2020 IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)

SEAMS-2020 Best Student Paper Award

DATESSO: Self-Adapting Service Composition with Debt-Aware Two Levels Constraint Reasoning

Satish Kumar
School of Computer Science
University of Birmingham, UK
s.kumar.8@cs.bham.ac.uk

Tao Chen
Department of Computer Science
Loughborough University, UK
t.t.chen@lboro.ac.uk

Rami Bahsoon
School of Computer Science
University of Birmingham, UK
r.bahsoon@cs.bham.ac.uk

Rajkumar Buyya
School of Computing and Information Systems
University of Melbourne, Australia
rbuyya@unimelb.edu.au

ABSTRACT

The rapidly changing workload of service-based systems can easily cause under-/over-utilization on the component services, which can consequently affect the overall Quality of Service (QoS), such as latency. Self-adaptive services composition rectifies this problem, but poses several challenges: (i) the effectiveness of adaptation can deteriorate due to over-optimistic assumptions on the latency and utilization constraints, at both local and global levels; and (ii) the benefits brought by each composition plan is often short term and is not often designed for long-term benefits—a natural prerequisite for sustaining the system. To tackle these issues, we propose a two levels constraint reasoning framework for sustainable self-adaptive services composition, called DATESSO. In particular, DATESSO consists of a refined formulation that differentiates the 'strictness' for latency/utilization constraints in two levels. To strive for long-term benefits, DATESSO leverages the concept of technical debt and time-series prediction to model the utility contribution of the component services in the composition. The approach embeds a debt-aware two level constraint reasoning algorithm in DATESSO to improve the efficiency, effectiveness and sustainability of self-adaptive service composition. We evaluate DATESSO on a service-based system with real-world WS-DREAM dataset and comparing it with other state-of-the-art approaches. The results demonstrate the superiority of DATESSO over the others on the utilization, latency and running time whilst likely to be more sustainable.

CCS CONCEPTS

• **Software and its engineering** → **Software performance**; *Model-driven software engineering*.

KEYWORDS

Self-adaptive systems, service composition, technical debt, constraint reasoning, search-based software engineering

SEAMS '20, October 7–8, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7962-5/20/05…$15.00
https://doi.org/10.1145/3387939.3391604

ACM Reference Format:
Satish Kumar, Tao Chen, Rami Bahsoon, and Rajkumar Buyya. 2020. DATESSO: Self-Adapting Service Composition with Debt-Aware Two Levels Constraint Reasoning. In *IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '20), October 7–8, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3387939.3391604

1 INTRODUCTION

Service composition allows software to be built by seamlessly composing readily available service components, each of which offers different guarantee on Quality-of-Services (QoS), where latency can be of paramount importance [52]. Dynamically composing services is an enabling property for service-based systems supported by Cloud, Edge, Smart and Internet-of-Things environments. However, a known difficulty in service-based systems is the presence of rapidly changing workload, leading to under-/over-utilization on the services components [32]. On one hand, increasing workload can enhance the over-utilization of a services component within a composite service, which in turns, would negatively affect the latency and may violate the Service Level Agreement (SLA) [41] [32]. On the other hand, decreasing workload may lead to under-utilization of the capacity of component services, reducing the revenue that should have been achieved as the infrastructural resources also impose monetary cost. To address those issues, self-adaptation on service composition is promising, but the adaptation needs to be effective while being efficient and render benefits over time (i.e., sustainable).

When reasoning about self-adaptation for service composition, there are often two levels of latency/utilization constraints: the local constraint that relates to the individual constituent services and the global one for the entire service composition. Both of them are critical, as they can affect what the alternative composition plans to be searched during the adaptation [42]. However, existing work on self-adapting service composition often rely on over-optimistic assumptions, such that both local and global constraints are hard and can always be satisfied [6, 30, 34, 42, 48]. This can negatively influence the adaptation quality and efficiency, rendering lengthy reasoning process, especially when the given constraints are unrealistic/inappropriate. Further, the manifestation of strong assumptions may completely ignore the fact that certain composition plans may temporally violate the constraint, but are likely to create much larger benefits after a certain period of time.

Given the rapidly changing workload, it is important to ensure that each adaptation can be effective over a period of time and would avoid unnecessarily frequent adaptations. However, current work informing adaptation tends to render short-term benefits, i.e., the immediate improvement of a composition plan. These improvements, for example, can be in response to (predicted) latency constraint violation/undesired utilization [8, 29, 50]. Additionally, immediate low utilization/high latency in the short term may not necessarily mean an undesired composition plan; in fact, it can be the source that stimulates largely increased benefit in the long term. For example, under-utilization could be desirable temporarily in order to prepared for a largely increased workload in the long term. Similarly, over-utilization may be acceptable in short time, as long as the workload is only a 'spike' and the loss can be paid off by long-term benefits. As a result, despite that adapting with composition plan that has the best immediate improvement may lead to short-term advantages, it can easily create instability and hinder the possibility of achieving higher benefits in the long term.

To address the above challenges, we propose a framework that leverages **d**ebt-**a**ware **t**wo l**e**vels con**s**traint reasoning for **s**elf-adapting service c**o**mposition (hence called DATESSO). We show that DATESSO can achieve better utilization/latency in the long term while being faster than state-of-the-art approaches, providing more sustainable self-adaptive service-based systems. In a nutshell, the major contributions of this paper are summarized as follows:

- Instead of formalizing the constraints at both local and global levels as hard ones, we refine the global constraints as the soft ones. This has enabled us to tailor the reasoning process in self-adaptation and mitigate over-optimism.
- We propose temporal debt-aware utility, a new concept that extends from the technical debt metaphor, to model the long-term benefit contribution of possible component services that constitute to a composition plan.
- Drawing on the above, we design an efficient two level constraint reasoning algorithm in DATESSO that is debt-aware, and utilizes the different strictness of the two level constraints to reduce the search space.
- We evaluate DATESSO on a commonly used service-based system [23, 24, 33] whose component services are derived from the real-world WS-DREAM dataset [53] and under the FIFA98 workload trace [7]. The results show that, in contrast to state-of-the-art approaches [6] [29, 39], DATESSO achieves better utilization and latency while having smaller overhead, leading to more sustainable self-adaptation in service composition.

The remaining of the paper is organized as follows: Section 2 presents the background information of service composition, the constraints, technical debt and a running example of the issues. Section 3 shows an overview of DATESSO. Section 4 discusses our formalization of the two level constraints with different strictness. The temporal debt-aware utility model and the debt-aware two level reasoning algorithm are specified in Section 5 and 6, respectively. Then, we presents the experiment results in Section 7, following by discussion of threats to validity in Section 8. Section 9 compares DATESSO with existing work and Section 10 concludes the paper.

2 PRELIMINARIES

2.1 Self-Adaptation in Service Composition

A service composition is a special software form that consists of a particular workflow of connected abstract services, denoted as $\{a_1, a_2, ..., a_x\}$. Each of these abstract services can be realized by using a readily available component service selected from the Internet. Typically, there could be multiple component services to be selected, and the yth component service for the xth abstract service is denoted as c_{xy}. Therefore the possible component services for the xth abstract service form a set, denoted as $\{c_{x1}, c_{x2}, ...\}$, each of which has different generic latency guarantee on its capacity. For example, c_{xy} has a capacity to process 50 requests in 0.5 seconds.

In such a context, a SLA may be legally negotiated to ensure the performance of a service composition by contract. The most notable elements in the SLA are the constraints on the utilization of service capacity and the achieved latency level per request, which we will elaborate in the next section.

As the workload changes, at runtime, the goal of self-adaptation for service composition is to find the composition plan, $\{c_{11}, c_{23}, ..., c_{xy}\}$, that improves utilization and latency so that they satisfy all the constraints for as long as possible.

2.2 Constraints in Service Composition

In service composition, constraints denote the stakeholders' expectation of the latency guarantee. Most commonly, a SLA can define these constraints by specifying the bound of the latency and utilization [30]. For example, a service's latency should not exceed 10s or the utilization is at least 0.7. Typically, there are two levels of constraints:

- **Global constraint:** The global constraint specifies the minimum expectation of latency/utilization for the entire service composition. It is often the most common requirement in a service-based systems [39] [6].
- **Local constraint:** The local constraints are specified for the latency/utilization on each abstract service[1]. This is important, as each abstract services can be realized by the component service from different parties; any violation of the local constraint would in fact cause severe failure in the composition, leading to an outage [30] [6].

It is worth noting that, satisfying all local constraints does not necessarily mean that the global constraint can be satisfied, since each of the constraints is documented separately [48]

2.3 Technical Debt

Technical debt is a widely recognized metaphor in software development [3, 9, 46]. Its core idea is to describe the extra cost incurred by actions that compromise long-term benefits of the developed software, e.g., maintainability for short-term gains due to the need of timely software release.

[1] For latency, this constraint would be applied for each request.

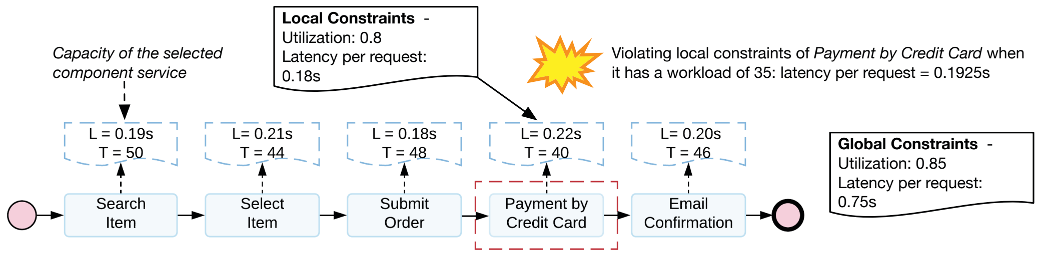

Figure 1: A running example of issues in service composition (*L* and *T* mean that the selected component service of an abstract service can process all *T* requests in *L* seconds)

The technical debt metaphor was initially introduced by Cunningham [28] in the context of agile software development, where the definition is described as:

> *"Shipping first-time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. The danger occurs when the debt is not repaid. Every minute spent on not quite right code counts as interest on that debt."*

In this regards, technical debt is often used in an economic-driven decision approach for communicating the technical trade-off between short-term advantages and long-term benefits in software projects [46]. In the context of service composition, the notion of technical debt can be perfectly aligned with the requirement of long-term benefits: each possible component service may associate with a debt with respect to constraint violation. Such a debt, once selected, may or may not be repaid over a period of time, depending on the actual workload.

2.4 Running Example

In this section, we present a simple example of service composition to explain the problems. As shown in Figure 1, there is a service composition in the form of sequentially connected abstract service, each of which has been realized by a particular component service. In this case, each selected component service has its own overall capacity, e.g., the selected component service for Search Item abstract service can process all 50 requests in 0.19 seconds.

As mentioned, each abstract service, along with the entire service composition, are legally documented with separated constraints on the utilization and latency per request, as specified in the SLA. Suppose that in this scenario, the local constraint of utilization and latency of each request for the abstract service Payment by Credit Card could be 0.8 and 0.18 seconds, respectively. Meanwhile, the global constraint of utilization and latency of each request for the service composition is 0.85 and 0.75 seconds, respectively. Given the changing workload, it is likely that either (or both) levels of constraint may be violated, which requires self-adaptation to replace the component services. However, there are two issues with this:

(1) In this context, the different constraints are negotiated independently to each others. While it is relatively easy to find the alternative component service that satisfy the local constraints, searching for the composition plan that satisfies the global constraints is difficult, or we may not know whether one exists. As a result, existing approaches that treats both levels of constraints as hard constraints suffers the issue of being over-optimistic: they may struggle to find a satisfactory composition plan, especially under a scenario where such a plan barely exists. Further, this would completely eliminate the composition plan that may cause temporary violation of the global constraint(s), but can create much larger long-term benefits.
(2) When self-adaptation is required, a possible component service and the entire composition plan may provide short-term immediate benefit in relieving constraint violation, but it is difficult to know whether such a benefit can be sustainable. In contrast, it is possible to temporally accept a composition plan that may still violate the global constraint(s), but will generate larger benefit in the long term. Therefore, self-adapting service composition without having any guarantee on the long term can lead to frequent adaptations with merely short-term benefits, which generate unnecessary overhead.

The DATESSO proposed in this work was designed to explicitly address these two issues in self-adapting service composition.

3 DATESSO OVERVIEW

Figure 2 illustrates the overview of DATESSO. As can bee seen, there are three key stages, namely *Formalization*, *Modeling* and *Reasoning*, each of which is specified as follows:

(1) *Formalization:* This is the very first stage in DATESSO and it relies on the *Two Levels Formalizer* component. Generally, it has two tasks at step 1: (i) formulating and recording the global/local level constraints as documented in the SLA; (ii) monitoring the service composition and informing the *Modeling* stage, along with any information of the constraints, when any violations are detected. More details are discussed in Section 4. Note that here, we trigger adaptation only based on local constraint violations, as we formalize the global ones as soft constraints. However, the global constraint is implicitly considered in the *Reasoning* stage.
(2) *Modeling:* Once the local constraint violation has been detected, at step 2, the *Workload Predictor* keeps track of the historical workload on each abstract service, and provides a

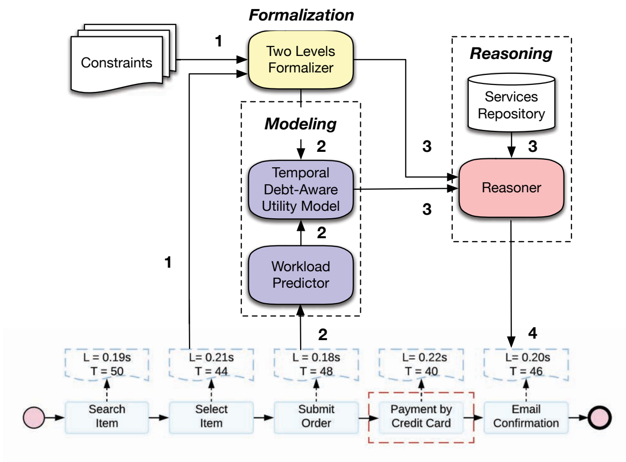

Figure 2: The general processes in DATESSO

time-series model to be embedded with the constraint information, which together form the temporal debt-aware utility model. A detailed discussion will be presented in Section 5

(3) *Reasoning:* At the final stage, the utility model that is debt-aware, the two level constraints and the *Service Repository* with all possible component services would be exploited by the *Reasoner* at step 3. Specifically, we design a debt-aware two levels constraint reasoning algorithm that (i) enables more efficient processing by reducing the original search space based on the constraint information, and (ii) produces a composition plan that is likely to have the highest long-term benefit, without explicitly using global constraints as caps or thresholds. Such a composition plan would then be sent for execution (step 4). The algorithm will be illustrated in greater details at Section 6.

Indeed, the components in DATESSO can be formulated with a MAPE loop of self-adaptation [27], but we did not explicitly perform such in this work for the purpose of better generality. In fact, DATESSO is agnostic to the concrete architectural pattern, providing that the patterns meet with the needs of the components.

4 TWO LEVELS CONSTRAINTS WITH DIFFERENT STRICTNESS

As mentioned, we consider both local and global constraints for latency/utilization in the *Formalization* stage of DATESSO. Instead of assuming hard constraint for both of them, we treat the global constraint as a soft one, which helps to mitigate the problem of being over-optimistic. The formal model and strictness of the two level constraints are discussed in the following subsections.

For each level, constraint can be related to both utilization and latency values. The utilization is a direct measurement of under-utilized situation, whilst the latency value reflects the problem of over-utilization, as a too high utilization usually means the component service is over-stressed, which results in latency degradation.

4.1 Hard Local Constraints

As discussed in Section 2, the local constraint is usually hard [1, 6], which should not be violated. This is because at the service level, any violation of the constraint would in fact cause severe failure in the workflow execution. For example, a violation of latency/utilization caused by a workload that exceeds the capacity would simply bring the individual service down, which cause outage of the entire service composition.

Locally, for each component service c_{xy} that has a capacity to process $T_{c_{xy}}$ requests in $L_{c_{xy}}$ seconds, we model the normalized constraint ($\mathbb{CL}_{c_{xy}}$) on the normalized actual latency of each request ($\mathbb{L}_{c_{xy}}$) to be satisfied as below, both of which are within $[0, 1]$[2]:

$$\mathbb{L}_{c_{xy}} = \frac{L_{c_{xy}} \times W_{t,c_{xy}}}{T_{c_{xy}}} \leq \mathbb{CL}_{c_{xy}} \tag{1}$$

where $W_{t,c_{xy}}$ is the workload for the corresponding abstract service (hence for c_{xy} too) at timestep t. Likewise, the local constraint ($\mathbb{CU}_{c_{xy}}$) on utilization ($\mathbb{U}_{c_{xy}}$) to be satisfied can be formulated as[3]:

$$\mathbb{U}_{c_{xy}} = \frac{L_{c_{xy}} \times W_{t,c_{xy}}}{\mathbb{CL}_{c_{xy}} \times T_{c_{xy}}} \geq \mathbb{CU}_{c_{xy}} \tag{2}$$

Since the local constraints are hard, we say a component service as *feasible* if, and only if, both utilization and latency constraints are satisfied. Otherwise it is termed *infeasible*.

4.2 Soft Global Constraints

Unlike existing work that model global constraint as hard threshold, we model its soft version that can tolerate certain violation, with an aim to mitigate the issue of over-optimism. Indeed, the way of aggregating the local latency toward the global value for the entire service composition depends on the connectors, which may be sequential, parallel or recursive etc. However, as shown in [2, 51], sequential connector is the most fundamental type and all other connectors can be converted into a sequential one. Therefore in this work, we focus on sequential connector in our models.

Similar to its local counterpart, for all selected component services in the entire service composition, the satisfaction on normalized global constraint ($\mathbb{CL}_{global} \in [0, 1]$) and the normalized actual latency of each request ($\mathbb{L}_{global} \in [0, 1]$) can be calculated by aggregating the locally achieved latency. Specifically, when all the connectors are sequential or they have been converted into sequential ones, the satisfaction of global latency can be formulated as[4]:

$$\mathbb{L}_{global} = \sum_{x} \sum_{y} \mathbb{L}_{c_{xy}} \preceq \mathbb{CL}_{global} \tag{3}$$

Likewise, the global constraint ($\mathbb{CU}_{global}$) on utilization ($\mathbb{U}_{global}$) to be satisfied can be formulated as:

$$\mathbb{U}_{global} = \frac{1}{N} \times \sum_{x} \sum_{y} \mathbb{U}_{c_{xy}} \succeq \mathbb{CU}_{global} \tag{4}$$

[2]Normalization can be achieved by using the lower and upper bounds of possible latency values.

[3]Utilization naturally sits within [0, 1], as any requests go beyond the capacity would be discarded.

[4]We use $\preceq$ to reflect the 'soft' nature of global constraints.

whereby N denotes the total number of abstract services. As mentioned, there is no guarantee that satisfying the local parts at component level can lead to global satisfaction. However, it is easy to see that a violation of a global constraint is contributed by some (or all) of the component services selected, even though their local constraints may have been satisfied.

5 TEMPORAL DEBT-AWARE UTILITY MODEL

In the *Modeling* stage of DATESSO, we propose temporal debt-aware utility model, a notion derived from technical debt metaphor [28], that quantifies the long-term benefit of each service component that support a composition plan. To this end, we adopt the notion of principal and interest [3, 5, 46] to analyze the debt values related to a single component service that is feasible. Built on the concept of two level constraints and their different strictness, a debt can quantify each feasible component service's local contribution to the overall debt at the global level over a period of time.

5.1 Modeling Temporal Debt Value

5.1.1 Principal. The principal, denoted as $P_{c_{xy}}$, is the one-off cost of the processes on adapting a component services c_{xy}. It can be calculated as:

$$P_{c_{xy}} = O_{c_{xy}} \times C_{com} \tag{5}$$

Suppose that the actuation process for adding a component service requires an overhead of 5 seconds (denoted as $O_{c_{xy}}$) and the execution cost of computing resource is \$ 0.005 per second (denoted as C_{com}), then it takes a principal as 5×0.005 = \$ 0.025. Note that $P_{c_{xy}}$ here is a normalized value in the range of $[0, 1]$, based on the lower/upper bounds of the possible execution cost and composition time. The $O_{c_{xy}}$ can be easily known by analyzing the time for previous rounds of composition. Alternatively, it can be obtained via profiling the service broker, as what we have done in this work.

5.1.2 Accumulated interest. Over time, interests can be accumulated due to continuous constraint violations. Since the local constraints are hard, there will be no interest incurred directly at this level. However, because we model the global constraints as the soft ones, any violation of a global constraint is contributed by the component services at the local level, even if the local constraint has been satisfied. In particular, according to Equation 3 and 4, over a period of time, any possible violation of a global constraint would be contributed by all component services that have local utilization/latency worse than the global constraint, which causes potential interest. With this in mind, the accumulated interests of a component service c_{xy} between timestep n and m can be modeled as:

$$I_{n,m,c_{xy}} = \alpha_{n,m,c_{xy}} + \beta_{n,m,c_{xy}} \tag{6}$$

and

$$\alpha_{n,m,c_{xy}} = \sum_{t=n}^{m} (\mathbb{CU}_{global} - \mathbb{U}_{c_{xy}}),\ \forall t \ \stackrel{\bullet}{\equiv}\ \mathbb{CU}_{global} \geq \mathbb{U}_{c_{xy}} \tag{7}$$

$$\beta_{n,m,c_{xy}} = \sum_{t=n}^{m} (\mathbb{L}_{c_{xy}} - \mathbb{CL}_{global}),\ \forall t \ \stackrel{\bullet}{\equiv}\ \mathbb{L}_{c_{xy}} \geq \mathbb{CL}_{global} \tag{8}$$

whereby $\stackrel{\bullet}{\equiv}$ represents 'such that'. Hence, $\alpha_{n,m,c_{xy}}$ and $\beta_{n,m,c_{xy}}$ consider only those timesteps between n and m, at which contribution to the possible violation of a global constraint exists. In particular, these equations guarantee that $\alpha_{n,m,c_{xy}} \geq 0$ and $\beta_{n,m,c_{xy}} \geq 0$.

It is easy to know that in general, if $\alpha_{n,m,c_{xy}} = 0$ and $\beta_{n,m,c_{xy}} = 0$, which means c_{xy} does not contribute to any possible global violation at all, then the overall accumulated interest for c_{xy} over a period of time is 0. Otherwise, the interest, incurred by the contribution to the possible violation of either global utilization or latency constraint (or both), would be part of the debt. For example, when $\mathbb{CU}_{global} = 0.9$ and $\mathbb{CL}_{global} = 0.7$, at a particular timestep t, a feasible component service has utilization and latency of $\mathbb{U}_{c_{23}} = 0.7$ and $\mathbb{L}_{c_{23}} = 0.85$, respectively. In this case, for any possible violation of the global utilization and latency constraint at this timestep, c_{23} would contribute a total of $I_{t,t,c_{23}} = 0.9 - 0.7 + 0.85 - 0.7 = 0.35$ interest (and thus part of the debt) to cause the violations. The overall interest over a range of timesteps would be the sum of the interest incurred by the above case under each timestep.

5.1.3 Connecting debt and utility. Finally, we calculate the debt for a feasible component service between timestep n and m as:

$$D_{n,m,c_{xy}} = P_{c_{xy}} + I_{n,m,c_{xy}} \tag{9}$$

Since both $P_{c_{xy}}$ and $I_{n,m,c_{xy}}$ are normalized or naturally sit between $[0, 1]$, the numeric stability can be improved. Drawing on the above, we then be able to obtain a debt-aware utility score ($S_{n,m,c_{xy}}$) for c_{xy} between n and m, defined as:

$$S_{n,m,c_{xy}} = \sum_{t=n}^{m} \mathbb{U}_{c_{xy}} - \sum_{t=n}^{m} \mathbb{L}_{c_{xy}} - D_{n,m,c_{xy}} \tag{10}$$

A larger $S_{n,m,c_{xy}}$ implies that the component service c_{xy} is more likely to contribute to the satisfaction of global constraints in the long term. Here, it is clear that we will accept certain debt, as long as it can be paid back by achieving better overall utility across the timesteps considered. In this way, during the reasoning process, DATESSO is able to quantify the long-term benefit of each feasible component service over a range of timesteps, based on which enabling better informed reasoning.

5.2 Time-Series Workload Prediction

Predicatively analyzing debt is not uncommon for managing technical debt in software development [28]. Often, the fact of whether a debt can be paid off depends on the present and future values of the debt [10, 44]. This is also an equivalent and important concept in our research, and therefore we seek to predict the future workload of the component services, which in turn, enabling informed reasoning of long-term benefit during self-adaptation.

In DATESSO, we use Autoregressive Fractionally Integrated Moving Average model (ARFIMA) [49], a widely used time-series model, to predict the workload of each abstract service. It is chosen over its counterparts (e.g., ARMA) because it handles a time-series with long memory pattern well.

Accordingly, for each abstract service that is realized by a component service, we prepared the data at each time point to contain a number of observed requests, which would be used by the ARFIMA to predict the likely requests workload for a future timestep. The general expression of ARFIMA (p, d, q) for the process X_t is written as:

$$\Phi(B)(1-B)^d X_t = \Theta(B)\varepsilon_t \tag{11}$$

where $(1-B)^d$ is the fractional differencing operator and the fractional number d is the memory parameter, such that $d \in (-0.5, 0.5)$. The operator B is the backward shift operator. For this, we have $\Phi(B) = 1 - \phi_1 B - \phi_2 B^2 - \ldots - \phi_p B^p$ is the autoregressive polynomial of order p and $\Phi(B) = 1 + \theta_1 B + \theta_2 B^2 + \ldots + \theta_q B^q$ is the moving average polynomial of order q. $BX_t = X_{t-1}$ and ε_t represent the white noise process.

In Section 7.1, we will explain how and what tools we use to determine the values of the parameter p, d and q.

6 DEBT-AWARE TWO LEVELS CONSTRAINT REASONING

Drawing on our formalization of soft/hard constraints at two levels, along with the proposed temporal debt-aware utility model, we design a simple yet efficient reasoning algorithm for self-adapting service composition in the *Reasoning* stage. In a nutshell, once violation on local constraints is detected, the algorithm has two main functions that are run in order:

(1) IDENTIFICATION: In this function, we firstly identify which are the component services that violate the local constraints, as this was what triggered the adaptation. Then, the identified infeasible component services would need to be replaced, as they also contribute to the likely violation of the global constraint(s). It is possible that all component services need to be replaced.
(2) SEARCH: Once we identify the set of abstract services whose component service needs a replacement, this function works on each individual abstract service. The aim is to search for the best feasible component service for each identified abstract service, such that it satisfies the local constraint[5] while having the best long-term debt-aware utility, over all timesteps up to the future timestep m (Equation 10). As a result, the newly selected component services would less likely to cause local/global constraint violation in the future.

Each of the key steps are discussed in details as follows.

6.1 Identifying Infeasible Component Services

As mentioned, since the constraint at local level is hard, the IDENTIFICATION function is designed to filter all the service components that are 'working fine'. In fact, this steps is an effective way to reduce the search space, as only the problematic component services that violates the hard constraints are considered. These infeasible component services can actually contribute to the global constraint violation, if any.

[5]Given that the local constraint is specified at the local level, there will be at least one readily available component service to satisfy such constraint at a particular timestep, or otherwise the constraint may be too strong and needs to be relaxed.

Algorithm 1: IDENTIFICATION

```
Input: S: Set of selected component services and their abstract
    services at current timestep n
Output: S_inf ← ∅: Set of abstract services whose component
    service needs a replacement
for ∀c_xy ∈ S do
    if (𝕃_cxy > ℂ𝕃_cxy or 𝕌_cxy < ℂ𝕌_cxy) then
        S_inf ← a_x
    end
end
return S_inf
```

The corresponding algorithmic procedure has been illustrated in Algorithm 1. As can be seen, the returned result is a set, denoted as S_{inf}, that contains every abstract service (i.e., a_x) whose component service becomes infeasible at the current timestep n.

6.2 Searching for the Best Long-term Debt-Aware Utility

The special design in the SEARCH function is that, instead of having to examine every combination of the service composition globally, we only search for the component service with the highest long-term debt-aware utility for each identified abstract service independently.

This is because, according to Equation 10, the problem of searching the highest long-term debt-aware utility (between timestep n and m) for the entire service composition can be defined as follow:

$$\textbf{argmax} \sum_{x=1}^{Z} S_{n,m,c_{xy}} \tag{12}$$

whereby Z is the total number of abstract services whose component service need a replacement. Clearly, this is a typical linear programming problem, in which achieving the best utility of the service composition is equal to finding the optimal value of each $S_{n,m,c_{xy}}$. From Equation 10, we know that the best $S_{n,m,c_{xy}}$ is solely equivalent to the highest debt-aware utility from all the feasible component services of the xth abstract service. In other words, the highest $S_{n,m,c_{xy}}$ can be searched on each abstract service locally, in order to have the highest utility for the service composition globally. With this consideration, our reasoning algorithm decomposes the problem and reduces the search complexity from $O(Y^X)$ (when all combinations need to be searched at the global level) down to $O(Y \times X)$, where X is the number of problematic abstract service, each with Y feasible component services[6].

The corresponding algorithmic procedure has been illustrated in Algorithm 2. Specifically, suppose that the S_{inf} has been found by Algorithm 1, and that the current timestep is n and we are interested in the debt up to a given timestep m in the future, there are three important steps:

(1) From line 4 to 14, for each problematic abstract service a_x, we firstly construct an ordered list of vectors denoted as M_x. Each vector in M_x has a size of $m - n$ and it contains all the

[6]Y may differ for different abstract services, but in this example we assume that same as our aim is merely to intuitively illustrate the reduction of complexity.

feasible component service for a_x under every particular timestep between n and m.

(2) From line 15 to 20, for each M_x, we find the largest timestep m_x since n such that there is at least one feasible component service that satisfies the local constraint on every timestep between n and m_x. Next, we use the smallest m_x across all M_x to serve as the new m. This process ensures that all problematic abstract services would have at least one component service which can be treated as feasible on all timesteps considered. Here, since there is at least one feasible component service for a particular timestep, the worst case would be $m = n + 1$.

(3) From line 21 to 24, for each a_x, we find the set of feasible component services (S_x) that satisfy the local constraints on every timestep between n and m. The SearchUtility function searches locally on the set S_x, and returns the one with the highest $S_{n,m,c_{xy}}$ as part of the composition plan. Note that, SearchUtility can be realized by any search algorithm, e.g., exhaustive search or stochastic search like Genetic Algorithm. Since in this work the S_x has been reduced to a computationally tractable size, we simply apply an exhaustive search.

As the global constraints are soft, the reasoning algorithm has never explicitly used them to act as caps or thresholds for the search (like what we did for the hard local constraints), but the global constraints, along with their potential violations contributed by the component services, are implicitly embedded in the debt-aware utility model. In this way, we aim to mitigate the problem of being over-optimism on the global constraint, while at the same time, promoting larger chance to satisfy the global constraint in the long term.

Algorithm 2: Search

1 **Input**: R_x: *The set of possible component services for the xth abstract service*
2 S_{inf}: *Set of abstract services whose component service needs a replacement*
3 **Output**: $S_{optimal}$: *Service composition plan with the optimal long-term debt-aware utility between current timestep n and the future timestep m*
4 **for** $\forall a_x \in S_{inf}$ **do**
  /* M_x denotes the ordered list of vectors of the feasible component services for the xth abstract service at every timestep from n to a future timestep m */
  /* $\mathbb{S}_{x,t}$ denotes the vector of the feasible component services for the xth abstract service at timestep t */
5   $M_x = \{\mathbb{S}_{x,n}, \mathbb{S}_{x,n+1}, \ldots, \mathbb{S}_{x,m}\}$
6   **for** $\forall c_{xy} \in R_x$ **do**
7   **for** $t \leftarrow n + 1$ ***to*** m **do**
8   **if** $(\mathbb{L}_{c_{xy}} \leq \mathbb{CL}_{c_{xy}}$ ***and*** $\mathbb{U}_{c_{xy}} \geq \mathbb{CU}_{c_{xy}})$ **then**
9         $\mathbb{S}_{x,t} \leftarrow c_{xy}$
10   **end**
11   **end**
12   **end**
13   $M \leftarrow M_x$
14 **end**
15 **for** $\forall M_x \in M$ **do**
  /* According to M_x, the function getLargestFeasibleStep returns the largest timestep m_x from n such that there is at least one component service that satisfies the local constraint on every timestep between n and m_x */
16   m_x = getLargestFeasibleStep(M_x)
17   **if** $m_x < m$ **then**
18     $m = m_x$
19   **end**
20 **end**
21 **for** $\forall M_x \in M$ **do**
  /* According to M_x and the new m, the function getFeasibleServices returns the component services that satisfy the the local constraint on every timestep between n and m */
22   S_x = getFeasibleServices(M_x,m)
  /* Function searchUtility returns the component service with the highest $S_{n,m,c_{xy}}$ for a_x */
23   $S_{optimal} \leftarrow$ searchUtility(S_x,n,m)
24 **end**
25 **return** $S_{optimal}$

7 EVALUATION

To evaluate DATESSO, we design experiments to assess the performance of our technique on self-adapting service composition by means of comparing it with the state-of-the-art approaches. In particular, we aim to answer the following research questions (RQs):

- **RQ1:** Can DATESSO achieve better global utilization and latency than the state-of-the-art approaches? If so, which parts contribute to the improvement?
- **RQ2:** Is DATESSO more sustainable than the state-of-the-art approaches?
- **RQ3:** What is the running overhead of the reasoning process in DATESSO comparing with the others?

7.1 Experimental Setup

Our experiments have used a commonly applied service-based system [23, 24, 33] with 10 abstract services, each of which has 10 possible component services to be selected. Without considering reduction, the system would have a search space of 10^{10} possible composition plans for self-adaptation. All the values of latency and throughput capacity for the component services are randomly chosen from the WS-DREAM dataset [53].

To emulate realistic workload for each abstract service that is realized by a component service, we extracted the FIFA98 trace [7] for the length of 6 hours with 7200 timesteps, which forms the workload dataset. Such a workload trace is used on all the different workflows of service composition. We pre-processed the first four hours of workload trace as the samples for training the time-series prediction model, while the remaining two hours of workload data, which equals to 7200 seconds, is used for testing the accuracy. In DATESSO, we feed the training data into the ARFIMA, which is implemented using the `arfima` package [47] and the fdGPH

Table 1: Parameter settings

Parameter	Value
$\mathbb{CL}_{c_{xy}}$: local latency constraint per request	0.09s
$\mathbb{CL}_{global}$: global latency constraint per request	1s
$\mathbb{CU}_{c_{xy}}$: local utilization constraint	0.8
$\mathbb{CU}_{global}$: global utilization constraint	0.9
C_{com}: cost of computing resource	\$0.0025
m: future timestep m from current timestep n	$n+5$

function in R [40]. The values of p, d and q are also identified therein.

Table 1 shows the parameter settings of the SLA used in the experiments, including the executing resource of selecting a component service (C_{com}), the local and global constraint for latency ($\mathbb{CL}_{c_{xy}}$ and $\mathbb{CL}_{global}$) and utilization ($\mathbb{CU}_{c_{xy}}$ and $\mathbb{CU}_{global}$). For simplicity of exposition, we have set the same local constraint for all abstract services. All the settings above have been tailored to be reasonable throughout the experiments.

All experiments were carried out on a machine with Intel Core i7 2.60 GHz. CPU, 8GB RAM and Windows 10.

7.2 Comparative Approaches

To answer all the RQs, we examine the performance of DATESSO against the following approaches:

- **Two Level Hard Constraints Approach (TLHCA):** This is similar to DATESSO, which differs only on the way about how the strictness of the two levels constraints is formulated. TLHCA assumes that both local and global constraints are hard, and thereby in the reasoning algorithm (Algorithm 2), when the final composition plan violates the global constraint (for every timestep between n and the newly defined m) then we examine whether all abstract services have been considered in this run. If not, we then rerun the algorithm with consideration that all the abstract services are subject to replacement; if all abstract services has been considered but the global constraint(s) is still violated, we would have no choice but to trigger the adaptation. Here, the adaptation is triggered based on both local and global constraint violations. This approach follows the existing work [6] that makes the same formulation, and by this mean, we aim to examine the usefulness of formulating the global constraints as the soft ones.
- **Debt-Oblivious Approach (DOA):** This is a similar copy of DATESSO but without the temporal debt-aware utility model. Instead, DOA assumes the predicted utility of the aggregated latency and utilization, i.e., Equation 10 without the debt, which is then used in the reasoning algorithm to find the composition plan for self-adaptation. Such a predicted approach has been used in existing work [29], and DOA helps us to examine the effectiveness of incorporating debt information for achieving long-term benefit in self-adaptation.
- **Region-Based Composition (RBC)** This is an implementation of a state-of-the-art approach, proposed by Lin et al. [39], that relies on regions, where for each abstract services, the component service is selected according to its region. Each of these regions are clustered based on the historical utilization and latency of the component services. Here, the adaptation is triggered based on global constraint violations only. RBC is chosen as it is one of the most widely known representative approaches for dynamic service composition.

7.3 Metrics

We leverage the following metrics to assess the results:

- **Global utilization:** This is the value calculated by Equation 4 for each timestep.
- **Global latency:** This is the value calculated by Equation 3 for each timestep.
- **Accumulated debt:** Since the interests are accumulated, so does the debt. A lower debt means that component services, which are less likely to contribute to global constraint violation in the long term, are preferred. Therefore, we measure the accumulated debt of the service composition from the beginning to the timestep t using:
$$D_{1,t} = \sum_{x}\sum_{y} D_{1,t,c_{xy}} \tag{13}$$
- **Sustainability score:** We measure sustainability as follows:
$$Score_{n,m} = \frac{1}{V} \times \left(\frac{S_{n,m} - S_{min,n,m}}{S_{max,n,m} - S_{min,n,m}} + 1\right) \tag{14}$$
whereby $S_{n,m} = \sum_{x=1}^{Z} S_{n,m,c_{xy}}$; $n = 1$ and $m = 7200$; Z is the total number of abstract services; V is the total number of local and global constraint violations. $S_{min,n,m}$ and $S_{min,n,m}$ are the lower and upper value among all approaches. $Score_{n,m} \in [1, 2]$ and a higher value means that the adaptations would generate more benefits in general when mitigating each constraint violation.
- **Running time:** This is the required running time for the reasoning process to produce a composition plan.

Whenever overall results are reported, we use the pairwise version of the Kruskal Wallis test ($\alpha = .05$) [31] and η^2 value [26] to measure statistical significance and effect size, respectively.

7.4 RQ1: Performance of DATESSO

Figure 3 and 4 respectively illustrate the global utilization and latency for all approaches and timesteps. As can be seen, the comparison between DATESSO and any other three are statistically significant with large effect size. In particular, when comparing with RBC, DATESSO achieves much better utilization and latency overall, while at the same time, it has smaller variance than RBC.

To better understand which of our contributions in DATESSO enable such improvement, we firstly compare it with TLHCA and DOA. As shown in the boxplots, we see that DATESSO achieves much better utilization and smaller variance. For latency, DATESSO is slightly more deviated, but provides overall better result. This has proved that, in general, the formalization of two levels constraints with different strictness can help to improve self-adaptation performance. Next, we compare DATESSO with DOA, for which we see that again,

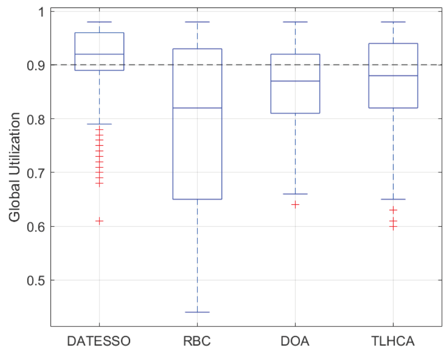

Figure 3: Global utilization yield by all approaches over 7200 timesteps (Comparisons between DATESSO and others are statistically significant ($p < .05$) and with large effect size)

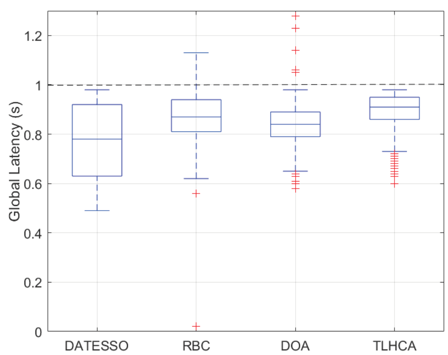

Figure 4: Global latency yield by all approaches over 7200 timesteps (Comparisons between DATESSO and others are statistically significant ($p < .05$) and with large effect size)

DATESSO achieves generally better and more stable results on utilization and latency. This evidences that the predicted debt model can provide more benefit than simply having a predicted model based solely on utilization and latency.

Remarkably, DATESSO achieves full satisfaction for the global constraint on latency and satisfy that of utilization for majority of the cases, which are generally superior to the other three. Therefore, for **RQ1**, we conclude that:

> **Answering RQ1:** DATESSO is more effective than the state-of-the-arts in terms of the utilization and latency, with better satisfactions. Both the design of formalizing global constraints as the soft ones and the temporal debt-aware utility model have contributed to the improvement.

7.5 RQ2: Sustainability of DATESSO

We now assess the sustainability of adaptation achieved by using the accumulated debt and sustainability score. Figure 5 shows the accumulated debt, in which we see that all approaches have accumulated debt in a linear and steady manner. However, clearly,

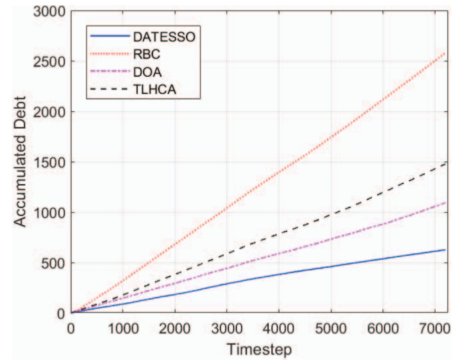

Figure 5: Total debt accumulated by all approaches over 7200 timesteps

Table 2: Sustainability scores

Approach	$\sum_{x=1}^{Z} S_{n,m,c_{xy}}$	V	$Score_{n,m}$
DATESSO	417.10	113	.0177
RBC	−3146.66	187	.0053
DOA	−910.61	102	.0160
TLHCA	−1478.67	133	.0110

DATESSO results in significantly less debt than the other three as it accumulates overtime, suggesting that DATESSO favours component services that is less likely to contribute to global constraint violation in the long term.

Table 2 shows the sustainability scores for all approaches. As can been seen, despite that DATESSOand DOA have similar total number of constant violations, DATESSO has achieved the best $Score_{n,m}$ value among others. This implies that the adaptations in DATESSO would create the greatest benefit in mitigating per violation. All the above conclude that:

> **Answering RQ2:** DATESSO is more sustainable than the other three, as it has less accumulated debt and with the highest sustainability score. This means that DATESSO favors more reliable component services in the long term, and that it offers greater benefit when dealing with each violation overall.

7.6 RQ3: Running Time of DATESSO

Figure 6 illustrates the running time for all approaches. We can clearly see that RBC is the slowest due to the region based algorithm. TLHCA is the 2nd slowest because of the frequent need of replacing all component services. Since DATESSO and DOA differ only on whether having the debt calculation, they have similar running overhead ($p > .05$) but are significantly faster than the others. This is because only the problematic abstract services, along with those component services that satisfy all considered timesteps, are involved in the actual search, which reduces the search space. However, as we have shown, DATESSO offers much better performance and sustainability than DOA. In summary, we have:

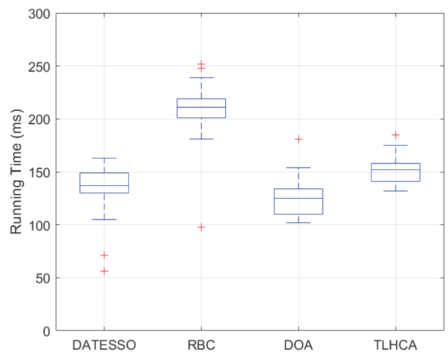

Figure 6: Running time on all approaches (Comparisons between DATESSO and others are statistically significant ($p < .05$) and with large effect size, except for DOA)

Answering RQ3: DATESSO and DOA both have similar running time, but they are faster than the other two.

8 THREATS TO VALIDITY

Threats to construct validity can be related to the metric and evaluation methods used. To mitigate such, we use a broad range of metrics for evaluating different aspects of DATESSO, including utilization, latency and sustainability etc. To examine the effectiveness of each contribution, we have compared DATESSO with specifically designed approaches, i.e., TLHCA and DOA, in addition to a direct implementation of existing work (RBC). Further, we plot all the data points in a trace, and applied statistical test and effect size interpretation when it is difficult to show all the data points.

Threats to internal validity can be mainly related to the value of the parameters for DATESSO. Particularly, the setup has been designed in a way that it produces good trade-off between the quality and overhead. They have been shown to be reasonable following preliminary runs in our experiments. The future timestep m is also specifically tailored and the used value tends to be sufficient. However, it is worth noting that the actual future timesteps to use is updated dynamically depending on whether there is a feasible component service that satisfies all considered timesteps.

Threats to external validity can be associated with the environment and the dataset that are used in the experiment. To improve generalization, we apply commonly used service-based system [23, 24, 33], whose data is randomly sampled from the real-world WS-Dream dataset [53], along with the FIFA98 workload trace [7]. A more comprehensive evaluation on different dataset and more complicated structures are parts of the future work.

9 RELATED WORK

Self-adapting service composition is certainly not new for research on service-based systems. Among others, Lin et al. [39] and Li et al. [38] rely on region-based composition, in which an expand region algorithm is proposed to identify the region of each component service, which forms a reduced search space. Dai et.al. [29] leverage time-series prediction on workload when reasoning about the self-adaptation. Chen et al. [23, 24] seed the multi-objective evolutionary algorithms to accelerate the reasoning process of service composition. The commonality of the above work is the fact that they all assume both local and global constraints are hard ones during reasoning, which can be over-optimistic. Therefore, they can easily lead to the situation of 'no satisfactory composition plans found'. DATESSO, in contrast, formalizes the global one as soft constraint, which mitigates the issues of over-optimism and also reward some plans that may temporarily cause global violation, but tends to be more sustainable with larger long-term benefit.

Technical debt has been studied in service composition [4, 43] and in a wider context of self-adaptive systems [16]. For example, Chen et al. [16] have used technical debt as a metaphor to model the problem of *to adapt or not to adapt*. To resolve such, an online classifier, combined with debt calculation, is proposed. However, the above work does not explicitly consider time-varying and accumulated properties of the debt.

In summary, the key additions in DATESSO are that

- DATESSO formalizes different strictness for the two levels constraint in service composition.
- DATESSO makes use of a new debt model that was designed based on the different strictness of the two levels constraints and time-series prediction. It is therefore temporal, capable of quantifying accumulated debt and tailored to the problem context.
- Drawing on the above, DATESSO proposes to leverage a simple but effective and efficient reasoning algorithm that reduces the search space and focuses on the long-term benefits of self-adaptation.

The benefits of all the above contributions have been experimentally demonstrated in Section 7.

10 CONCLUSION

In this paper, we propose a debt-aware two level constraint reasoning approach, dubbed DATESSO, for self-adapting service composition. DATESSO formalizes the global constraints as the soft ones while leaving only the local ones as hard constraints. Such formalization is then used to built a temporal debt-aware utility model, supported by time-series prediction. The utility model, together with the different strictness of the two level constraints, enable us to design a simple yet efficient and effective reasoning algorithm in DATESSO. Experimental results demonstrate that DATESSO is more effective that state-of-the-art in terms of utilization, latency and running time, while being about to make each self-adaptation more sustainable.

In future work, we seek to extend DATESSO for better synergy between Software Engineering and Artificial Intelligence driven self-adaptation [19, 20, 35], particularly on stochastic multi-objective search algorithms which have been shown to provide promising results on scenarios with complex trade-off surface for self-adaptive software systems [13, 15, 18, 21, 22, 25, 36, 37, 45]. Online learning based prediction on the satisfaction of local/global constraints [11, 12, 14, 17] is also part of our ongoing research agenda.

REFERENCES

[1] Mohammad Alrifai and Thomas Risse. 2009. Combining global optimization with local selection for efficient QoS-aware service composition. In *Proceedings of the 18th international conference on World wide web*. 881–890.

[2] Mohammad Alrifai, Thomas Risse, and Wolfgang Nejdl. 2012. A hybrid approach for efficient Web service composition with end-to-end QoS constraints. *ACM Transactions on the Web (TWEB)* 6, 2 (2012), 7:1–7:31. https://doi.org/10.1145/2180861.2180864

[3] Nicolli SR Alves, Thiago S Mendes, Manoel G de Mendonça, Rodrigo O Spínola, Forrest Shull, and Carolyn Seaman. 2016. Identification and management of technical debt: A systematic mapping study. *Information and Software Technology* 70 (2016), 100–121.

[4] Esra Alzaghoul and Rami Bahsoon. 2013. CloudMTD: Using real options to manage technical debt in cloud-based service selection. In *2013 4th International Workshop on Managing Technical Debt (MTD)*. IEEE, 55–62.

[5] Areti Ampatzoglou, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, and Paris Avgeriou. 2015. The financial aspect of managing technical debt: A systematic literature review. *Information and Software Technology* 64 (2015), 52–73.

[6] Danilo Ardagna and Barbara Pernici. 2005. Global and local QoS constraints guarantee in web service selection. In *IEEE International Conference on Web Services (ICWS'05)*. IEEE.

[7] Martin Arlitt and Tai Jin. 2000. A workload characterization study of the 1998 world cup web site. *IEEE network* 14, 3 (2000), 30–37.

[8] Rafael Aschoff and Andrea Zisman. 2011. QoS-driven proactive adaptation of service composition. In *International Conference on Service-Oriented Computing*. Springer, 421–435.

[9] Paris Avgeriou, Philippe Kruchten, Ipek Ozkaya, and Carolyn Seaman. 2016. Managing technical debt in software engineering (dagstuhl seminar 16162). In *Dagstuhl Reports*, Vol. 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[10] Frank Buschmann. 2011. To pay or not to pay technical debt. *IEEE software* 28, 6 (2011), 29–31.

[11] Tao Chen. 2019. All versus one: an empirical comparison on retrained and incremental machine learning for modeling performance of adaptable software. In *Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Marin Litoiu, Siobhán Clarke, and Kenji Tei (Eds.). ACM, 157–168. https://doi.org/10.1109/SEAMS.2019.00029

[12] Tao Chen and Rami Bahsoon. 2013. Self-adaptive and sensitivity-aware QoS modeling for the cloud. In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2013, San Francisco, CA, USA, May 20-21, 2013*. 43–52. https://doi.org/10.1109/SEAMS.2013.6595491

[13] Tao Chen and Rami Bahsoon. 2015. Toward a Smarter Cloud: Self-Aware Autoscaling of Cloud Configurations and Resources. *IEEE Computer* 48, 9 (2015), 93–96. https://doi.org/10.1109/MC.2015.278

[14] Tao Chen and Rami Bahsoon. 2017. Self-Adaptive and Online QoS Modeling for Cloud-Based Software Services. *IEEE Trans. Software Eng.* 43, 5 (2017), 453–475. https://doi.org/10.1109/TSE.2016.2608826

[15] Tao Chen and Rami Bahsoon. 2017. Self-Adaptive Trade-off Decision Making for Autoscaling Cloud-Based Services. *IEEE Trans. Services Computing* 10, 4 (2017), 618–632. https://doi.org/10.1109/TSC.2015.2499770

[16] 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 *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering, ICPE 2018, Berlin, Germany, April 09-13, 2018*. 48–55. https://doi.org/10.1145/3184407.3184413

[17] Tao Chen, Rami Bahsoon, and Xin Yao. 2014. Online QoS Modeling in the Cloud: A Hybrid and Adaptive Multi-learners Approach. In *Proceedings of the 7th IEEE/ACM International Conference on Utility and Cloud Computing, UCC 2014, London, United Kingdom, December 8-11, 2014*. 327–336. https://doi.org/10.1109/UCC.2014.42

[18] Tao Chen, Rami Bahsoon, and Xin Yao. 2018. A Survey and Taxonomy of Self-Aware and Self-Adaptive Cloud Autoscaling Systems. *ACM Comput. Surv.* 51, 3 (2018), 61:1–61:40. https://doi.org/10.1145/3190507

[19] Tao Chen, Rami Bahsoon, and Xin Yao. 2020. Synergizing Domain Expertise with Self-Awareness in Software Systems: A Patternized Architecture Guideline. *Proc. IEEE* in press (2020).

[20] Tao Chen, Funmilade Faniyi, Rami Bahsoon, Peter R. Lewis, Xin Yao, Leandro L. Minku, and Lukas Esterle. 2014. The Handbook of Engineering Self-Aware and Self-Expressive Systems. *CoRR* abs/1409.1793 (2014). arXiv:1409.1793 http://arxiv.org/abs/1409.1793

[21] Tao Chen, Ke Li, Rami Bahsoon, and Xin Yao. 2018. FEMOSAA: Feature-Guided and Knee-Driven Multi-Objective Optimization for Self-Adaptive Software. *ACM Trans. Softw. Eng. Methodol.* 27, 2 (2018), 5:1–5:50. https://doi.org/10.1145/3204459

[22] Tao Chen, Miqing Li, Ke Li, and Kalyanmoy Deb. 2020. Search-Based Software Engineering for Self-Adaptive Systems: One Survey, Five Disappointments and Six Opportunities. *CoRR* abs/2001.08236 (2020). arXiv:2001.08236 https://arxiv.org/abs/2001.08236

[23] Tao Chen, Miqing Li, and Xin Yao. 2018. On the effects of seeding strategies: a case for search-based multi-objective service composition. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, July 15-19, 2018*. 1419–1426. https://doi.org/10.1145/3205455.3205513

[24] Tao Chen, Miqing Li, and Xin Yao. 2019. Standing on the shoulders of giants: Seeding search-based multi-objective optimization with prior knowledge for software service composition. *Information & Software Technology* 114 (2019), 155–175. https://doi.org/10.1016/j.infsof.2019.05.013

[25] Tao Chen, Miqing Li, and Xin Yao. 2020. How to Evaluate Solutions in Pareto-based Search-Based Software Engineering? A Critical Review and Methodological Guidance. *CoRR* abs/2002.09040 (2020). arXiv:2002.09040 https://arxiv.org/abs/2002.09040

[26] Jacob Cohen. 2013. *Statistical power analysis for the behavioral sciences*. Routledge.

[27] Autonomic Computing et al. 2006. An architectural blueprint for autonomic computing. *IBM White Paper* 31, 2006 (2006), 1–6.

[28] Ward Cunningham. 1992. The WyCash portfolio management system. *ACM SIGPLAN OOPS Messenger* 4, 2 (1992), 29–30.

[29] Yu Dai, Lei Yang, and Bin Zhang. 2009. QoS-driven self-healing web service composition based on performance prediction. *Journal of Computer Science and Technology* 24, 2 (2009), 250–261.

[30] Martine De Cock, Sam Chung, and Omar Hafeez. 2007. Selection of web services with imprecise QoS constraints. In *IEEE/WIC/ACM International Conference on Web Intelligence (WI'07)*. IEEE, 535–541.

[31] William H Kruskal and W Allen Wallis. 1952. Use of ranks in one-criterion variance analysis. *Journal of the American statistical Association* 47, 260 (1952), 583–621.

[32] Satish Kumar, Rami Bahsoon, Tao Chen, and Rajkumar Buyya. 2019. Identifying and Estimating Technical Debt for Service Composition in SaaS Cloud. In *2019 IEEE International Conference on Web Services (ICWS)*. IEEE, 121–125.

[33] Satish Kumar, Rami Bahsoon, Tao Chen, Ke Li, and Rajkumar Buyya. 2018. Multi-Tenant Cloud Service Composition Using Evolutionary Optimization. In *24th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2018, Singapore, December 11-13, 2018*. 972–979. https://doi.org/10.1109/PADSW.2018.8644640

[34] Touraj Laleh, Joey Paquet, Serguei Mokhov, and Yuhong Yan. 2017. Constraint adaptation in Web service composition. In *2017 IEEE International Conference on Services Computing (SCC)*. IEEE, 156–163.

[35] Peter R. Lewis, Arjun Chandra, Funmilade Faniyi, Kyrre Glette, Tao Chen, Rami Bahsoon, Jim Tørresen, and Xin Yao. 2015. Architectural Aspects of Self-Aware and Self-Expressive Computing Systems: From Psychology to Engineering. *IEEE Computer* 48, 8 (2015), 62–70. https://doi.org/10.1109/MC.2015.235

[36] Ke Li, Zilin Xiang, Tao Chen, Shuo Wang, and Kay Chen Tan. 2020. Understanding the Automated Parameter Optimization on Transfer Learning for CPDP: An Empirical Study. In *Proceedings of the 42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea*.

[37] Miqing Li, Tao Chen, and Xin Yao. 2018. A critical review of: "a practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering": essay on quality indicator selection for SBSE. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results, ICSE (NIER) 2018, Gothenburg, Sweden, May 27 - June 03, 2018*. 17–20. https://doi.org/10.1145/3183399.3183405

[38] Ying Li, Yuanlei Lu, Yuyu Yin, Shuiguang Deng, and Jianwei Yin. 2010. Towards qos-based dynamic reconfiguration of soa-based applications. In *2010 IEEE Asia-Pacific Services Computing Conference*. IEEE, 107–114.

[39] Kwei-Jay Lin, Jing Zhang, Yanlong Zhai, and Bin Xu. 2010. The design and implementation of service process reconfiguration with end-to-end QoS constraints in SOA. *Service Oriented Computing and Applications* 4, 3 (2010), 157–168.

[40] Maintainer Martin Maechler. 2019. Package 'fracdiff'. (2019).

[41] Franco Raimondi, James Skene, and Wolfgang Emmerich. 2008. Efficient online monitoring of web-service SLAs. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. 170–180.

[42] Florian Rosenberg, Predrag Celikovic, Anton Michlmayr, Philipp Leitner, and Schahram Dustdar. 2009. An end-to-end approach for QoS-aware service composition. In *2009 IEEE International Enterprise Distributed Object Computing Conference*. IEEE, 151–160.

[43] Georgios Skourletopoulos, Constandinos X Mavromoustakis, Jordi Mongay Batalla, George Mastorakis, Evangelos Pallis, and Georgios Kormentzas. 2016. Quantifying and evaluating the technical debt on mobile cloud-based service level. In *2016 IEEE International Conference on Communications (ICC)*. IEEE, 1–7.

[44] Will Snipes, Brian Robinson, Yuepu Guo, and Carolyn Seaman. 2012. Defining the decision factors for managing defects: a technical debt perspective. In *2012 Third International Workshop on Managing Technical Debt (MTD)*. IEEE, 54–60.

[45] Dalia Sobhy, Leandro L. Minku, Rami Bahsoon, Tao Chen, and Rick Kazman. 2020. Run-time evaluation of architectures: A case study of diversification in IoT. *J. Syst. Softw.* 159 (2020). https://doi.org/10.1016/j.jss.2019.110428

[46] Edith Tom, AybüKe Aurum, and Richard Vidgen. 2013. An exploration of technical debt. *Journal of Systems and Software* 86, 6 (2013), 1498–1516.

[47] Justin Q Veenstra, AI McLeod, and Maintainer JQ Veenstra. 2015. Package 'arfima'. (2015).
[48] PengWei Wang, ZhiJun Ding, ChangJun Jiang, and MengChu Zhou. 2013. Constraint-aware approach to web service composition. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 44, 6 (2013), 770–784.
[49] Jin Xiu and Yao Jin. 2007. Empirical study of ARFIMA model based on fractional differencing. *Physica A: Statistical Mechanics and its Applications* 377, 1 (2007), 138–154.
[50] Lei Yang, Yu Dai, and Bin Zhang. 2009. Performance Prediction Based EX-QoS Driven Approach for Adaptive Service Composition. *Journal of Information Science & Engineering* 25, 2 (2009).
[51] Tao Yu, Yue Zhang, and Kwei-Jay Lin. 2007. Efficient algorithms for Web services selection with end-to-end QoS constraints. *ACM Transactions on the Web (TWEB)* 1, 1 (2007), 6. https://doi.org/10.1145/1232722.1232728
[52] Liangzhao Zeng, Boualem Benatallah, Anne HH Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. 2004. QoS-aware middleware for web services composition. *IEEE Transactions on software engineering* 30, 5 (2004), 311–327.
[53] Zibin Zheng, Yilei Zhang, and Michael R Lyu. 2012. Investigating QoS of real-world web services. *IEEE transactions on services computing* 7, 1 (2012), 32–39.