation can be cost-effective strategy to first assess the performance and then adapt, if the option deems to be sensible. Further, transfer learning methodology (Pan et al., 2010) has been recently adopted in Jamshidi et al. (2017), where the QoS measurements are taken from a simulator and only a few samples are taken from the real system leading to much lower cost and faster learning. In this context, the approach could learn from both simulated and run-time data, which is subject for future work.

A major challenge in mobile crowdsensing is the resource constraint of the mobile devices. This can constrain the processing which can be done on the devices forming the *dao* in our approach. On the other hand, mobile devices that are not constrained in resources can have a high energy consumption to be able to perform on-device computations. High energy consumption is therefore another challenge to our approach. However, the new mobile devices are designed to handle processing tasks with the acceptable energy consumption.

The IoT environment is highly dynamic and characterized by hyper-connectivity, due to high refresh rates and continuous upgrades. More specifically, it is expected that nodes can leave and join; nodes can be subject to upgrades and replacements; some nodes could be replaced by inferior ones; some new nodes can share common characteristics with its predecessors, offering enhancements for some qualities. Though it would be difficult for the approach to predict all possible types of IoT devices and versioning that can be encountered in real settings, our approach assumes the evaluation of reference architectures for this setting, where commonalities and variabilities are analyzed as part of the diversification procedure for incepting and evaluating the diversified architecture options.

10. Related work

One definition of software architecture is “the set of principal design decisions made about the system” (Taylor et al., 2009). Architecture evaluation is a milestone in the decision-making process. Evaluation is intended to assess the extent to which the architecture design decisions meet the quality requirements and their trade-offs. The evaluation can aid in early identification and mitigation of design risks; the exercise is intended to save integration, testing and evolution costs (SEI, 2018). A common issue in architecture evaluation is the presence of uncertainty. In architecture evaluation and decision-making, uncertainty is the lack of complete knowledge about the outcomes of deploying the architecture options (Letier et al., 2014). For instance, the architects may be uncertain about the effect of a proposed architecture on benefit (e.g. performance, availability, etc) and cost. Uncertainty may also arise due to unpredictable situations in dynamic applications, such as IoT. For instance, sensor aging effects, varying internet connectivity

and mobility of sensors, fluctuations in QoS and so forth (Gubbi et al., 2013; Narendra and Misra, 2016; Online, 2017; Nahrstedt et al., 2016). To this extent, we need an overview of current architecture evaluation approaches and how do they manage uncertainty and other related aspects.

We first highlight on the methods which addressed architecture evaluation from the design-time perspective. The CBAM (Kazman et al., 2001) is a static method for the economic modeling of architecture decisions, which builds on the ATAM (Kazman et al., 2000). It determines the costs and benefits of different architecture candidates. There are other design-time methods, which use probabilistic distributions (e.g. Doyle, 2010; Meedeniya et al., 2011), mathematical models (e.g. Al-Naeem et al., 2005; Esfahani et al., 2013b), and options theory (e.g. Ozkaya et al., 2007; Sobhy et al., 2016) to select the optimal architecture at early stages of architecting under uncertainty. More specifically, they rely on expert judgment in the evaluation. Most of the classical design-time approaches do not support adaptivity and automation, whereas our approach takes inspiration from self-adaptive systems to support that. We acknowledge that these approaches are fundamental for constructing the "suitable" architecture. However, the increase in uncertainty and dynamicity in environments such as IoT calls for a systematic method to complement design-time evaluations.

The software engineering and architecture state-of-art and -practice have witnessed increased reliance on solutions that embrace uncertainty in operations through engineering self-adaptivity (De Lemos et al., 2013) and autonomic management (Kephart and Chess, 2003). In particular, some approaches adopt utility functions while others leverage machine learning approaches to improve the decision-making. We examine some representative approaches in light of our approach. Utility functions have been used at run-time for self-adaptive and self-managed systems (e.g. Cheng, 2008; Heaven et al., 2009; Cooray et al., 2013; Esfahani et al., 2013a; Ghezzi and Sharifloo, 2013; Esfahani et al., 2011; Fredericks et al., 2014). For example, Heaven et al. (2009) reported on an approach tailored for self-managed software systems. The approach provides the following features: high-level task planning, architectural configuration and reconfiguration, and component-based control. The approach uses weighted utility functions to represent the quality attributes and determine the total utility of configurations by taking into account reliability and performance concerns. Esfahani et al. (2011) proposed an approach that elicits from stakeholders their belief towards uncertainty with respect to quality attributes, such as network bandwidth. In particular, the stakeholders provide an estimate for the range of uncertainty with respect to the expected level of input variation. The approach also quantifies the uncertainty through profiling by comparing the actual values with estimates from stakeholders and hence provide probability distributions for the variation in data collection. After that the overall uncertainty is computed using fuzzy math. Ghezzi and Sharifloo (2013)'s method is one of the few methods which complement design-time with run-time analysis. At design-time, the approach integrates goal-refinement methodologies with Discrete Time Markov Chain to determine all possible execution paths to the goal. At run-time, it exploits utility functions to measure the overall utility of paths, which are based on assumptions. For example, the utility for a 5ms response time is 1 and so forth. After that the hill climbing algorithm is used to search for the optimal goal. Cooray et al. (2013) proposed a proactive model-driven approach, which continuously updates the reliability predictions in response to environmental changes. The approach has proved its efficiency in adapting the system before it experiences a significant performance drop. However, the approach does not encounter cost and suffers from scalability issues. Generally, the major problem of the prior approaches is Krupitzer et al. (2015): (i) the high reliance on stakeholders for utility estimations, which is also subject to their experience; (ii) the utility functions are hard to be defined; (iii) there is complexity and uncertainty in the quantification of utility values.

Machine Learning approaches in the context of self-adaptive architectures (De Lemos et al., 2013) have been explored in Sabhnani and Serpen (2003), Tesauro et al. (2006), Tesauro (2007), Kim and Park (2009), Zhao and Hu (2011), Bennaceur et al. (2012) and Esfahani et al. (2013a) which encounter the observations of the system properties over time. In the context of using reinforcement learning techniques, Tesauro et al. (2006) snd Tesauro (2007) integrated queuing policies with reinforcement learning, forming a hybrid approach to enhance the dynamic resource-allocation decision-making process in data centers. The approach suffers from scalability and performance overheads. A reinforcement online learning planning technique was used by Kim and Park (2009) to improve robot's practices with respect to changes in the environment, by dynamically discovering the appropriate adaptation plans. However, it does not continuously evaluate the cost-effectiveness of architecture decisions over time. These approaches (Tesauro et al., 2006; Kim and Park, 2009) along with Sabhnani and Serpen (2003); Zhao and Hu (2011) tend to be domain-specific. FUSION (Esfahani et al., 2013a) is another learning-driven approach that adopts machine learning algorithm named Model Trees Learning (MTL) to tune the adaptation logic towards unpredictable triggers, rather than using static analytical models. It also uses utility functions to determine the benefit of models in question. The major pro of FUSION is its ability to learn over time and improve the adaptation actions due to the promising learning accuracy. However, FUSION has the following limitations: (i) it is specifically tailored to feature modelling; and (ii) it only detects goal violation, i.e. constraints, but does not have the ability to check if current architecture option is getting worse. Recently, further improvement on MOP for self-adaptive systems has been made in FEMOSAA (Chen et al., 2018), where the knee point strategy is adopted to select the optimal architecture option rather than using a weighted aggregation as most of the self-adaptive systems do. Though our work also leverages knee point strategy as FEMOSAA, our work adopts exponential decay functions to trigger adaptation only when there are significant changes, whereas FEMOSAA always adapts at every timestep (i.e. it suffers from instability).

To this extent, there is no systematic architecture evaluation method for intertwining the design-time with run-time. In this context, our proposed approach can complement some of the prior approaches and aid the architect in assessing the architecture design decisions under uncertainty through the following: (i) evaluating, complementing, and refining design-time decisions with run-time ones; (ii) tuning the relative importance of present/past of data for knowledge accumulation about an architecture which can aid in learning the behavior of architecture decisions over time; (iii) efficiently assessing the value of architecture decisions at different monitoring intervals; (iv) allowing the architect to tune the sensitivity of the approach to changes, and therefore how stable/unstable it will be over time.

11. Conclusion

In this paper, we proposed a novel run-time architecture evaluation method suited for systems that exhibit uncertainty and dynamism in their operation, such as IoT. This method provides continuous assessment of design-time decisions. It can inform deployment, refinement and/or phasing out decisions. Specifically, we used strategies derived from machine learning and cost-benefit analysis at run-time to continuously profile and evaluate the architecture decisions for their added value. We demonstrated the

use and significance of our approach by applying it to the case of designing diversification in an urban traffic monitoring IoT application to cater for the uncertainty in meeting quality requirements. Moreover, we evaluated our run-time approach with respect to baseline design-time approaches. Based on our experimental evaluation, our method could assist architects in evaluating design-time decisions at run-time, which could improve their decision-making process.

This paper has studied a representative web-operated IoT; nevertheless, the application of the method to other dynamic and variant-intensive systems can benefit from the approach. In our future work, we plan to explore how various learning techniques can be "orchestrated" at run-time to better support the evaluation. In particular, we aim to forecast the future potentials of architecture options and demonstrate its impact on the state of run-time architecture evaluation. The change detection method, being used in this paper, has been chosen because it is one of the most widely used methods in the concept drift detection literature. Future work could further investigate other change detection methods (i.e. the tests introduced in Section 2.3). Another potential direction is the further use of simulator along with the running system. This could show the extra insights the approach can provide in a run-time context. More specifically, we aim to demonstrate the usefulness of transfer learning methodology on the run-time architecture evaluation approach. In this paper, we focused on the QoS fluctuation in terms of benefit (e.g. response time and energy consumption). However, the exploration of the impact of high price volatility (i.e. using exponential decay for the cost) on the decision-making would be investigated as a future work. Finally, we also propose to extend the modeling of quality aggregation functions to consider dependencies between architecture decisions as future work.

Acknowledgments

The authors would like to thank Prof. Rajkumar Buyya and Redowan Mahmud for their valuable comments and helpful suggestions with respect to iFogsim tool.

Appendix A. Utility Trees

In this appendix, we illustrate the generated utility trees used in experiment related to RQ1.

References

Al-Naeem, T., Gorton, I., Babar, M.A., Rabhi, F., Benatallah, B., 2005. A quality-driven systematic approach for architecting distributed software applications. In: Proceedings of the 27th International Conference on Software Engineering. ACM, pp. 244–253.

Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., Meedeniya, I., 2013. Software architecture optimization methods: a systematic literature review. IEEE Trans. Softw. Eng. 39 (5), 658–683.

Arcuri, A., Briand, L., 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proceedings of the 33rd International Conference on Software Engineering (ICSE). IEEE, pp. 1–10.

Avizienis, A., Kelly, J.P., 1984. Fault tolerance by design diversity: concepts and experiments. Computer (8) 67–80.

Avižienis, A., Laprie, J.-C., Randell, B., Landwehr, C., 2004. Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Depend. Secure Comput. 1 (1), 11–33.

AWS, 2018. Deploy an end-to-end IOT application. https://aws.amazon.com/getting-started/projects/deploy-iot-application/services-costs/.

Balasubramanian, N., Balasubramanian, A., Venkataramani, A., 2009. Energy consumption in mobile phones: a measurement study and implications for network applications. In: Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement. ACM, pp. 280–293.

Baudry, B., Allier, S., Monperrus, M., 2014. Tailored source code transformations to synthesize computationally diverse program variants. In: Proceedings of the 2014 International Symposium on Software Testing and Analysis. ACM, pp. 149–159.

Baudry, B., Monperrus, M., Mony, C., Chauvel, F., Fleurey, F., Clarke, S., 2014. Diversify: Ecology-inspired software evolution for diversity emergence. In: Proceedings of the Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE). IEEE, pp. 395–398.

Bengtsson, P., Lassing, N., Bosch, J., van Vliet, H., 2004. Architecture-level modifiability analysis (alma). J. Syst. Softw. 69 (1), 129–147.

Bennaceur, A., Issarny, V., Sykes, D., Howar, F., Isberner, M., Steffen, B., Johansson, R., Moschitti, A., 2012. Machine learning for emergent middleware. In: Proceedings of the International Workshop on Eternal Systems. Springer, pp. 16–29.

Bittencourt, L.F., Diaz-Montes, J., Buyya, R., Rana, O.F., Parashar, M., 2017. Mobility-aware application scheduling in fog computing. IEEE Cloud Comput. 4 (2), 26–35.

Blair, G., Bencomo, N., France, R.B., 2009. Models@ run. time. Computer 42 (10), 22–27.

Brandão, L.E., Dyer, J.S., Hahn, W.J., 2005. Using binomial decision trees to solve real-option valuation problems. Decision Anal. 2 (2), 69–88.

Branke, J., Deb, K., Dierolf, H., Osswald, M., 2004. Finding knees in multi-objective optimization. In: Proceedings of the International Conference on Parallel Problem Solving from Nature. Springer, pp. 722–731.

Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A., Buyya, R., 2011. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Exper. 41 (1), 23–50.

Chen, T., Li, K., Bahsoon, R., Yao, X., 2018. Femosaa: feature-guided and knee-driven multi-objective optimization for self-adaptive software. ACM Trans. Softw. Eng. Methodol. 27 (2), 5:1–5:50. doi:10.1145/3204459.

Chen, Y., Kunz, T., 2016. Performance evaluation of iot protocols under a constrained wireless access network. In: Proceedings of the International Conference on Selected Topics in Mobile & Wireless Networking (MoWNeT). IEEE, pp. 1–7.

Cheng, S.-W., 2008. Rainbow: cost-effective software architecture-based self-adaptation. ProQuest.

Cooray, D., Kouroshfar, E., Malek, S., Roshandel, R., 2013. Proactive self-adaptation for improving the reliability of mission-critical, embedded, and mobile software. IEEE Trans. Softw. Eng. 39 (12), 1714–1735.

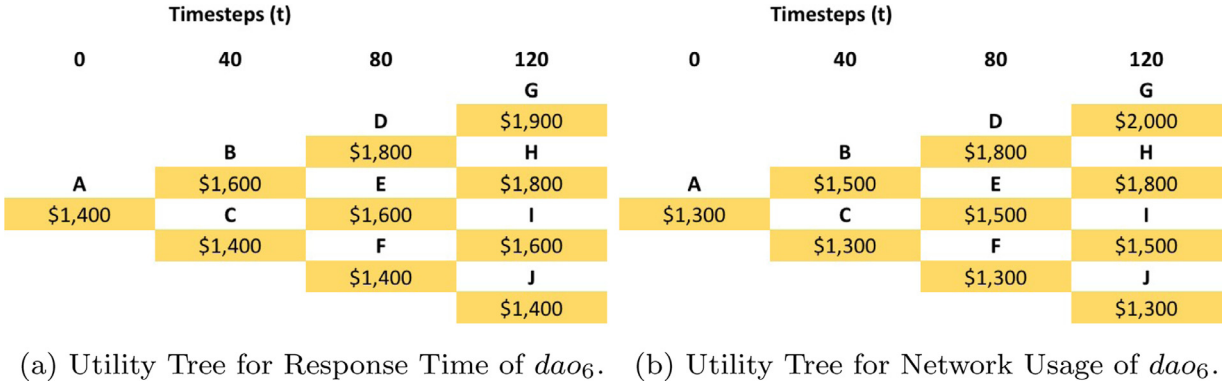


Fig. A.1. Utility Tree for dao_6 (Best Case Scenario). The root node corresponds to the utility of dao_6 at the first timestep and the final node corresponds to the utility of the 120th timestep, in monetary terms. Each node to the top/bottom right corresponds to a likely utility in case the quality of the *dao* improves/deteriorates after 40 timesteps. The probabilities of the utility going up or down over time are calculated based on expert's assumptions, and so are the utility values.

Cox, J.C., Ross, S.A., Rubinstein, M., 1979. Option pricing: a simplified approach. J. Financ. Econ. 7 (3), 229–263.
Da Xu, L., He, W., Li, S., 2014. Internet of things in industries: a survey. IEEE Trans. Ind. Inf. 10 (4), 2233–2243.
De Lemos, R., Giese, H., Müller, H.A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B., Tamura, G., Villegas, N.M., Vogel, T., et al., 2013. Software engineering for self-adaptive systems: A second research roadmap. In: Software Engineering for Self-Adaptive Systems II. Springer, pp. 1–32.
Deb, K., Gupta, S., 2011. Understanding knee points in Bicriteria problems and their implications as preferred solution principles. Eng. Optim. 43 (11), 1175–1204.
Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-ii. IEEE Trans. Evolut. Comput. 6 (2), 182–197.
Dejun, J., Pierre, G., Chi, C.-H., 2010. EC2 performance analysis for resource provisioning of service-oriented applications. In: Proceedings of the Workshops Service-Oriented Computing. ICSOC/ServiceWave. Springer, pp. 197–207.
Ditzler, G., Roveri, M., Alippi, C., Polikar, R., 2015. Learning in nonstationary environments: a survey. IEEE Comput. Intell. Mag. 10 (4), 12–25.
Doyle, G.S., 2010. A Methodology for Making Early Comparative Architecture Performance Evaluations. George Mason University.
Esfahani, N., Elkhodary, A., Malek, S., 2013. A learning-based framework for engineering feature-oriented self-adaptive software systems. IEEE Trans. Softw. Eng. 39 (11), 1467–1493.
Esfahani, N., Kouroshfar, E., Malek, S., 2011. Taming uncertainty in self-adaptive software. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering. ACM, pp. 234–244.
Esfahani, N., Malek, S., 2013. Uncertainty in self-adaptive software systems. In: Software Engineering for Self-Adaptive Systems II. Springer, pp. 214–238.
Esfahani, N., Malek, S., Razavi, K., 2013. Guidearch: guiding the exploration of architectural solution space under uncertainty. In: Proceedings of the 35th International Conference on Software Engineering (ICSE). IEEE, pp. 43–52.
Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are, 2016. Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are. Technical Report. Cisco Systems.
Fredericks, E.M., DeVries, B., Cheng, B.H., 2014. Towards run-time adaptation of test cases for self-adaptive systems in the face of uncertainty. In: Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self–Managing Systems. ACM, pp. 17–26.
Gama, J., Medas, P., Castillo, G., Rodrigues, P., 2004. Learning with drift detection. In: Advances in artificial intelligence–SBIA. Springer, pp. 286–295.
Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A., 2014. A survey on concept drift adaptation. ACM Comput. Surv. (CSUR) 46 (4), 44.
Ghezzi, C., Sharifloo, A.M., 2013. Dealing with non-functional requirements for adaptive systems via dynamic software product-lines. In: Software Engineering for Self-Adaptive Systems II. Springer, pp. 191–213.
Gokhale, S.S., 2007. Architecture-based software reliability analysis: overview and limitations. IEEE Trans. Depend. Secure Comput. 4 (1).
Gorbold, J., 2010. Power consumption, amd phenom ii x6 1090t be.
Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M., 2013. Internet of things (IoT): a vision, architectural elements, and future directions. Future Generat. Comput. Syst. 29 (7), 1645–1660.
Guérout, T., Monteil, T., Da Costa, G., Calheiros, R.N., Buyya, R., Alexandru, M., 2013. Energy-aware simulation with dvfs. Simul. Model. Pract. Theory 39, 76–91.
Gupta, H., Vahid Dastjerdi, A., Ghosh, S.K., Buyya, R., 2017. Ifogsim: a toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. Softw. Pract. Exper. 47 (9), 1275–1296.
Hachem, S., 2014. Service-Oriented middleware for the large-scale mobile Internet of Things. Université de Versailles-Saint Quentin en Yvelines Ph.D. thesis.
Hawthorne, M.J., Perry, D.E., 2004. Applying design diversity to aspects of system architectures and deployment configurations to enhance system dependability. In: Proceedings of the DSN 2004 Workshop on Architecting Dependable Systems,(DSN-WADS).
Heaven, W., Sykes, D., Magee, J., Kramer, J., 2009. A case study in goal-driven architectural adaptation. In: Software Engineering for Self-Adaptive Systems. Springer, pp. 109–127.
Holt, C.C., 2004. Forecasting seasonals and trends by exponentially weighted moving averages. Int. J. Forecast. 20 (1), 5–10.
rey Horn, J., Nafpliotis, N., Goldberg, D.E., 1994. A niched pareto genetic algorithm for multiobjective optimization. In: Proceedings of the first IEEE Conference on Evolutionary Computation, IEEE world congress on computational intelligence, 1. Citeseer, pp. 82–87.
Hull, J.C., 2006. Options, Futures, and Other Derivatives. Pearson Education India.
Issarny, V., Bouloukakis, G., Georgantas, N., Billet, B., 2016. Revisiting service-oriented architecture for the IoT: a middleware perspective. In: Proceedings of the International Conference on Service-Oriented Computing. Springer, pp. 3–17.
Jamshidi, P., Velez, M., Kästner, C., Siegmund, N., Kawthekar, P., 2017. Transfer learning for improving model predictions in highly configurable software. In: Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. IEEE Press, pp. 31–41.
Kazman, R., Asundi, J., Klein, M., 2001. Quantifying the costs and benefits of architectural decisions. In: Proceedings of the 23rd International Conference on Software Engineering. IEEE Computer Society, pp. 297–306.
Kazman, R., Haziyev, S., Yakuba, A., Tamburri, D.A., 2018. Managing energy consumption as an architectural quality attribute. IEEE Softw. 35 (5), 102–107.
Kazman, R., Klein, M., Clements, P., 2000. ATAM: Method for architecture evaluation. Technical Report. DTIC Document.
Kephart, J.O., Chess, D.M., 2003. The vision of autonomic computing. Computer 36 (1), 41–50.
Kim, D., Park, S., 2009. Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software. In: Proceedings of the ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. IEEE, pp. 76–85.
Krupitzer, C., Roth, F.M., VanSyckel, S., Schiele, G., Becker, C., 2015. A survey on engineering approaches for self-adaptive systems. Pervas. Mob. Comput. 17, 184–206.
Kuncheva, L.I., 2004. Classifier ensembles for changing environments. In: Proceedings of the International Workshop on Multiple Classifier Systems. Springer, pp. 1–15.
Letier, E., Stefan, D., Barr, E.T., 2014. Uncertainty, risk, and information value in software requirements and architecture. In: Proceedings of the 36th International Conference on Software Engineering. ACM, pp. 883–894.
Mahmud, R., Kotagiri, R., Buyya, R., 2018. Fog computing: A taxonomy, survey and future directions. In: Internet of Everything. Springer, pp. 103–130.
Meedeniya, I., Moser, I., Aleti, A., Grunske, L., 2011. Architecture-based reliability evaluation under uncertainty. In: Proceedings of the Joint ACM SIGSOFT conference–QoSA and ACM SIGSOFT Symposium–ISARCS on Quality of Software Architectures–QoSA and Architecting Critical Systems–ISARCS. ACM, pp. 85–94.
Nahrstedt, K., Li, H., Nguyen, P., Chang, S., Vu, L., 2016. Internet of mobile things: Mobility-driven challenges, designs and implementations. In: Proceedings of the IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI). IEEE, pp. 25–36.
Narendra, N., Misra, P., 2016. Research challenges in the internet of mobile things. IEEE IoT Newsletter. https://iot.ieee.org/newsletter/march-2016/research-challenges-in-the-internet-of-mobile-things.html, https://www.infoq.com/news/2018/01/amazon-ec2-spot-streamlining
Nord, R.L., Barbacci, M.R., Clements, P., Kazman, R., Klein, M., 2003. Integrating the Architecture Tradeoff Analysis Method (ATAM) with the cost benefit analysis method (CBAM). Technical Report. DTIC Document.
Online, [cited 12/8/2017]. [link]. 2017.
Opel, S., 2018. AWS streamlines amazon EC2 spot instance pricing model and operational complexity.
Ozkaya, I., Kazman, R., Klein, M., 2007. Quality-attribute based economic valuation of architectural patterns. In: Proceedings of the First International Workshop on the Economics of Software and Computation, 2007. ESC'07. IEEE. 5–5.
Pan, S.J., Yang, Q., et al., 2010. A survey on transfer learning. IEEE Trans. knowl. Data Eng. 22 (10), 1345–1359.
Ramírez, A., Parejo, J.A., Romero, J.R., Segura, S., Ruiz-Cortés, A., 2017. Evolutionary composition of QOS-aware web services: a many-objective perspective. Expert Syst. Appl. 72, 357–370.
Reporter, C., 2016. Research challenges in the internet of mobile things.
Sabhnani, M., Serpen, G., 2003. Application of machine learning algorithms to KDD intrusion detection dataset within misuse detection context.. In: Proceedings of the MLMTA, pp. 209–215.
SEI, S.E.I., 2018. Reduce Risk with Architecture Evaluation. Technical Report. SEI/CMU.
Sobhy, D., Bahsoon, R., Minku, L., Kazman, R., 2016. Diversifying software architecture for sustainability: a value-based perspective. Proceedings of the European Conference on Software Architecture (ECSA).
Sutton, R.S., Barto, A.G., 1998. Reinforcement Learning: An Introduction, 1. MIT press Cambridge.
Taylor, R.N., Medvidovic, N., Dashofy, E.M., 2009. Software Architecture: Foundations, Theory, and Practice. Wiley Publishing.
Tesauro, G., 2007. Reinforcement learning in autonomic computing: a manifesto and case studies. IEEE Int. Comput. 11 (1), 22–30.
Tesauro, G., Jong, N.K., Das, R., Bennani, M.N., 2006. A hybrid reinforcement learning approach to autonomic resource allocation. In: Proceedings of the IEEE International Conference on Autonomic Computing. IEEE, pp. 65–73.
Wagner, F., Ishikawa, F., Honiden, S., 2016. Robust service compositions with functional and location diversity. IEEE Trans. Serv. Comput. 9 (2), 277–290.
Wang, S., Minku, L.L., Yao, X., 2015. Resampling-based ensemble methods for online class imbalance learning. IEEE Trans. Knowl. Data Eng. 27 (5), 1356–1368.
Zeng, L., Benatallah, B., Ngu, A.H., Dumas, M., Kalagnanam, J., Chang, H., 2004. Qos-aware middleware for web services composition. IEEE Trans. Softw. Eng. 30 (5), 311–327.
Zhao, D., Hu, Z., 2011. Supervised adaptive dynamic programming based adaptive cruise control. In: Proceedings of the IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL). IEEE, pp. 318–323.
Zhu, X., Wu, X., 2004. Class noise vs. attribute noise: a quantitative study. Artif. Intell. Rev. 22 (3), 177–210.

Dalia M. Sobhy is currently working toward the Ph.D degree in the School of Computer, University of Birmingham. She received the MSc degree in Computer Engineering Department at the Arab Academy of Science and Technology and Maritime Transport (AASTMT), Alexandria, Egypt in 2009, and BSc degree from the same university, in 2014. She was also awarded gold medal as the top ranked candidate 1st in the graduating class. She held a teaching post in AASTMT. Dalia's main research interests are software architecture, economics-driven software engineering, machine learning for software engineering, search-based software engineering, and cloud and services software engineering. Her focus lies predominantly in the area of software architectures for dynamic environments, such as IoT. Her work has been published in internationally renowned conferences such as European Conference of Software Architectures (Springer). Among other roles, Dalia used to be a sub-

reviewer in IEEE International Conference on Services Computing; Women in Engineering (WIE) chair; and IEEE board member for AASTMT branch.

Dr. Leandro L. Minku is a Lecturer in Intelligent Systems at the School of Computer Science, University of Birmingham (UK). Prior to that, he was a Lecturer in Computer Science at the University of Leicester (UK) for three years, and a research fellow at the University of Birmingham (UK) for five years. He received the Ph.D degree in Computer Science from the University of Birmingham (UK) in 2010. Dr. Minku's main research interests are machine learning for software engineering, search-based software engineering, machine learning for non-stationary environments / data stream mining, and ensembles of learning machines. His work has been published in internationally renowned journals such as IEEE Transactions on Software Engineering, ACM Transactions on Software Engineering and Methodology, IEEE Transactions on Knowledge and Data Engineering, and IEEE Transactions on Neural Networks and Learning Systems. Among other roles, Dr. Minku is the general chair for the International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE 2019 and 2020), the co-chair for the Artifacts Evaluation Track at the International Conference on Software Engineering (ICSE 2020), an associate editor for the Journal of Systems and Software, an editorial board member for Neurocomputing and a conference correspondent for IEEE Software.

Dr. Rami Bahsoon is a Senior Lecturer (Associate Professor) at the School of Computer Science, University of Birmingham, UK. Bahsoon's research is in the area of software architecture, cloud and services software engineering, self-aware software architectures, self-adaptive and managed software engineering, economics-driven software engineering and technical debt management in software. He co-edited four books on Software Architecture, including Economics-Driven Software Architecture; Software Architecture for Big Data and the Cloud; Aligning Enterprise, System, and Software Architecture. He was a Visiting Scientist at the Software Engineering Institute (SEI), Carnegie Mellon University, USA (June-August 2018) and was the 2018 Melbourne School of Engineering (MSE) Visiting Fellow of The School of Computing and Information Systems, the University of Melbourne(August to Nov 2018). He holds a Ph.D in Software Engineering from University College London(2006) and was MBA Fellow in Technology at London Business School(2003–2005). He is a fellow of the Royal Society of Arts and Associate Editor of IEEE Software - Software Economies.

Dr. Tao Chen is currently a Lecturer in Computer Science at the Department of Computer Science, Loughborough University, UK. He is also an Honorary Research Fellow at the School of Computer Science, University of Birmingham, UK where he received his Ph.D. in 2016, on the topic of self-aware and self-adaptive cloud autoscaling systems. He has broad research interests on software engineering, including but not limited to performance engineering, search-based software engineering, self-adaptive software systems, services computing, cloud computing and computational intelligence. As the lead author, his work has been published in internationally renowned journals such as IEEE Transactions on Software Engineering, ACM Transactions on Software Engineering and Methodology, IEEE Transactions on Services Computing and ACM Computing Surveys. Among other roles, Dr. Chen regularly serves as a PC member for the IEEE International Congress on Internet of Things, IEEE World Congress on Services, and an associate editor for the Services Transactions on Internet of Things.

Rick Kazman is a Professor at the University of Hawaii and a Research Scientist at the Software Engineering Institute of Carnegie Mellon University. His primary research interests are software architecture, design and analysis tools, software visualization, and software engineering economics. Kazman has created several highly influential methods and tools for architecture analysis, including the SAAM (Software Architecture Analysis Method), the ATAM (Architecture Tradeoff Analysis Method), the CBAM (Cost-Benefit Analysis Method) and the Dali and Titan tools. He is the author of over 200 publications, and co-author of several books, including Software Architecture in Practice, Designing Software Architectures: A Practical Approach, Evaluating Software Architectures: Methods and Case Studies, and Ultra-Large-Scale Systems: The Software Challenge of the Future. His research has been cited over 20,000 times, according to Google Scholar.