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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

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 events1 (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

1The occurrence of event is indicated by the observation (or prediction) of the case when some fixed thresholds are hit.
the events may still cause extra pressure on the software system, providing little reward and/or worsening the performance, because 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 of 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 RU-BiS [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 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.

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

<table>
<thead>
<tr>
<th>Software Development (asset is the software)</th>
<th>Running SAS (asset is the SAS itself)</th>
</tr>
</thead>
<tbody>
<tr>
<td>If to improve software</td>
<td>If not to improve software</td>
</tr>
<tr>
<td><strong>Principal</strong></td>
<td>The case dependent cost for changing the software, e.g., extra money paid to employee for extra person/month.</td>
</tr>
<tr>
<td><strong>Interest</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Revenue</strong></td>
<td>Bonus, e.g., more users are paying for the software, from the improved software as a result of wisely spent efforts, optimized code, etc.</td>
</tr>
</tbody>
</table>

![Figure 1: Example SLA fragment of a SAS](image-url)
Environment, e.g., dynamic and uncertain workload. Optimize, e.g., response time and energy consumption.

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., \(v(t)\) and \(v(t-1)\), 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 \((v(t-1), \text{i.e., the circumstance})\), 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., \(v(t)\). 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:

\[
\text{Principal}(t-1) = C_{unif} \times U(t-1)
\]

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 \(C_{unif}\) 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 adapt at \(t-1\); or (ii) the SAS did not adapt at \(t-1\). Formally, the subtraction is:

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

whereby \(\Delta Q_i(t, t-1)\) 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:

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

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

Figure 3: Different cases in the labeling process (now at \(t\)).
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>
<thead>
<tr>
<th>Classifier</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hoefding Tree (HT) [17]</td>
<td>N/A</td>
</tr>
<tr>
<td>Naive Bayes (NB) [24]</td>
<td>N/A</td>
</tr>
<tr>
<td>Stochastic Gradient Descent (SGD) [9]</td>
<td>N/A</td>
</tr>
<tr>
<td>k-Nearest Neighbors (KNN) [1]</td>
<td>(k = 3)</td>
</tr>
<tr>
<td>Multi-Layer Perceptron (MLP) [21]</td>
<td>Sigmoid function and 3 layers</td>
</tr>
</tbody>
</table>

Table 2: The studied online learning classifiers
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., $F(t-1) = \langle \text{Workload of search} = 19 \text{ req/s}, \text{cacheMode} = \text{off}, \ldots \rangle$, 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., $F(t) = \langle \text{Workload of search} = 54 \text{ req/s}, \text{cacheMode} = \text{off}, \ldots \rangle$, 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.5

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, which are complex since the variability of SAS is around $1.3 \times 10^{16}$ 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.

---

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

<table>
<thead>
<tr>
<th>Input</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control features</td>
<td>N/A</td>
<td>10 control knobs, e.g., maxThread and Memory etc.</td>
</tr>
<tr>
<td>Environmental features</td>
<td>N/A</td>
<td>The workload (number of requests) for each of the 26 services in RUBiS (i.e., 26 environmental features) under the FIFA98 trace [6].</td>
</tr>
<tr>
<td>Performance indicators and functions in Eq. (6)</td>
<td>N/A</td>
<td>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].</td>
</tr>
<tr>
<td>Adaptation effort</td>
<td>N/A</td>
<td>The measured utilized CPU time of planning. This can be replaced by other types of effort, e.g., energy used by planning, etc.</td>
</tr>
<tr>
<td>$c_{unit}$ in Eq. (1), from the SLA</td>
<td>$0.01$</td>
<td>Monetary rate of penalty/reward per second differences between mean response time and $t_1$.</td>
</tr>
<tr>
<td>$t_1$ in Eq. (2), from the SLA</td>
<td>$0.05s$</td>
<td>Requirement of mean response time of an interval.</td>
</tr>
<tr>
<td>$A_1$ in Eq. (2), from the SLA</td>
<td>$3.5$</td>
<td>Monetary rate of penalty/reward per second differences between mean response time and $t_1$.</td>
</tr>
<tr>
<td>$A_2$ in Eq. (2), from the SLA</td>
<td>$0.05$</td>
<td>Requirement of mean energy consumption of an interval.</td>
</tr>
<tr>
<td>$B_2$ in Eq. (2), from the SLA</td>
<td>$0.05$</td>
<td>Monetary rate of penalty/reward per wait differences between mean energy consumption and $T_f$.</td>
</tr>
</tbody>
</table>

---

**Figure 4: Combining labeling and online classification.**
Table 4: The accuracy of DLDA with different classifiers against the ground truth over 102 timesteps (best in bold)

<table>
<thead>
<tr>
<th></th>
<th>HT</th>
<th>NB</th>
<th>SGD</th>
<th>kNN</th>
<th>MLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>(MOO) Labeling</td>
<td>83%</td>
<td>88%</td>
<td>86%</td>
<td>87%</td>
<td>83%</td>
</tr>
<tr>
<td>(MOO) Prediction</td>
<td>89%</td>
<td>89%</td>
<td>81%</td>
<td>79%</td>
<td>77%</td>
</tr>
<tr>
<td>(SOO) Labeling</td>
<td>84%</td>
<td>90%</td>
<td>87%</td>
<td>80%</td>
<td>82%</td>
</tr>
<tr>
<td>(SOO) Prediction</td>
<td>81%</td>
<td>88%</td>
<td>84%</td>
<td>75%</td>
<td>88%</td>
</tr>
</tbody>
</table>
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 from 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][54]: 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 defined in the features extracted 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 Pazzani [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


