To Adapt or Not to Adapt? Technical Debt and Learning Driven Self-Adaptation for Managing Runtime Performance

Tao Chen
Department of Computing and Technology, Nottingham Trent University, UK;
CERCIA, School of Computer Science, University of Birmingham, UK
t.chen@cs.bham.ac.uk

Rami Bahsoon, Shuo Wang
CERCIA, School of Computer Science, University of Birmingham, UK
{r.bahsoon,s.wang}@cs.bham.ac.uk

Xin Yao
Department of Computer Science and Engineering, Southern University of Science and Technology, China;
CERCIA, School of Computer Science, University of Birmingham, UK
x.yao@cs.bham.ac.uk

ABSTRACT

Self-adaptive system (SAS) can adapt itself to optimize various key performance indicators in response to the dynamics and uncertainty in environment. In this paper, we present Debt Learning Driven Adaptation (DLDA), an framework that dynamically determines when and whether to adapt the SAS at runtime. DLDA leverages the temporal adaptation debt, a notion derived from the technical debt metaphor, to quantify the time-varying money that the SAS carries in relation to its performance and Service Level Agreements. We designed a temporal net debt driven labeling to label whether it is economically healthier to adapt the SAS (or not) in a circumstance, based on which an online machine learning classifier learns the correlation, and then predicts whether to adapt under the future circumstances. We conducted comprehensive experiments to evaluate DLDA with two different planners, using 5 online machine learning classifiers, and in comparison to 4 state-of-the-art debt-oblivious triggering approaches. The results reveal the effectiveness and superiority of DLDA according to different metrics.

CCS CONCEPTS

• **Software and its engineering → Software performance**;

KEYWORDS

Self-adaptive systems, performance, technical debt, learning

ACM Reference Format:
Tao Chen, Rami Bahsoon, Shuo Wang, and Xin Yao. 2018. To Adapt or Not to Adapt? Technical Debt and Learning Driven Self-Adaptation for Managing Runtime Performance. In *ICPE '18: ACM/SPEC International Conference on Performance Engineering, April 9–13, 2018, Berlin, Germany.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3184407.3184413

1 INTRODUCTION

Self-adaptive system (SAS) is capable of planning and adapting itself at runtime, through a set of known control features (e.g., thread pool size and cache size, *etc*), to continually optimize for different key performance indicators, e.g., response time and energy consumption, under changing environment such as dynamic workload [16] [31]. SAS often operate under formally negotiated legal binding [33][18], e.g., Service Level Agreements (SLA) [3], especially in paradigms such as services and cloud computing. This binding allows us to translate the performance of SAS into a more intuitive monetary way, e.g., instead of saying the SAS's response time is 2s in average, we are able to state the SAS creates a total of $54 profit (or debt) for the owner. The real money that the SAS carries (either as profit or debt) determines its economic health.

While majority of SAS research has focused on the runtime ***planning*** phase of the SAS that determines *what and how to adapt* (e.g., rule-based [7], search-based [11][29][12] or control theoretic planners [32]), there is little research that explicitly tackles the challenge of *when and whether to adapt* the SAS, i.e., how to design the trigger [31]. We argue that deciding on when adaptation should be triggered is also non-trivial [31], because the effectiveness of the diverse planners can vary with the changing ***circumstances***, i.e., SAS's status and environment conditions. Even if we assume perfect planning, it still comes with cost, e.g., planning delay and extra resource/energy consumptions, etc. The key problem, which we address in this paper, is how to make a binary decision at each point in time: *whether to adapt* the SAS, considering dynamic and uncertain monetary cost-benefit of adapting the SAS or not.

Existing work on SAS falls into one of the two categories when dealing with the trigger: either adapt periodically [29][21] or adapt upon some observed or predicted events[1] (e.g., violation of requirement thresholds) at certain level of significance [18][13][36]. Adapting periodically is grounded on the principle that *we constantly adapt the SAS with the best possible adaptation solution, regardless whether the SAS breaks (e.g., violate performance requirements).* However, the problem with this method is obvious: since the adaptation may not significantly improve the performance under all circumstances, adapting when it is better not to adapt would generate unnecessary pressure, resulting additional costs and/or even degradation in performance, especially when the problem is difficult to solve, e.g., under heavy workload. Conversely, not adapting when it is needed would reduce the ability of the SAS to react to the changing environment. In contrast, adaptation upon the events relies on the principle that *if the SAS works (e.g., no requirements violation), do not change it; otherwise trigger adaptation.* Yet, adaptation upon

[1]The occurrence of event is indicated by the observation (or prediction) of the case when some fixed thresholds are hit.

ICPE '18, April 9–13, 2018, Berlin, Germany

ACM ISBN 978-1-4503-5095-2/18/04…$15.00
https://doi.org/10.1145/3184407.3184413

Table 1: The elements of technical debt and net debt in the context of software development and SAS

			Software Development **(asset is the software)**		***Running SAS*** **(asset is the SAS itself)**	
			If to improve software	*If not to improve software*	*If to adapt SAS*	*If not to adapt SAS*
Net Debt	*Technical Debt*	*Principal*	The case dependent cost for changing the software, e.g., extra money paid to employee for extra person/month.	N/A	The case dependent cost of adaptation, e.g., the rate per unit in SLA × the delay and extra resource/energy consumption of planning, etc.	N/A
		Interest	Cost, e.g., money paid for work, penalty of bad software quality, etc, incurred by old/new defects in the software as a result of wrongly spent efforts, flawed planning and bad code, etc.	Cost, e.g., penalty of bad software quality, etc, due to defects in the software.	Penalty (e.g., rate per unit from SLA × the units of violation) due to ineffective, flawed, sub-optimal or delayed planning and adaptation, etc, or too difficult environment, e.g., heavy and spiked workload.	Penalty (e.g., rate per unit from SLA × the units of violation) due to inability to react to the changing environment.
	Revenue		Bonus, e.g., more users are paying for the software, from the improved software as a result of wisely spent efforts, optimized code, etc.	Bonus, e.g., more users are paying for the software, from the software as a result of quick release, the expectation is met or exceeded, e.g., no defects reported.	Reward (e.g., reward per unit from SLA × the units above expectation) as a result of effective/optimized planning and adaptation.	Reward (e.g., reward per unit from SLA × the units above expectation) for performing as expected or better than requirements without adaptation.

the events may still cause extra pressure on the software system, providing little reward and/or worsening the performance, becuase the dynamic and uncertain cost-benefit of planning, in terms of real money, was not modelled explicitly.

In this paper, we propose the Debt Learning Driven Adaptation (DLDA), an automated framework that combines technical debt [15] and online learning [30] to determine when and whether to adapt a running SAS. The principle of DLDA is that *we adapt the SAS, if and only if, it can make the SAS economically healthier (less debt) than that of not adapting it.* Particularly, our contributions include: (i) We propose the *temporal adaptation debt* to quantify the ***net debt*** of SAS, which expresses the extent to which the SAS can repay its debt, if any, and create net profit from its decision (adapt or not). (ii) The labeling data is then used to train a binary and online classifier, which continuously classifies a re-emergent or unforeseen circumstances into the class label (i.e., to adapt or not) that can bring less debt, then inform the planner of SAS. (iii) DLDA is independent to the online learning classifier and planner for adaptation, in which DLDA also learns the effectiveness of a planner.

We evaluated DLDA on a complex SAS which contains the RUBiS [35] and a stack of software, under the FIFA98 trace [6] with different conditions, and in comparison to existing approaches. The results confirm the effectiveness and superiority of DLDA.

This paper is structured as follows. A high level mapping of technical debt analogy in SAS is presented in Section 2. Section 3 provides an overview of DLDA. The temporal adaptation debt and the related labeling are discussed in Section 4. Section 5 presents the combination of labeling and classification process. In Section 6, DLDA is extensively evaluated using a real-world SAS. Sections 7 and 8 discuss related work and conclude the paper, respectively.

2 THE TECHNICAL DEBT ANALOGY IN SAS

Technical debt for software engineering was coined by Cunningham [15], to help deciding whether to improve the software considering the costs and benefits of improvement versus that of not improving it. Like financial debt in the economic context, technical debt and its net value are associated with three elements:

- **Principal:** an one-off investment to an asset, e.g., a software.
- **Interest:** the extra cost of the asset accumulated over time.
- **Revenue:** the benefit of the asset accumulated over time.

While technical debt equals to $Principal + Interest$, its net value (***net debt***) is calculated as $Principal + Interest - Revenue$. Note that the net debt can be smaller than zero, i.e., it represents net profit. Those concepts in software development bear many similarities with the problem of *when and whether to adapt* in SAS, but with different meanings of asset, principal, interest and revenue. A high level mapping of the analogy to the contexts is shown in Table 1. In both contexts, the aim is to minimize the net debt.

Generally in the software development, the debt is calculated based on real money, e.g., the salary for employing engineers to do extra work and the monetary loss/profit generated by the software. In SAS context, the debt is viewed from the monetary terms of SAS and their interplay with the runtime performance. This can be achieved by extracting the monetary rate per unit from SLA, which is a formal legal binding negotiated between the software company and the end users before the SAS is deployed [33][18]. For example, suppose the SLA states that the rate for the cost of adaptation is \$0.345 per CPU second and an adaptation utilized 2s, then the principal would be \$0.69. Similarly, the SLA may contain a penalty rate of mean response time violation as \$0.043/s for a requirement of 2s, and if there is a mean response time of 2.5s for a period, then the penalty for it would be $(2.5 - 2) \times 0.043 = \$0.0215$. All those results can be combined to form the net debt, which represents the real money related to the SAS. The SLA negotiation can be achieved using many well-established methods form the literature [37][3], thus in this work, we assume that the SLA and its performance related elements have been instrumented before using DLDA.

```
<wsag:GuaranteeTerm Name="ResponseTime">
  <wsag:ServiceScope ServiceName="SAS"/>
  <wsag:QualifyingCondition>
      {"Function" : "AVG EVERY 120s"}
  </wsag:QualifyingCondition>
  <wsag:ServiceLevelObjective>
       <wsag:KPITarget>
          <wsag:KPIName>MeanTime</wsag:KPIName>
          <wsag:CustomServiceLevel>
              {"constraint" : "MeanTime LESS THAN 0.05s"}
           </wsag:CustomServiceLevel>
       </wsag:KPITarget>
  </wsag:ServiceLevelObjective>
  <wsag:BusinessValueList>
       <wsag:Penalty>
          <wsag:AssessmentInterval>
              <wsag:TimeInterval>120s</wsag:TimeInterval>
          </wsag:AssessmentInterval>
          <wsag:ValueUnit>USD_PER_SECOND</wsag:ValueUnit>
          <wsag:ValueExpression>3.5</wsag:ValueExpression>
       </wsag:Penalty>
       <wsag:Reward>
          <wsag:AssessmentInterval>
              <wsag:TimeInterval>120s</wsag:TimeInterval>
          </wsag:AssessmentInterval>
          <wsag:ValueUnit>USD_PER_SECOND</wsag:ValueUnit>
          <wsag:ValueExpression>3.5</wsag:ValueExpression>
       </wsag:Reward>
  </wsag:BusinessValueList>
</wsag:GuaranteeTerm>
```

```
<wsag:GuaranteeTerm
      Name="PlanningCPUTime">
   <wsag:ServiceScope
      ServiceName="SAS-Engine"/>
   <wsag:ServiceLevelObjective>
      <wsag:KPITarget>
         <wsag:KPIName>
            CPUTime
         </wsag:KPIName>
         <wsag:CustomServiceLevel>
            {"constraint" :
                  "CPUTime LESS THAN 0s"}
         </wsag:CustomServiceLevel>
      </wsag:KPITarget>
   </wsag:ServiceLevelObjective>
   <wsag:BusinessValueList>
      <wsag:Penalty>
         <wsag:AssessmentInterval>
            <wsag:Count>1</wsag:Count>
         </wsag:AssessmentInterval>
         <wsag:ValueUnit>
            USD_PER_SECOND
         </wsag:ValueUnit>
         <wsag:ValueExpression>
            0.01
         </wsag:ValueExpression>
      </wsag:Penalty>
   </wsag:BusinessValueList>
</wsag:GuaranteeTerm>
```

Figure 1: Example SLA fragment of a SAS

An example fragment of the possible SLA for SAS, derived from the well-known WS-Agreement [3], is shown in Figure 1.

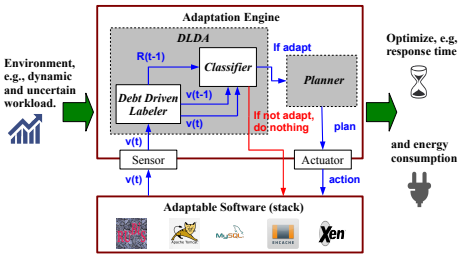


Figure 2: Overview of the DLDA framework on SAS.

3 DLDA OVERVIEW

As in Figure 2, a SAS generally has a feedback loop, with an adaptable software (e.g., a stack) that being managed at runtime, and an engine that controls the adaptation. DLDA runs in the adaptation engine and it has two components: *Debt Driven Labeler* and *Classifier*. While DLDA works within any feedback controller, it could be best placed in the *Analysis* phase of the MAPE-K loop [16].

The *Debt Driven Labeler* firstly analyzes the net debt for the past interval using the data vector from the most recent time points, i.e., $\mathbf{v}(t)$ and $\mathbf{v}(t-1)$, and the predefined SLA terms, it then produces a result, $R(t-1)$, labeling whether 'to adapt 'or 'not to adapt 'under the circumstance at time *t-1* can lead to less net debt (see Section 4).

In *Classifier*, the class label from the *Debt Driven Labeler*, together with the past status of SAS and the environmental factors in a vector $(\mathbf{v}(t-1)$, i.e., the circumstance), are used to train an online learning classifier (see Section 5). The decision of whether to adapt or not under the circumstance at the current point in time is then predicted by the updated classifier using the current vector of information, i.e., $\mathbf{v}(t)$. As such, DLDA can be used as an independent filter before any planner, which decides *what and how to adapt* [12][29].

4 TEMPORAL ADAPTATION DEBT MODEL

We propose the temporal adaptation debt to quantify the net debt for triggering the SAS at runtime. Like technical debt, adaptation debt equals to $Principal + Interest$, and its net debt is $Principal + Interest - Revenue$, i.e., how much money a SAS earns or costs.

Since the problem of *when and whether to adapt* SAS is a decision to be made at every point in time that could exhibit different circumstances (i.e., SAS status and environment), at the low level, temporal adaptation debt models the *newly incurred net debt*, including its one-off principal, accumulated interests and revenue *over a time interval*. This net debt expresses how the SAS performs, in terms of monetary value ($), over that time interval. The idea is that, if DLDA can predict whether adapt (or not) at each point in time can lead to less net debt, and react accordingly, then globally the net debt related to the SAS can be minimized. To this end, considering temporal notion is important as our purpose is to correlate the past circumstance of a given point in time to the class label (adapt or not) that can lead to less net debt, which in turn, will serve as a data sample to guide the learning classifier.

In the following, we transpose the high level notions from Table 1 into the low level, particularly in regards to the temporal notion.

4.1 Temporal Principal

At the low level, ***we use the temporal principal to describe the temporally invested cost of planning and adaptation at a unit of time***. Intuitively, to influence the SAS for the interval between time *t-1* and *t*, the principal of adaptation invested at time *t-1* is:

$$Principal(t-1) = C_{unit} \times U(t-1) \tag{1}$$

where $U(t-1)$ is the utilized units of certain adaptation effort (given by the engineers) measured at runtime, e.g., the delay of planning, the extra resource/energy consumption for planning, etc; and C_{unit} is the monetary rate per unit extracted from the SLA.

4.2 Subtracting Temporal Interest and Revenue

At the low level, the subtraction of temporal interest and revenue observed at time *t* is the subtraction of accumulated interest and revenue between *t-1* and *t*, representing the temporal result of two mutually exclusive cases: ***(i) the SAS did adapted at t-1; or (ii) the SAS did not adapt at t-1.*** Formally, the subtraction is:

$$S(t) = Interest(t) - Revenue(t) = \sum_{i=1}^{n} (\Delta Q_i \times M_i) \tag{2}$$

$$\Delta Q_i = \begin{cases} Q_i(t, t-1) - T_i & if\ minimize\ Q_i \\ T_i - Q_i(t, t-1) & if\ maximize\ Q_i \end{cases}$$

whereby $Q_i(t, t-1)$ is the given accumulated function, from the SLA, that returns the performance of the *i*th performance indicator that accumulated over the time interval between *t-1* and *t*, e.g., mean, total, maximum or definite integral function, etc. Such functions monitor the actual performance of SAS at runtime. T_i is the corresponding requirement constraint for the accumulated performance over a time interval from the SLA and *n* is the total number of indicators. M_i is the given monetary penalty (if violating requirement) or reward (if outperforming requirement) per unit for the related indicator over a time interval in the SLA . We assume that the reward and penalty share the same unit rate, but the formula can be easily changed to handle different rates. Note that we do not need to distinguish the interest and revenue, as what we care is the subtraction of their accumulated results, which is collectively reflected by the accumulated SAS performance.

4.3 Temporal Net Debt Driven Labeling

Suppose now we are at time *t*, the labeling process labels whether the SAS should adapt or not for the past circumstance at time *t-1* by comparing the net debt associated with "*to adapt*" and "*not to adapt*". Beside the formal discussion below, an intuitive illustration of the different cases in the labeling process is shown in Figure 3.

1) True Class: the temporal net debt for the class of '*the SAS should adapt under the circumstance at t-1*' , $D_{adapt}(t-1)$, is:

$$D_{adapt}(t-1) = \begin{cases} Principal(t-1) + S(t) & if\ adapted\ at\ t\text{-}1 \\ 0 & otherwise \end{cases} \tag{3}$$

Now, in practice, there are two further cases to consider:

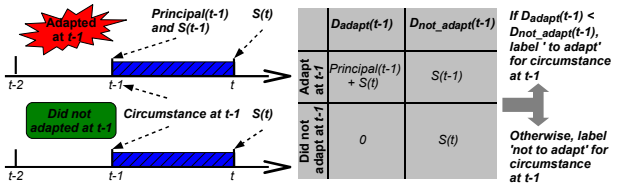


Figure 3: Different cases in the labeling process (now at *t*).

—**(i) the SAS did adapted at *t-1***. If there was indeed an adaptation at time *t-1*, $D_{adapt}(t-1)$ can be computed directly because the related $S(t)$, which represents the subtraction of accumulated interest and revenue between *t-1* and t caused by the adaptation made at *t-1*, is observable. In this case, adding the $S(t)$ immediately after adaptation to the invested principal is revealing, because adaptation may not lead to positive effects under some circumstances. If $S(t) < 0$, then $D_{adapt}(t-1)$ would be rewarded for the positive effects of adaptation on the performance of the SAS over the time interval. Conversely, if $S(t) > 0$, then $D_{adapt}(t-1)$ would be penalized as the adaptation causes marginal change or degradation on the SAS's performance over the interval.

—**(ii) the SAS did not adapted at *t-1***. If there was no adaptation at time *t-1*, $D_{adapt}(t-1)$ becomes incomputable as we cannot observe the $S(t)$ resulted from adaptation. Thus, we set $D_{adapt}(t-1) = 0$ since it is difficult to reason about the $S(t)$ related to an adaptation that has not been triggered.

2) False Class: the temporal net debt for the class '*the SAS should not adapt under the circumstance at t-1*', $D_{not\_adapt}(t-1)$, is:

$$D_{not\_adapt}(t-1) = \begin{cases} S(t-1) & \textit{if adapted at t-1} \\ S(t) & \textit{otherwise} \end{cases} \tag{4}$$

Again, in practice, there are two further cases to consider:

—**(i) the SAS did adapted at *t-1***. On contrary to $D_{adapt}(t-1)$, $D_{not\_adapt}(t-1)$ is only computable when there was no adaptation at time *t-1* as this is the only case that the related $S(t)$, which represents the subtraction of accumulated interest and revenue between *t-1* and t as a result of not adapting at *t-1*, is observable. if there was indeed an adaptation at time *t-1*, we assume that the accumulated performance of the indicators between *t-1* and t is similar to that between *t-2* and *t-1*, as a result of not adapting the SAS at *t-1*; in other words, we assume $D_{not\_adapt}(t-1) = S(t-1)$. This assumption is reasonable because the sampling interval of SAS can be tuned, as what we have done in this work, such that the local environment changes for two adjacent intervals are similar.

—**(ii) the SAS did not adapted at *t-1***. In this case, $D_{not\_adapt}(t-1)$ can be computed via $S(t)$ directly.

When the $D_{not\_adapt}(t-1)$ (or $D_{adapt}(t-1)$) is smaller than zero, it means that the SAS did not adapt (or the SAS adapted) at *t-1* creates net profit over the time interval.

Finally, we produce a class label $R(t-1)$, indicating whether it was economically healthier to adapt (or not to adapt) the SAS under the past circumstance at time *t-1*:

$$R(t-1) = \begin{cases} \textit{true, (to adapt)} & \textit{if } D_{adapt}(t-1) < D_{not\_adapt}(t-1) \\ \textit{false, (not to adapt)} & \textit{otherwise} \end{cases} \tag{5}$$

Overall, the adaptation debt model is able to quantify the temporal debt increment related to the decision of "*to adapt*" or "*not*", and can help to label which decision tends to have less debt on a past circumstance. Net debt is the most intuitive and important criteria for the practitioners of SAS, as it shows how much money the SAS earns or costs. Further, it fully exploits the domain knowledge embedded in the SLA and it is highly interpretable; this is the benefit of analytical model over other black-box ones, e.g., regression model, which ignore existing knowledge and is hard to understand.

5 DEBT AWARE LEARNING AND PREDICTION TO TRIGGER ADAPTATION

Next, to predict whether we should adapt the SAS at the current and possibly unforeseen circumstance, we feed the information of past circumstances, i.e., SAS's status and environment (as features), together with their class labels from the *Debt Driven Labeler*, into an classifier (in *Classifier*) for learning the correlation between the circumstance and the class (to adapt or not) that leads to less net debt. As such, given the current unforeseen circumstance, the classifier decides whether to adapt in favor of less net debt.

5.1 Features of Circumstance As Training Data

Features represent the characteristics of a circumstance. Note that these features should not be confused with the functionality of software; they are quantifiable properties of the SAS in machine learning. Here, we have used the status of SAS (i.e., control features and requirement features) and environmental features as training data to describe the circumstance when training the classifier:

Control Features: This refers to different control knobs that can be adjusted to affect the SAS, e.g, number of threads.

Environmental Features: This refers to the uncontrollable yet important stimuli that cause dynamics and uncertainties. Examples include the workload, order of requests, size of incoming jobs, etc.

Requirement Features: This calculates the extents to which a requirement is violated or its satisfaction is outperformed for each performance indicator i.e., $-\Delta Q_i$ in Eq. (2).

5.2 Online Machine Learning Classifiers

In DLDA, we train the classifiers following standard online learning paradigm [26]: instead of completely retraining a classifier when a new sample becomes available, we update the existing classifier with the new sample, after which the sample is discarded. Online learning is particularly fit for SAS: it eliminates the need to store data samples and significantly shortens training time without much degradation on accuracy [30]. In this work, we perform updates for every new sample, i.e., only one sample to learn each time. It is worth noting that Eq. (2) has aggregated all performance indicators into a single formula, thereby the classification is only concerned with a binary decision based on that formula, which is scalable and SAS agnostic. This design, together with the fact that only one sample to learn for the classifier, has provided wide applicability and great efficiency for the classifier to make decision at SAS runtime.

As for initial training, the classifier can be trained at design time using any readily available data, or it can be directly constructed at runtime. In both cases, it will gradually improve its accuracy using the most up-to-date data. This follows the standard online learning approach [30]. Specifically, since DLDA works with a wide range of classifiers, in this work, we have combined our temporal net debt driven labeling and 5 widely used classifiers from the literature with setups tailored to our subject SAS (see Table 2).

Table 2: The studied online learning classifiers

Classifier	***Setting***
Hoeffding Tree (HT) [17]	N/A
Naive Bayes (NB) [24]	N/A
Stochastic Gradient Descent (SGD) [9]	N/A
k-Nearest Neighbors (kNN) [1]	$k = 3$
Multi-Layer Perceptron (MLP) [23]	Sigmoid function and 3 layers

Table 3: The inputs and SLA terms for the subject SAS

Input	*Setting*	*Description*
Control features	N/A	10 control knobs, e.g., $maxThread$ and $Memory$ *etc.*
Environmental features	N/A	The workload (number of requests) for each of the 26 services in RUBiS (i.e., 26 environmental features) under the FIFA98 trace [6].
Performance indicators and functions in Eq. (2)	N/A	Functions and sensors that return accumulated mean response time and energy consumption between time *t* and *t-1*. The response time is the time between a request and the response [11] while energy consumption is measured by PowerAPI [10].
Adaptation effort	N/A	The measured utilized CPU time of planning. This can be replaced by other types of effort, e.g., energy used by planning, etc.
C_{unit} in Eq. (1), from the SLA	\$0.01	Monetary rate per second of the CPU time utilized by planning.
T_1 in Eq. (2), from the SLA	0.05s	Requirement of mean response time of an interval.
M_1 in Eq. (2), from the SLA	\$3.5	Monetary rate of penalty/reward per second differences between mean response time and T_1.
T_2 in Eq. (2), from the SLA	5watt	Requirement of mean energy consumption of an interval.
M_2 in Eq. (2), from the SLA	\$0.5	Monetary rate of penalty/reward per watt differences between mean energy consumption and T_2.

5.3 Training and Prediction Procedure

As shown in Figure 4, at time *t*, once the temporal net debt driven labeling is completed (step 1-2), we use the vector of features measured at time *t-1*, e.g., $\mathbf{F}(t-1) = \langle Workload\_of\_search = 19\ req/s, cacheMode = off, ...\rangle$, as inputs and the class label $R(t-1)$ (via (5) from the *Debt Driven Labeler*) as output to update the classifier (step 3-4). Therefore, the class label is reasoned and corrected by the labeling process in favor of less debt. While a deep discussion of training classifiers online is beyond the scope of this paper, interested readers can refer to [26][30] for details. Next, the vector of features at the current time *t*, e.g., $\mathbf{F}(t) = \langle Workload\_of\_search = 54\ req/s, cacheMode = off, ...\rangle$, are entered into the classifier for prediction—the classifier outputs a decision as to adapt or not (step 5-6). Following online learning paradigm, it is easy to see that the classifier predicts once it is updated by the new data. The classifier is reinforced and thus it can be continually consolidated.

6 EXPERIMENTAL EVALUATION

We run comprehensive experiments to evaluate DLDA variants with all classifiers (or simply called DLDA) and to compare them with state-of-the-art triggering approaches under different metrics.[2]

1) Experiments Settings and Verifiability: The subject SAS has a complex software stack that contains RUBiS [35], which is a well-known software benchmark for SAS, and a set of real-world software including Tomcat [19], MySQL [14] and Ehcache [20] running on an adaptable guest virtual machine. To emulate a realistic workload within the capacity of our testbed, we vary the number of clients according to the compressed FIFA98 workload [6] (from June to July), which can dynamically generate up to 600 parallel read-write requests. The SAS provides 10 important control features that influence its performance[3], which are complex since the variability of SAS is around 1.3×10^{16} alternatives. The SAS would adapt those control features, as the workload changes, to optimize for its response time and energy consumption.

To separate the adaptation engine and the adaptable software, we used Xen [27] to create a virtualized environment on a dedicated server. We have implemented DLDA using Java, and it is deployed on the Dom0 of Xen. The SLA terms of experiments are given in Table 3, which are fair settings tailored to fit with the subject SAS. In Section 6.4, we will discuss the critical parameters of DLDA.

To evaluate the generality of DLDA, we use it with two planners from the literature: one is the Multi-Objective Optimizer (MOO) that exploits pareto-dominance based, keen-point driven optimization to SAS at runtime [12][28][11]; the other relies on an equally weighted Single Objective Optimizer (SOO), in which we use equal weights to aggregate all objectives [29][21]. The objective functions are created using ensemble learning [11]. These planners are chosen as they are widely-adopted and capable to make effective, black-box planning under highly-variable SAS as the one we consider. Under each planner, we run DLDA with each of the 5 classifiers mentioned in Section 5 using their implementations in WEKA [22] and MOA [8]; we have used default settings unless otherwise stated. For all experiments, the sampling interval of the SAS is 120s for a total of 102 time points, which leads to around 5 system running hours per experiment run including end-users' thinking time.

2) Triggering Approaches in SAS: We compare DLDA with the following state-of-the-arts and debt-oblivious triggering approaches:

—**Event-driven (*Event*).** This is a typical category of approaches (e.g., in [18][4][7]) where adaptation is triggered upon certain event. In this work, we have used the SLA requirement violation as the event, which is the most commonly used setup (as in Table 3).

—**Prediction-based (*Pred*).** This category represents the work, e.g., in [5][36][2], that predicts the occurrence of an event, i.e., violation of performance requirement. The prediction results is then further analyzed by statistical inference; thus only the significant, reliable and persistent violations would trigger adaptation.

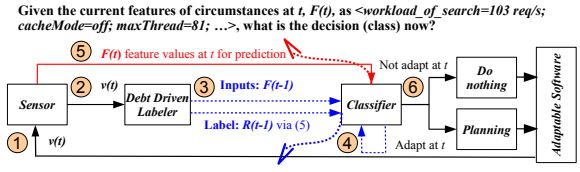
—**High-frequency (*High-f*).** This category represents the work (e.g., in [29][21]) that adapts the SAS based on high frequency, i.e., it triggers adaptation at every time point.

—**Low-frequency (*Low-f*).** This is similar to *High-f* but with low frequency, i.e., one adaptation every 10 time point.

—**Ground Truth (*GT*).** To determine whether it is indeed better to adapt (or not) at every time point under each planner, for each time point, we manually collected the decision (adapt or not) that leads to the smaller net debt by the end of the interval. Finally, the results of all those data points and their decisions together serve as an *approximate ground truth* in our evaluation.

[2] All code and data can be accessed at GitHub: https://github.com/taochen/ssase

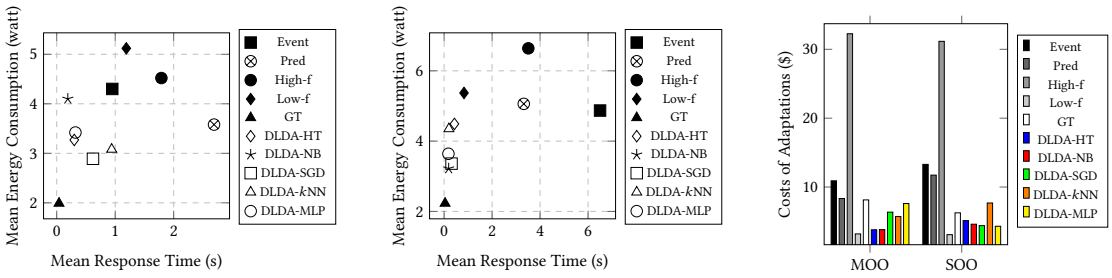
[3] The control features are, e.g., $maxThread$, $Memory$, etc. A complete specification can be found at https://github.com/taochen/ssase/blob/master/misc/DLDA-SAS.pdf



Example of Past Training Data Samples

Features of Circumstances		*Class Label*	
F(1)	*<Workload_of_search=78 req/s; cacheMode=off; maxThread=97; ...>*	*R(1)*	*True ("to adapt"): D_{adapt}(t-1) =\$3.1 < D_{not_adapt}(t-1)=\$7.2*
⋮	⋮	⋮	⋮
F(t-1)	*<Workload_of_search=18 req/s; cacheMode=Zipped; maxThread=53; ...>*	*R(t-1)*	*False ("not to adapt"): D_{adapt}(t-1) =\$1.5 > D_{not_adapt}(t-1)= -\$0.56*

Figure 4: Combining labeling and online classification.



(a) MOO Planner **(b) SOO Planner** **(c) Total Costs of Adaptations**

Figure 5: Comparing the overall performance and total costs of adaptations between DLDA and other state-of-the-art triggering approaches using MOO and SOO in the planning over 102 timesteps.

3) Metrics: The metrics we considered in the experiments are:

—**Accuracy.** The accuracy of the labels produced by labeling and the prediction against the actual classes from ground truth. In online learning, accuracy is often calculated as c/n: c is the number of correctly labeled/classified sample when there are n samples.

—**Performance.** The measured mean value of each performance indicator over time. While DLDA works with any quantifiable indicators, we used response time and energy consumption as the performance indicators for simplicity of exposition.

—**Total costs of adaptations.** This is the total principal in order to achieve the measured performance level of SAS. Recall from Section 4, the total cost is calculated as $\sum_{t=2}^{n} Principal(t-1) = \sum_{t=2}^{n} C_{unit} \times U(t-1)$. The more adaptations, the higher total cost.

—**Total net debt.** We report on the total net debt incurred throughout the experiment run, in which the temporal net debt between *t-1* and t is calculated as $Principal(t-1) + S(t)$ where $Principal(t-1) = 0$ if no adaptation at *t-1*; the total net debt is simply $\sum_{t=2}^{n} Principal(t-1) + S(t)$, as explained in Section 4.

—**SLA compliance.** The average values of performance of each indicator exceeding its SLA threshold over all intervals.

—**Overshoot.** The worst value of a performance indicator exceeding the SLA threshold during the transient.

—**Overhead.** The average training and prediction time of DLDA.

6.1 Accuracy

To evaluate accuracy, we compare the results of the labels produced by temporal net debt driven labeling and the predictions of classifiers against the actual classes in ground truth (see Section 6). As from Table 4, in general, both the labeling and prediction processes in DLDA exhibit high accuracy under both MOO and SOO planners. The results, range from 75% to 90%, clearly beat a random guess, which is likely to produce an accuracy around 50% for binary classification. Notably, DLDA-NB exhibits the best accuracy for both prediction and the labeling process that guides the learning.

We will examine, in contrast to the other triggering approaches, if the good accuracy can help DLDA to improve self-adaptation.

Table 4: The accuracy of DLDA with different classifiers against the ground truth over 102 timesteps (best is in bold)

	HT	NB	SGD	*k*NN	MLP
(MOO) Labeling	83%	**88%**	86%	87%	83%
(MOO) Prediction	**89%**	**89%**	81%	79%	77%
(SOO) Labeling	84%	**90%**	87%	80%	82%
(SOO) Prediction	81%	**88%**	84%	75%	**88%**

6.2 Performance and Adaptation Costs

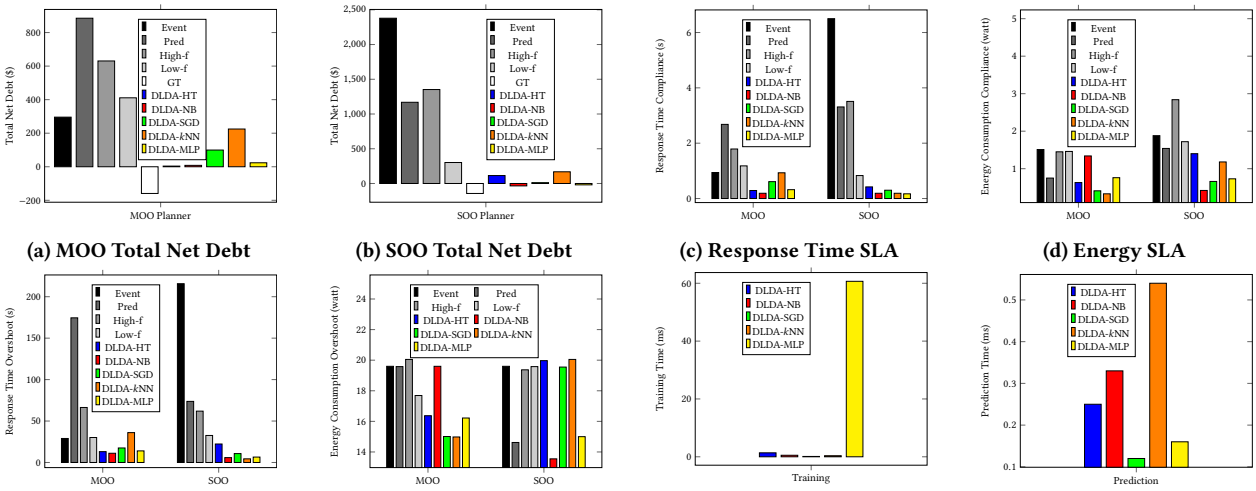
We now report on the mean performance values of all approaches over all time points, together with the total costs of adaptation. To validate statistical significance of the performance comparisons, we applied Wilcoxon Signed-Rank test (two-tailed) when comparing DLDA variants with the others. The results have confirmed statistical significance ($p < 0.05$) on at least one performance indicator with non-trivial effect sizes following the guidance in [25] .

As we can see from Figure 5a and 5b, under both planners, DLDA of all classifiers dominates *Event*, *Pred*, *High-f* and *Low-f* on both performance indicators. The only exception is that DLDA-NB tends to have slightly higher energy consumption than *Pred* when using MOO planner, which we have found to be statistically insignificant. As for the total costs of adaptations in Figure 5c, we note that DLDA with all classifiers and both planners achieve the superior performance by using remarkably smaller costs of adaptations, as when compared with *Event*, *Pred* and *High-f*. DLDA's costs however, as expected, is higher than that of *Low-f*. When comparing DLDA with the ground truth (*GT*) under both planners, there is still room for DLDA to improve on both performance indicators, but it is clearly closer to the performance of ground truth than the state-of-the-art triggering approaches. Further, its costs of adaptations (on all classifiers and planners) is similar to that of *GT*. These results also reveal that our temporal debt model is effective in guiding the classifiers, as DLDA significantly outperforms the others on performance and adaptation costs regardless the classifier used.

When comparing the DLDA variants on both planners from Figure 5, although DLDA-NB has higher energy consumption on the MOO planner, it generally has the best performance overall with relatively lower costs of adaptations, which can be attributed to the facts that it has the best prediction accuracy and the labeling process generates the most accurate labels to guide the classifier. However, the differences between DLDA variants are marginal.

6.3 Net Debt, SLA Compliance, Overshoot and Overhead

Next, we compare DLDA with the others on the total net debts, SLA compliance and overshoot. We have also illustrate the overhead of DLDA under different classifiers. The comparisons between DLDA and others under all metrics have been validated using Wilcoxon Signed-Rank test (two-tailed); the results have revealed statistical significance ($p < 0.05$) with non-trivial effect sizes.



(a) MOO Total Net Debt **(b) SOO Total Net Debt** **(c) Response Time SLA** **(d) Energy SLA**

(e) Response Time Overshoot **(f) Energy Overshoot** **(g) Training Overhead** **(h) Prediction Overhead**

Figure 6: Comparing total net debt, SLA compliance, overshoot and overhead over 102 timesteps.

Figures 6a and 6b show the total net debt of the approaches. We can see that, under both planners, DLDA with different classifier produces much less net debts than that of the state-of-the-art triggering approaches, and remarkably, up to three orders of magnitude less under the SOO planner. In particular, the total net debt of DLDA-NB and DLDA-MLP are smaller than zero under SOO planner, which means that they have created some net profit overall.

As shown in Figures 6c and 6d , under both planners, DLDA variants generally outperform state-of-the-art triggering approaches on the SLA compliance for both performance indicators. In particular, when comparing with the others, DLDA's improvements for the SLA compliance on response time is much greater than that of the energy consumption (DLDA-HT and DLDA-MLP is even worse than *Pred*). This is because the two performance indicators are conflicting, and DLDA has learned that favoring response time more would help to better reduce the total net debt, as evident in Figures 6a and 6b. Similarly, in Figures 6e and 6f, the overshoot of response time in DLDA is much smaller than that of the others. However, in general, its superiority on the overshoot of energy consumption is less obvious due to the same reason stated above.

The temporal net debt driven labeling in DLDA has negligible running time of less than 0.1ms and thus we focus on the training and prediction time required for the classification. As we can see in Figures 6g and 6h, most of the DLDA variants are very efficient in training, generating an overhead less than 3ms only. The only exception is DLDA-MLP, which requires 60ms, as the complex MLP needs more computation to converge to a good training error. In general, the high efficiency is enabled by the online learning paradigm where the data samples are learned one by one as they become available, thus the overhead is not sensitive to the size of training samples. As for prediction, DLDA has negligible overhead.

6.4 Discussion on the DLDA Parameters

As shown, DLDA can be affected by the settings of the requirement thresholds and rates per unit in the SLA. In this work, the settings in Table 3 are tuned w.r.t. to our testbed to create reasonable and fair comparisons. In particular, as in most of the practical scenarios, the requirement thresholds were reasonably tailored according to the SAS studied, i.e., they are neither too strong nor too relax. In contrast, the rates per unit on planning, reward and penalty in DLDA are more subjective, as they can be in any scales depending on the business purpose. Given an effective planner, those rates can influence the trade-off between adaptivity and stability of the SAS, i.e., increase the penalty rate implies more intensive adaptivity while increase the planning rate and/or reward rate favor stability.

In general, as mentioned in Section 2, these parameters can be tailored using many well-established methods form the literature [37][3] during the normal SLA negotiation process.

7 RELATED WORK

Existing work often fall into one of the two categories on designing trigger of SAS: either adapt periodically or adapt upon the events (e.g., violation of requirement thresholds). Adapting the SAS periodically has been the default method for many planning mechanisms from the literature. The PLATO [29] framework is one example that adapts the SAS on every point in time, within which it relies on genetic algorithm to search for the optimal (or near-optimal) adaptation solution. Other examples that rely on the same trigger include FEMOSAA [12] and VAIKYRIE [21], etc. These approaches often do not require predefined requirement thresholds, instead, they intend to optimize the SAS at every circumstances without considering net debt. In contrast, DLDA triggers adaptation only when it tends be economically healthier than not adapting.

Control theory is also another popular paradigm for engineering SAS [32]. However, most control theoretic approaches focus on the planning problem, i.e., *what and how to adapt*, and they adapt the SAS at predefined frequency of signaling cycle. In contrast, DLDA tackles explicitly *when and whether to adapt*, creating greater benefit over the others. Further, DLDA works with, e.g., rule-based [7], search-based [11][29][12] and control theoretic [32] planner, etc.

Event-based triggers are vast, e.g., Prometheus [4] and FUSION [18] are frameworks that trigger adaptation when they detect requirements violation. Other types of event also exists [7][34]: for example, Bencomo *et al.* [7] triggered adaptation based on the violation of

design *claim*, e.g., the claim of *Redundancy prevents networks partitions* is invalid if two or more network links fail simultaneously. Note that the utility functions in the defined events from above work is different from DLDA as they do not declare monetary value, i.e., there is no model about the profit/debt that the SAS generates.

While an event is often used in a reactive manner, proactive and event driven adaptation can be achieved by using limited prediction [36][5][2]. For example, Wang and Pazat [36] adapted the SAS when it is predicted that there is a violation of requirements, and such violation is indeed significant after it is verified by an online learning classifier. However, adaptations are still triggered by the detected/predicted occurrences of predefined events and it is not related to the monetary cost-benefit of adapting and not adapting.

8 CONCLUSION AND FUTURE WORK

This paper presents DLDA, a novel framework that combines technical debt and online learning, to determine *when and whether to adapt* the SAS at runtime. We proposed a temporal adaptation debt model to quantify the net debt for the decision of adapting and not adapting the SAS, based on which we design a temporal net debt driven labeling that labels whichever leads to less net debt for a given circumstance. By formulating the problem of *when and whether to adapt* as a binary classification problem, we combine the labeling process and online learning classifier in DLDA to determine whether to adapt or not upon unforeseen circumstances, in favor of reducing net debt. We conducted comprehensive evaluations on DLDA with 5 classifiers and in comparison to 4 state-of-the-art debt-oblivious triggering approaches. The results reveal that DLDA is effective and better than the other on various SAS metrics.

Our future work includes investigating the possibility of predicting for the long-term adaptation triggers, and how short-/long-term prediction could affect the trigger of adaptation. We also plan to apply DLDA on extreme domains of SAS, e.g., mobile environment.

ACKNOWLEDGMENT

This work is supported by the DAASE Programme Grant from the EPSRC (Grant No. EP/J017515/1).

REFERENCES

[1] Naomi S Altman. 1992. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician* 46, 3 (1992), 175–185.
[2] Ayman Amin, Alan Colman, and Lars Grunske. 2012. Statistical detection of qos violations based on cusum control charts. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*. ACM, 97–108.
[3] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. 2007. Web services agreement specification. In *Open grid forum*, Vol. 128. 216.
[4] Konstantinos Angelopoulos, Fatma Başak Aydemir, Paolo Giorgini, and John Mylopoulos. 2016. Solving the next adaptation problem with prometheus. In *Research Challenges in Information Science, International Conference on*. 1–10.
[5] Konstantinos Angelopoulos, Alessandro V Papadopoulos, Vítor E Silva Souza, and John Mylopoulos. 2016. Model predictive control for software systems with CobRA. In *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 35–46.
[6] Martin Arlitt and Tai Jin. 2000. A workload characterization study of the 1998 world cup web site. *IEEE network* 14, 3 (2000), 30–37.
[7] Nelly Bencomo, Amel Belaggoun, and Valerie Issarny. 2013. Dynamic Decision Networks for Decision-making in Self-adaptive Systems: A Case Study. In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 113–122.
[8] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. 2010. Moa: Massive online analysis. *Journal of Machine Learning Research* 11 (2010), 1601–1604.
[9] Léon Bottou. 1998. Online Algorithms and Stochastic Approximations. In *Online Learning and Neural Networks*, David Saad (Ed.). Cambridge University Press, Cambridge, UK. http://leon.bottou.org/papers/bottou-98x revised, oct 2012.
[10] Aurélien Bourdon, Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. 2013. PowerAPI: A Software Library to Monitor the Energy Consumed at the Process-Level. *ERCIM News* 92 (Jan. 2013), 43–44.
[11] Tao Chen and Rami Bahsoon. 2017. Self-Adaptive Trade-off Decision Making for Autoscaling Cloud-Based Services. *IEEE Transactions on Services Computing* 10, 4 (July 2017), 618–632.
[12] Tao Chen, Ke Li, Rami Bahsoon, and Xin Yao. 2018. FEMOSAA: Feature Guided and Knee Driven Multi-Objective Optimization for Self-Adaptive Software. *ACM Transactions on Software Engineering and Methodology* (2018). in press.
[13] Shang-Wen Cheng, Vahe V. Poladian, David Garlan, and Bradley Schmerl. 2009. Improving Architecture-Based Self-Adaptation Through Resource Prediction, in Software Engineering for Self-Adaptive Systems. Springer-Verlag, 71–88.
[14] Oracle Corporation. 1995. MySQL. https://www.mysql.com/. (1995).
[15] Ward Cunningham. 1993. The WyCash portfolio management system. *ACM SIGPLAN OOPS Messenger* 4, 2 (1993), 29–30.
[16] Rogério de Lemos et al. 2013. *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–32.
[17] Pedro Domingos and Geoff Hulten. 2000. Mining High-speed Data Streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '00)*. ACM, New York, NY, USA, 71–80.
[18] Naeem Esfahani, Ahmed Elkhodary, and Sam Malek. 2013. A learning-based framework for engineering feature-oriented self-adaptive software systems. *IEEE transactions on software engineering* 39, 11 (2013), 1467–1493.
[19] Apache Software Foundation. 1999. Tomcat. http://tomcat.apache.org/. (1999).
[20] Apache Software Foundation. 2003. Ehcache. http://www.ehcache.org/. (2003).
[21] Erik M. Fredericks. 2016. Automatically Hardening a Self-adaptive System Against Uncertainty. In *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. 16–27.
[22] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.* 11, 1 (Nov. 2009), 10–18.
[23] Simon S Haykin. 2001. *Neural networks: a comprehensive foundation*. Tsinghua University Press.
[24] George H. John and Pat Langley. 1995. Estimating Continuous Distributions in Bayesian Classifiers. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence (UAI'95)*. 338–345.
[25] Vigdis By Kampenes, Tore Dybå, Jo E Hannay, and Dag IK Sjøberg. 2007. A systematic review of effect size in software engineering experiments. *Information and Software Technology* 49, 11-12 (2007), 1073–1086.
[26] Leandro L Minku and Xin Yao. 2012. DDD: A new ensemble approach for dealing with concept drift. *IEEE transactions on knowledge and data engineering* 24, 4 (2012), 619–633.
[27] University of Cambridge Computer Laboratory. 2013. Xen: a virtual machine monitor. http://www.xenproject.org/. (2013).
[28] Gustavo G. Pascual, Roberto E. Lopez-Herrejon, MÃşnica Pinto, Lidia Fuentes, and Alexander Egyed. 2015. Applying multiobjective evolutionary algorithms to dynamic software product lines for reconfiguring mobile applications. *Journal of Systems and Software* 103 (2015), 392 – 411.
[29] Andres J. Ramirez, David B. Knoester, Betty H. C. Cheng, and Philip K. McKinley. 2011. Plato: a genetic algorithm approach to run-time reconfiguration inäautonomic computing systems. *Cluster Computing* 14, 3 (2011), 229–244.
[30] Jesse Read, Albert Bifet, Bernhard Pfahringer, and Geoff Holmes. 2012. Batch-incremental versus instance-incremental learning in dynamic and evolving data. In *International Symposium on Intelligent Data Analysis*. Springer, 313–323.
[31] Mazeiar Salehie and Ladan Tahvildari. 2009. Self-adaptive Software: Landscape and Research Challenges. *ACM Trans. Auton. Adapt. Syst.* 4, 2 (2009), 14:1–14:42.
[32] Stepan Shevtsov and Danny Weyns. 2016. Keep it SIMPLEX: Satisfying multiple goals with guarantees in control-based self-adaptive systems. In *Proceedings of the 24th International Symposium on Foundations of Software Engineering*. 229–241.
[33] James Skene, Franco Raimondi, and Wolfgang Emmerich. 2010. Service-level agreements for electronic services. *IEEE Transactions on Software Engineering* 36, 2 (2010), 288–304.
[34] C. Stier and A. Koziolek. 2016. Considering Transient Effects of Self-Adaptations in Model-Driven Performance Analyses. In *2016 12th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA)*. 80–89.
[35] Rice University. 2009. RUBiS. http://rubis.ow2.org/. (2009).
[36] Chen Wang and Jean-Louis Pazat. 2012. A Two-Phase Online Prediction Approach for Accurate and Timely Adaptation Decision. In *Proceedings of the 2012 IEEE Ninth International Conference on Services Computing*. 218–225.
[37] Farhana H Zulkernine and Patrick Martin. 2011. An adaptive and intelligent SLA negotiation system for web services. *IEEE Transactions on Services Computing* 4, 1 (2011), 31–43.