DebtCom: Technical Debt-Aware Service Recomposition in SaaS Cloud

Satish Kumar, Tao Chen, Rami Bahsoon, and Rajkumar Buyya, *Fellow, IEEE*

***Abstract*—Given the changing workloads from the tenants, it is not uncommon for a service composition running in the multi-tenant SaaS cloud to encounter under-utilization and over-utilization on the component services. Both cases are undesirable and it is therefore nature to mitigate them by recomposing the services to a newly optimized composition plan once they have been detected. However, this ignores the fact that under-/over-utilization can be merely caused by temporary effects, and thus the advantages may be short-term, which hinders the long-term benefits that could have been created by the original composition plan, while generating unnecessary overhead and disturbance via recomposition. In this article, we propose `DebtCom`, a framework that determines whether to trigger recomposition based on the technical debt metaphor and time-series prediction of workload. In particular, we propose a service debt model, which has been explicitly designed for the context of service composition, to quantify the debt. Our core idea is that recomposition can be unnecessary if the under-/over-utilization only cause temporarily negative effects, and the current composition plan, although carries debt, can generate greater benefit in the long-term. We evaluate `DebtCom` on a large scale service system with up to 10 abstract services, each of which has 100 component services, under real-world dataset and workload traces. The results confirm that, in contrast to the state-of-the-art, `DebtCom` achieves better utility while having lower cost and number of recompositions, rendering each composition plan more sustainable.**

***Index Terms*—Optimization, service composition, software adaptation, technical debt.**

I. Introduction

SERVICE composition has emerged as a standard way of building software application by composing multiple existing web services [1]. The resulted software application, namely composite service, is often deployed in the cloud, forming the basics of modern Software-as-a-Service (SaaS). A pronounced benefit of composite service in the SaaS cloud is the realization of multi-tenancy, where multiple tenants[1] are simultaneously served by the same composite service based on shared resources [2]. However, the workload of composite service can be changed rapidly during execution, causing dynamic behaviour of the composite services. On one hand, increasing workload can cause over-utilization for the component services[2] within a composite service, which in turn, would negatively affect the Quality of Service (QoS) and violate Service Level Agreements (SLAs) [3]. 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. All those bring a challenging task: when to (re)compose the component services such that the utility over time is maximized?

While service recomposition (or reconfiguration) has been widely studied [4], [5], [6], [24], existing researches have ignored a perhaps obvious, but complicated fact: a short-term degradation of the utility may not necessarily be a bad results; in fact, it can be the source that stimulates largely increased utility in the long term. For example, under-utilization could be desirable temporarily in order to be prepared for a largely increased workload for the long-term. Similarly, over-utilization may be acceptable in the short term, as long as the workload is only a 'spike' and the loss can be paid off by long term benefits. Simply ignoring such fact is non-trivial, because despite triggering recomposition immediately upon over-/under-utilization my have short-term advantages, it can easily create instability and hinder the possibility of achieving higher benefits for the composite services in the long-term.

To address the mentioned challenges and limitations, in this paper, we contribute to an economic-driven approach, namely `DebtCom`, for triggering dynamic service recomposition leveraging the principle of technical debt—a well-known software engineering concept [9], [10]. In particular, we argue that the technical debt could be the consequences of making poorly justified run-time decisions for recomposing the composite service during execution. Obviously, a rapidly changing workload on the composite service may lead to sub-optimal[3] utilization of the capacity of component services during execution. Therefore, sub-optimal composite service execution attributes the debt in a way to provide higher capacity component web services in

Manuscript received 6 September 2021; revised 25 September 2022; accepted 25 October 2022. Date of publication 17 January 2023; date of current version 8 August 2023. This work was partially supported by the Engineering and Physical Sciences Research Council (EPSRC), U.K. under Grant EP/T01461X/1). Recommended for acceptance by H. Mei. *(Corresponding author: Satish Kumar.)*

Satish Kumar is with the School of Computing, University of Leeds, LS2 9JT Leeds, U.K. (e-mail: s.kumar3@leeds.ac.uk).

Tao Chen is with the Department of Computer Science, Loughborough University, LE11 3TU Loughborough, U.K. (e-mail: txc818@gmail.com).

Rami Bahsoon is with the School of Computer Science, University of Birmingham, B15 2TT Birmingham, U.K. (e-mail: r.bahsoon@cs.bham.ac.uk).

Rajkumar Buyya is with the School of Computing and Information System, University of Melbourne, Melbourne, VIC 3010, Australia (e-mail: rbuyya@unimelb.edu.au).

Digital Object Identifier 10.1109/TSC.2023.3237043

[1]Tenants denote the end-users in SaaS Cloud

[2]Component service refers a web service in composition.

[3]Sub-optimal utilization denotes the condition of under-/over- utilization of component service capacity.

the composition than the service demand by the tenant. Consequently, the operation cost may outweigh the service revenue which is denoted as the accumulation of debt on the composite service execution. Furthermore, under-/over utilization of component service may incur interest over the debt. For example, over-utilization of service capacity may encounter the SLA violation and the penalty cost against the response time violation could be counted as interest over the technical debt. Here, technical debt indicates the cost of engineering efforts for maintaining end-user SLA.

However, our key idea is that the recomposition can be unnecessary if the under-/over-utilization only occur at a short-term, and the composition plan, although carries debt, can generate greater benefit in the long-term. What makes DebtCom unique is that it takes the long-term benefits of the current composition plan into account, and may therefore temporarily and intentionally accept the negative effects caused by under-/over-utilization, as long as they can be paid off in the long-term. Further, DebtCom provides one with the ability to make trade-off between short-term advantages and long-term benefits via a single value, denoted as k.

In a nutshell, the key contributions of this paper are summarized as follows:

- We propose technical debt as a novel metric for service recomposition and based on that a model, namely service debt that explicitly map the metaphor of technical debt in the contexts of service composition.
- We present an in-depth discussion and reasons that motivate our design of the service debt model, which significantly extends our prior work [7]. Such a model is capable of quantifying both good and bad debt accumulated in service composition.
- We tailor a time-series prediction method, namely ARFIMA model, into the service debt model for predictably learning future debt in composite service execution.
- The proposed service debt model, enhanced by the time-series prediction, allows us to build a utility model based on which an algorithm is proposed with an ability to decide whether to trigger recomposition or not, considering long-term benefits. In particular, the trade-off between short-term advantages and long-term benefits can be controlled by a single value k.
- We combine all components and develop a holistic debt-aware framework for recomposing services in SaaS cloud, namely DebtCom.
- We evaluate DebtCom on a service system in a SaaS cloud with up to 10 abstract services, each of which has 100 component services, under different QoS values derived from the real-world WS-DREAM dataset [23] and FIFA98 workload [25], as well as over ten Docker containers that are of diverse capacity. The results demonstrate the effectiveness of the service debt model and the superiority of DebtCom over the state-of-the-art approaches.

The rest of the paper is organized as follows. Section II presents the background of technical debt and a motivating scenario. Section III describes technical debt in the context of service composition and identify some critical situations in composite service execution environment that significantly contributes the technical debt. Section IV describes the time-series prediction method of service workload in DebtCom. Section V specifies the proposed notion of service debt, which is the core of DebtCom. Section VI illustrates the debt-aware triggers of recomposition in DebtCom, grounded on the time-series prediction and service debt model. Section VII explains the architecture and implementation details of DebtCom. Section VIII discusses the experimental results, particularly with respect to the state-of-the-art. Sections IX, X, and XI specified the threats to validity, related work and conclusion, respectively.

II. Preliminaries

A. Technical Debt

Technical debt is a widely recognized metaphor in software development [8], [9], [11]. Its core idea is to describe the extra cost incurred by actions that compromises long-term benefits of the developed software, e.g., maintainability, in order to gain short-term advantages (e.g. timely software release).

The technical debt metaphor was initially introduced by Cunningham [10] 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 technical trade-off between short-term advantages and long-term benefits in software projects [11].

Intuitively, technical debt makes an analogy with financial debt as described in economics [12]. Often, financial debt is employed to refer to the initial loan and the interest that accumulated over time. In this regards, technical debt leverages the similar concept of principal and interest; for example, the situation in which development team decides to take shortcuts (e.g., by skipping some technical tasks in software development) for getting benefits in terms of releasing timely software product. In this case, technical debt denotes the cost of the fact that some tasks are skipped and interest that may incur due to the extra cost of maintaining the software. Despite the similarities on the concepts, technical debt metaphor is not treated in the same way as the financial debt, because the interest associated with technical debt may or may not be paid off [3], [10]. However, the intuitive nature of technical debt allows the software engineers to reason about the trade-off between the related short-term advantages and long-tern benefits, aiming to make informed decisions based on when (or whether) the technical debt can be paid off [12].

B. Motivating Scenario

As shown in Fig. 1, the SaaS providers lease the IaaS providers infrastructure and deploys several functionally equivalent web

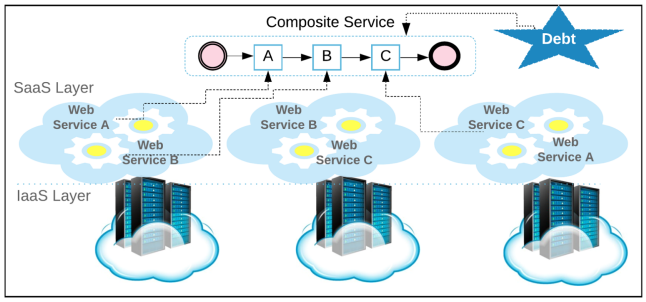


Fig. 1. A service composition scenario.

services into different computing capacities(e.g., container) at IaaS platform. According to an overall workflow of abstract services, the component services are selected and composed together, each of which matches the functional requirement of an abstract service, to form a service composition. The component services for an abstract service implement similar functionalities in the SaaS cloud but offer diverse levels of Quality of Services (QoS). The goal is to improve the QoS of the composite service, since there is a SLA that specific penalty for any violation. At the same time, low operation cost of the service composition is also desirable.

Given the changing workloads from tenants in this context, under-/over-utilization on each of the component services are likely to occur. In particular, under-utilization refers a situation in which a component service capacity is not fully utilized in the composition due to receiving less number of requests workload than request processing capacity (service throughput). As result, it reduces the service revenue and also accumulates the unnecessary debt on the service provider. In contrast, service over-utilization causes SLA violation due to receiving higher requests workload than the service processing capacity. Again, it negatively impacts the service revenue due to paying penalty cost against each request violation. Simply trigger recomposition as soon as over-/under-utilization is detected may provide short-term advantages to resolve the situation to some extends, but it could also hinder the long-term benefits that could have been created by the original composition plan, creating extra operation cost, and more importantly, generating instability. In contrast, doing nothing may suffer the risk that the situation would not change at all. The essential point is that, regardless to the negative effects and accumulated costs, both cases could be accepted as long as the costs can be paid off by benefit in the long-term. However, the fundamental difficulty is how to quantify such cost and benefit, especially taking into account the trade-off between short-/long-term effects.

In this regards, the technical debt metaphor naturally supports intuitive understanding and quantification on the trade-off between short-term advantages and long-term benefits for service recomposition. In particular, the over-/under-utilization caused by the current composition plan can be viewed as debt, which may be temporarily and intentionally accepted as long as they can be cleared and start to create added values by a reasonable point in the long-term. However, the fundamental challenges are to identify what type of technical debt the service composition has (e.g. good or bad)?; how much debt has been incurred? and when it will be paid off for improving overall utility?; and finally to answer the question of when to trigger recomposition? These questions motivate the need of `DebtCom`, a technical debt-aware framework for recomposing services, which we propose in this paper.

III. Technical Debt in Service Composition

During composite service execution, there are many situations when a composite service requires recomposition due to SLA violation, service failure, insufficient service revenue than operating cost (business objective), or QoS fluctuations, etc [24], [34]. We argue that, in this regard, the technical debt could be associated with an inappropriate engineering decision or poorly justified runtime decision of service recomposition that carries short-term advantages in terms of improving instant service utility but not geared for long-term benefits or future value creation. However, a little debt is not always bad if it can help the developers to speed the development process [14]. We look this argument as a valid point in service recomposition for creating long-term values and avoiding unnecessary recomposition that contributes extra overhead and cost. This is of high significance as recomposition comes with the operation cost, especially in SaaS cloud where the underlying resources are leased. Notably, technical debt could be incurred intentionally in service composition when we decide to defer the recomposition decision by taking into account the possibility of generating future values in current service composition. For example, we can accept such debt in a way to consider the future demand for scaling-up the service capacity that transforms the accumulated debt into future value creation. As a result, technical debt-aware decisions saves unnecessary service recomposition cost and improve the composite service utility in the SaaS execution environment.

On the other hand, unintentional technical debt may be the consequences of inappropriate or poorly justified runtime decisions of selecting component services in the recomposition process that produces weak composite service; which fails to process incoming requests workload generated by end-users in the SaaS environment. Consequently, weak composite service violates the end-user SLA and the cost of penalty against each request violation could be count as interest over the incurred technical debt. In this case, unintentional technical debt indicates the cost of efforts required to maintain end-user SLA by recomposing a new service composition plan to get better service utility value in changing request workload or to reduce the debt in current service composition. However, it is a rear condition when all participating component services (web services) in service composition are meeting the full utilization of their capacity. Therefore, technical debt always exists during service composition. Our objectives are to reduce the technical debt and avoid unnecessary recomposition towards improving composite service utility.

IV. Predicting Service Workload

Proactive decision making is not uncommon, especially in the software development context where the concept of technical debt was originally created [17], [18]. Often, the fact of whether a

debt can be paid off depends on the present and future cost of the debt [19]. 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 proactive decision making an debt estimation of recomposition.

To predict the requests workload of each component service in the composition, we adopt a widely used time series model named Autoregressive Fractionally Integrated Moving Average model (ARFIMA) [26] in the `DebtCom` framework because our time-series data contains non-standard time series features such as high volatility and long memory patterns. Moreover, variance changes over time with high or low volatility and long memory may cause slower decay of the autocorrelation function than would be implied by Autoregressive Integrated Moving Average `ARIMA` model [38], [39]. However, ARFIMA model is equipped with both phenomena and guarantees a better prediction accuracy than `ARIMA` [40] as shown in the Table II.

The workload prediction is required to capture an instant fraction of time (e.g., seconds), in which a request is either processed successfully or failed (SLA violation). Therefore, a time-series data (e.g., requests workload data) should handle such time patterns and the length of time-series interval can be adjustable to better fit the estimation. Accordingly, we prepared the data at each time point to contain a number of observed requests at each time interval (e.g., second) and feed this time-series data as an input to the ARFIMA for predicting the future requests workload at every timestep. The general expression of ARFIMA (p, d, q) for the process X_t is written as:

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

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

The prediction process in `DebtCom` is written as ARFIMA (p, d, q) process, in which we estimate the value of memory parameter d using `fdGPH()` function from the R forecast package proposed by Geweka and Porter-Hudak [21]. Specifically, the value of memory parameter d must be between -0.5 and 0.5 that confirms long memory patterns in time-series. The value of p is the autoregressive order that indicates the number of differenced lags appearing in the forecasting equation, and q is the moving average order that shows the number of lagged forecast error in the prediction equation. The values of p and q are identified based on the autocorrelation function and partial autocorrelation function, respectively, as supported by the `fdGPH()` function.

V. Service Debt in Service Composition

To quantify the debt in service composition at SaaS cloud, we adopt the notions of principal and interest [9], [12], [13] from technical debt metaphor into a contextualized model for the analysis. In `DebtCom`, we present a formal model, namely service debt, which connects these notions such that they are made readily available to our problem.

A. Definitions

In the following, we provide an overview of the key definitions that are transformed from the technical debt metaphor into the context of service composition.

Definition 5.1. **Service debt:** The service debt is a transformation of the technical debt concept particularly for the context of service composition in SaaS cloud. Similar to the technical debt, it quantifies the debt incurred for a certain period of time. In particular, it has two major components:

- *Recomposition principal:* This is the one-off cost of the processes that related to recompose a new set of component services.
- *Accumulated interest:* This is the cost of over-/under-utilization caused by workload changes, QoS fluctuation and inappropriate composition plan. The actual cost can be related to the penalty of SLA violation or the rented resources have not been fully utilized.

Definition 5.2. **Good debt:** A good debt is the service debt that will be paid off in the by the time k in the future. Specifically, this can be reflected by the fact that, by time k, the debt has been made smaller or the overall utility has been improved. A good debt often imply that no future recomposition is required.

Definition 5.3. **Bad debt:** Opposed to the good debt, a bad debt is the service debt that will not be paid off by the time k in the future. That is, by time k, there is no sign of improvement on either the service debt and the overall utility. The presence of a bad debt implies that the current composition plan needs to be changed to breakout from the existing situation.

B. Recomposition Principal

In the context of service composition, we use principal to denote the invested cost of recomposing the entire composite service for improving service utility. The principal can be derived from the resources usages, such as the CPU time or the effort spent by software engineer for the decision making of the service composition. Specifically, we compute the principal for recomposing a service using 2.

$$Principal = E \times C_{cpu} \tag{2}$$

Suppose that the recomposition process requires 2 seconds (denoted as E) and the execution cost of CPU is \$0.0025 per second (denoted as C_{cpu}), then it takes a principal as $2 \times 0.0025 = \$0.005$. The time for recomposing the services can be easily known by averaging the time for previous rounds of recomposition.

C. Accumulated Interest

An interest can be accumulated over time on the component service which may be under-utilized or over-utilized. In such context, the interests may be accumulated over time on the yth component service for the xth abstract service (denoted as CS_{xy}). For such a component service, the interests accumulated

from the last recomposition time m to n can be derived from the actual service capacity (i.e., service throughput denoted as T) and the workload at time t (i.e., W_t), which may be the actual workload or predicted one from 1, as shown below:

$$Int(CS_{xy})_{m,n} = \begin{cases} \sum_{t=m}^{n}((T - W_t) \times C) & \text{if } W_t \leq T \\ \sum_{t=m}^{n}\left(\left(\frac{(W_t \times R_{CS_{xy}})}{T} - R_{sla}\right) \times P\right) & \text{otherwise} \end{cases} \quad (3)$$

Clearly, the interests are different depending on two different scenarios of utilizing the capacity of a component service:

- *(a) Service under-utilization:* When the component service is under-utilized, i.e., the workload is smaller than or equals to the capacity of component service ($W_t \leq T$), interest can be calculated as the accumulated cost of unused service capacity. For example, on a component service, suppose that the execution cost of processing each request is \$0.00015 (denoted as C), and a component service has the capacity to process 55 requests per second while the workload on this component service is 48 requests per second. Assuming that the accumulated interests till now is \$1.02, then this component service will carry the interest as \$1.02 + (55-48) $\times$ 0.0015= \$1.0305.
- *(b) Service over-utilization:* When the component service is over-utilized, i.e., the workload is greater than the capacity of component service ($W_t > T$), the SLA requirement on latency (denoted as R_{SLA}) would often be violated [20], if current service latency ($R_{CS_{xy}}$) is larger than the defined SLA latency, and thus a penalty rate (denoted as P) would be used to compute the extra cost to be paid. Suppose again, for a component service, that the accumulated interests till now is \$1.02, and that a given SLA contains the requirement of 2 seconds latency and the penalty rate of latency violation is \$0.0025 per second. Now, assuming that the average service latency, derived from the workload and its capacity, is 3.5 seconds, then the interest would be \$1.02 + (3.5-2.0) $\times$ 0.0025 = \$1.0237. However, the penalty cost would not be counted if the end-user requests workload goes above a certain request rate defined in the SLA.

Finally, from the last recomposition time m to time n, the accumulated service debt (denoted as $D_{m,n}$) of a decision of recomposing the services can be identified and estimated according to the principal and accumulated interests, as shown in 4:

$$D_{m,n} = Principal + \sum_{x=1}^{h} Int(CS_{xy})_{m,n} \quad (4)$$

where h is the total number of abstract services and CS_{xy} is the selected component service (e.g., suppose that it is the yth component service) for the xth abstract service.

VI. Debt-Aware Recomposition

Deriving from the model of service debt, together with the time-series prediction, we developed a technical debt-aware decision approach to recompose services as part of `DebtCom`. In this approach, the service debt model is used to quantify the debt and utility for the period since the last recomposition. The quantified results would be used to compare with the quantification of future debt and utility, supported by the time-series prediction, and thereby taking into account the long term utility. The outcome is then used to trigger the optimization of recomposition plan if there is a needed.

A. Utility Model

To this end, quantifying the utility of service composition is important. To begin with, the revenue and the operation cost, which are fundamental parts in the utility of service composition, can be computed as follows:

$$R(CS_{xy}) = W_t \times C_{tenants} \quad (5)$$

$$C(CS_{xy}) = W_t \times C, \quad (6)$$

whereby W_t indicates the requests workload at time t, $C_{tenants}$ is the charge to the tenants per request, which directly contribute to the revenue generated by the composite service. C is again the cost per request to the SaaS provider for using a component service and its infrastructure.

The utility at the nth timesteps, denoted as U_n, can be calculated as:

$$U_n = \sum_{x=1}^{h} R(CS_{xy}) - \sum_{x=1}^{h} C(CS_{xy}) - D_{m,n}, \quad (7)$$

where h is again the total number of abstract services. In particular, such an equation can measure the actual utility of the service composition at time n, including the service debt accumulated from time m to n. Further, with the support of the time-series prediction, the utility can be used to quantify the future timesteps.

B. Good and Bad Debt

Our debt-aware trigger leverages on the notions of good and bad debt to drive the recomposition. According to our definition about the good and bad debt in Section V, the debt depends on the service utility and service debt over a period of time. In particular, the current service debt from the period between the last recomposition point (time m) and n, denoted as $D_{m,n}$, is good or bad with respect to a future time $n + k$ can be determined as follows:

$$D_{m,n} = \begin{cases} D^{bad} & \text{if } U_n > U_{n+k} \text{ and } D_{m,n} < D_{n,n+k} \\ D^{good} & \text{otherwise} \end{cases} \quad (8)$$

whereby U_n and $D_{m,n}$ is the utility for time n and service debt from the last point of recomposition m to time n, receptively. Similarly, U_{n+k} and $D_{n,n+k}$ are respectively the estimated utility at time $n + k$ and service debt between time n and time $n + k$. When $U_n \leq U_{n+k}$ or $D_{m,n} \geq D_{n,n+k}$, the implication is that the estimated utility and accumulated service debt between n and $n + k$ would become better, therefore the current service debt should be accepted. This is because the any service debts

Algorithm 1: Debt-Aware Recomposition Trigger.

1 **Input**: W_c, W_t, P_n, S
/* W_c:Current workload, W_j:Predicted workload at time j, P_n:current composition plans, S:Set of possible composition plans */
2 **Output**: P_j
/* P_j:Optimized composition plan for time j */
3 **Initialization**: $U_n \leftarrow 0, D_{m,n} \leftarrow 0, U_j \leftarrow 0, D_{n,j} \leftarrow 0, D^{good} \leftarrow 0, D^{bad} \leftarrow 0$
/* U_n and $D_{m,n}$ are the current utility and debt from last point of recomposition time m to current time n; U_j and $D_{n,j}$ are the predicted utility at j and debt from time n to j, where $n < j \leq n+k$; D^{good} and D^{bad} are the counter of good and bad debt, respectively */
4 $D_{m,n} \leftarrow$ calculatetDebt(P_n, W_c)
/* using equation (2), (3) and (4) */
5 $U_n \leftarrow$ calculateUtility($P_n, W_c, D_{m,n}$)
/* using equation (5), (6) and (7) */
6 **for** $j \leftarrow n+k$ ***to*** $n+1$ **do**
7 $D_{n,j} \leftarrow$ calculateDebt(P_n, W_j)
8 $U_j \leftarrow$ calculateUtility($P_n, W_j, D_{n,j}$)
9 **if** ($U_n > U_j$ ***and*** $D_{m,n} < D_{n,j}$) **then**
/* using equation (8) */
10 **if** $j == n+k$ **then**
11 $D^{bad}++$;
12 **end**
13 **if** $j == n+1$ **then**
14 $S'_{demand} \leftarrow \frac{W_j}{j-n}$
15 $P_j \leftarrow$ optimize(S, S'_{demand})
16 **end**
17 **else**
18 **if** $j == n+k$ **then**
/* No recomposition needed */
19 $D^{good}++$;
20 **else**
21 $S'_{demand} \leftarrow \frac{W_j}{j-n}$
22 $P_j \leftarrow$ optimize(S, S'_{demand})
23 **end**
24 **break;**
25 **end**
26 **end**
27 **return** P_j

would be paid off by time $n+k$, leading to an anticipated improvement on the overall utility. Otherwise, the service debt would not be paid off at time $n+k$, in which case another recomposition process should be triggered to seek alternative breakthrough.

C. Trigger and Decision Making of Recomposition

Deriving from the service debt model and time-series prediction, the basic idea of the debt-aware trigger in DebtCom is that if the current debt (at time n) is 'good' with respect to a future point in time, denoted as $n+k$, then no recomposition is needed. Otherwise, a recomposition should be trigger at the furthest future point from time n to $n+k$ by which the current debt is considered as 'good,' as this is the longest period of time before the current debt becomes 'bad'.

The algorithmic procedure of the debt-aware triggers and decision making process have been shown in Algorithm 1, which runs on the next timestep after each recomposition triggered. As can be seen, we calculate the utility since the last recomposition point till now as the current utility, denoted as U_n; the service debt for the same period is denoted as $D_{m,n}$ (line 4-5). Likewise, by leveraging the time-series prediction up to the future timestep $n+k$, the utility at time j ($n < j \leq n+k$) and service debt up to that timestamp can be estimated, denoted as U_j and $D_{n,j}$, receptively (line 7-8).

From the current time n to a future timestep $n+k$, comparing the above utilities and service debt values allow us to verify the need of recomposition based on the future opportunity on value creation at each point in time in the future, as shown at line 9. In particular, if the accumulated debt $D_{m,n}$ is recorded as 'good' with respect to a future timestep $n+k$, then no further action is required (line 17-19) and the loop breaks. Otherwise, the $D_{m,n}$ would be recorded as 'bad' with respect to $n+k$ (line 10-12), in which case the loops continue backwards till time $n+1$, and the recomposition would be triggered at the furthest timestep based on which the $D_{m,n}$ is considered as 'good' (line 20-23). If all the timesteps between n and $n+k$ would make $D_{m,n}$ 'bad,' then the recomposition happens at the next timestep $n+1$ (line 13-15). Here, the actual recomposition process is based on search-based evolutionary optimization, as derived from our previous work [22].

It is worth noting that, the k value controls the preference between short-term advantages and long-term benefit. A larger k implies stronger preference towards long-term benefit, in which case it is likely that less number of recompositions is required but could intentionally accept more bad debt. On the other hand, smaller k favors short term advantages by taking relatively immediate recomposition, which could hinder the benefits in long-term and generate too much operation cost. Indeed, it is possible that the benefit of DebtCom can be related to k, and therefore in Section VIII-H, we experimentally examine the sensitivity of DebtCom to k in terms of both the utility and running time.

VII. Architecture of DebtCom

This section presents the architecture of DebtCom for debt-aware service composition in SaaS cloud. This architecture is designed into three hierarchical levels: *runtime management level*, *Service execution level*, and *Back-end process and data repositories level* as shown in Fig. 2. Briefly, in the following, we discuss the general interaction between the components of these levels.

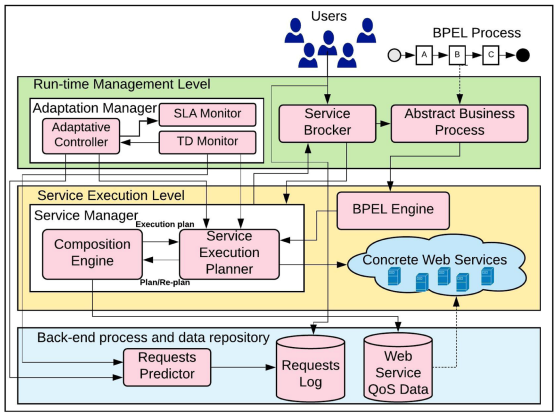


Fig. 2. The architecture of DebtCom.

A. Runtime Management Level

In the SaaS cloud, the end-users can be distinguished based on what type of SLA they have retained, which is often handled by the *Service Broker*. At this level, the services offered by SaaS provider have been designed into a *Abstract Business Process*, which is written in Business Process Execution Language (BPEL) [21]. Moreover, abstract business workflow represents the interaction of services in the composition structure. At runtime, *Broker* invokes the business process workflow component based on the request type.

The *Adaptation Manager* is the core component where our debt-aware trigger is implemented. In particular, it is responsible for carrying out runtime adaptation action (e.g. service recomposition) while determining the need of a new composite service plan in order to meet the end-users requirements. As shown in Fig. 2, *Adaptation Manager* comprises three sub-components: *Adaptive Controller*, *SLA Monitor* and *TD Monitor*. Here, the *SLA Monitor* observes the runtime behaviour of service composition and proactively captures dynamic changes (e.g., workload or arrival/departure of users in service pool) in the execution environment that may contribute to SLA violation.

Similarly, the *TD Monitor* examines the running service composition from a debt point of view by using the service debt model discussed in Section V. In particular, the *TD Monitor* continuously observes the accrued debt and service utility carried by a currently executed service composition plan. Furthermore, it interacts with the *Request Predictor* from the lowest level, supported by time-series prediction, for monitoring the predicted workload over the composite service, based on which the proactive estimation of the future service debt and utility becomes possible.

Finally, the *Adaptive Controller* takes the information from the *SLA Monitor* and *TD Monitor*, after which it determine whether to trigger the recomposition based on the the approach discussed in Section VI.

B. Service Execution Level

When a recomposition is indeed required, this level enables runtime decision making for optimizing the composition plan and invokes component services in the service composition.

The *BPEL Engine* is a software platform (e.g., WSO2 BPS [41]) that executes the business process, which represents the composite service produced by the *Service Execution Planner* as requested by the *Service Broker*. The *Service Execution Planner* also dynamically binds the end-users' request to the endpoint that exposes the service operation. The service endpoints are identified and selected from the *Concrete Web Service* pool based on the service composition plan generated by the *Composition Engine*.

In DebtCom, we design the *Composition Engine* based on our prior work [22], which is an evolutionary algorithm based optimization approach. It is worth noting that the *Composition Engine* is triggered only when the *Adaptation Manager* from the above level requires a new service composition plan.

C. Back-End Process and Data Repository Level

Here, as discussed in Section IV, the *Request Predictor* examines the past patterns of requests workload (*Requests Log*) generated by the system and predicts the requests workload over the executed composite service. The QoS values of each component service are stored in the *Web Service QoS Data* repository (e.g., WS- Dream QoS dataset [23]).

VIII. Experimental Evaluation

In this section, our goal is to assess the effectiveness of DebtCom in contrast to the baseline and state-of-the-art approaches for service composition in SaaS cloud. Further, we seek to understand whether the individual components of DebtCom, i.e., the time-series prediction and the debt-aware trigger, can indeed create benefit. Specifically, our experiments aim to answer the following research questions.

- **RQ1:** How accurate does DebtCom predict the workload?
- **RQ2:** Whether DebtCom can outperform the traditional baseline approach?
- **RQ3:** In contrast to the state-of-the-art, whether the time-series prediction and the debt-aware trigger in DebtCom can create benefit individually?
- **RQ4:** What is the running overhead of DebtCom?
- **RQ5:** What is the sensitivity of DebtCom to the k value?
- **RQ6:** How DebtCom can help improving the response time?

A. Experimental Setup

For experimental purpose, we developed an e-commerce application, which is formed as a service composition where there are 10 abstract services connected by *sequential* and *parallel* connectors. To emulate an environment of SaaS cloud, an abstract service can select 100 component services, each of which is deployed over 10 Docker containers with different capacities. Each of the component service exhibits different QoS values, which are randomly chosen from the WSDream [23]. The relevant setups of the subject service systems have been shown in Table I, which are the results of several runs of trial-and-error and tend to be the most reasonable settings as we observed in our experiment runs.

TABLE I
EXPERIMENTS PARAMETERS

Parameters	*Numerical Value*
C_{cpu}: Recomposition cost (e.g., engineering efforts plus CPU execution cost)	0.0025 ($)
C: Per request execution cost	0.0015 ($)
$C_{tenants}$: Per request tenant's cost	0.0025 ($)
P: Penalty per request	200% of its cost
R_{SLA}: Response time (SLA)	1 seconds

To emulate realistic workload for each component service, we extract the FIFA98 World Cup website trace [25] for the length of 6 hours, which forms the workload dataset. We pre-processed the first 4 hours workload trace as the samples for training the prediction model, while the remaining 2 hours workload data, which equals to 7200 seconds, is used for testing the accuracy. In DebtCom, we feed the training data into the ARFIMA, which is implemented using the ARFIMA package in R [26]. In all experiments, the k, which determines how many future timesteps to be predicted, is set to 5. This means that DebtCom predicts the service debt associated with 5 timestep ahead and take it into account when deciding whether to trigger recomposition. Notably, the k value of 5 is the most ideal trade-off between short-term advantages and long-term benefits, achieving the best utility as discussed within the the sensitivity analysis of DebtCom in Section VIII-H. Note that we use a sampling interval of 1 s in our experiments.

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

B. Compared Baseline and State-of-The-Art

According to the literature, we compare DebtCom with three different approaches for service composition in SaaS cloud. They are specified as follows.

- **Baseline:** We implemented a traditional service recomposition approach from the literature as the baseline [24]. In the approach, a neighborhood region of component services is predefined for each abstract services. The recomposition occurs whenever the violation of SLA has been detected, after which an exhaustive search is conducted to find the best composition plans based on different neighborhood regions, which forms a relatively small search space.
- **Passive:** Another state-of-the-art method that triggers the recomposition based upon the detection of SLA violation [6]. In this work, to achieve a fair comparison, we have applied the evolutionary optimization approach to search the composition plans, which are equivalent to DebtCom.
- **Proactive:** This is the state-of-the-art method that triggers recomposition when the workload is predicted to cause SLA violation [28]. However, the debt model is not explicitly captured during the triggering process. Again, we use the same ARFIMA for workload prediction as DebtCom. Further, similar to Passive, the acutal optimization mechanism is also the same.

TABLE II
ACCURACY OF TIME-SERIES PREDICTION FOR WORKLOAD

Method	*MEA*	*RMSE*	*Theil Coefficient*
ARFIMA	4.0190	5.0817	0.5732
ARIMA	5.1403	8.4315	0.7402

C. Metrics

We have used the standard metrics such as accuracy, utility and running overhead [24], [28], [31] for evaluating the performance of DebtCom. Further, to conduct the rigorous analysis of DebtCom's performance we have chosen the most appropriate metrics according to experiment design.

- *Accuracy:* By using Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE), we assess the accuracy of the ARFIMA in DebtCom for predicting workload of component services.
- *Utility:* For all approaches, we plot the utility of service composition over all timesteps, using (7). We show both the individual value (i.e., for each timestep) and accumulated result throughout the time series.
- *Service Debt:* We examine the service debt incurred for all timesteps by using Equation (4). Again, we plot both the value for each timestep and accumulated result throughout the time series.
- *Operation Cost:* We assess the resulted cost for all timesteps, using Equation (6). Simlar to the others, we plot both the value for each timestep and accumulated result throughout the time series.
- *Good/Bad Debt Count:* We count the number of good and bad debt produced by the approaches.
- *Running Overhead:* We evaluate the running overhead of the approaches in terms of the required running time.
- *Response Time:* We examine the service response time yield by all approaches.

D. RQ1: Accuracy on Workload Prediction

To answer **RQ1**, we plot the mean accuracy when DebtCom predicts the workload for all component services using different metrics, as shown in Table II. Further, we evaluate the prediction accuracy of our DebtCom prediction model namely ARFIMA [26] by comparing the results obtained from the state-of-the art ARIMA model [40] in Table II. Considering that the general workload varies between around 35 and 60 requests per second, the MAE and RMSE are in fact relatively low in ARFIMA model and thus the accuracy is acceptable. In addition, we also report on the Theil's coefficient, which indicates a good prediction if it lies between 0 and 1; or the prediction is deemed as poor otherwise [29]. As can be seen, the resulted Theil's coefficient is in between 0 and 1, and thereby suggesting a sufficient accuracy.

As a more detailed example, Fig. 3 illustrates the workload trace for a selected component service. As we can see, although there are some deviations between the predicted and the actual workload, the prediction has been able to capture the general

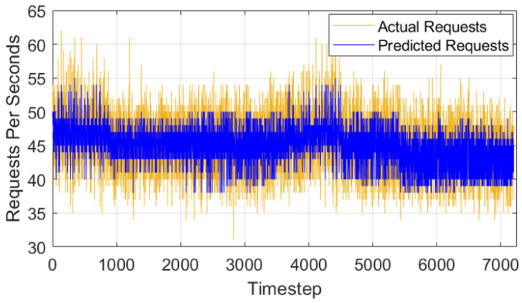


Fig. 3. Predicted and actual workload on the component services.

TABLE III
IDENTIFIED GOOD AND BAD DEBT

Debt Type	*DebtCom*	*Baseline*	*Debt Impacts*
Total Good Debts	1417	1132	Improving overall service utility
Total Bad Debts	983	1268	Degrades overall service utility

pattern of trace of, e.g., the spike between 4000 and 5000 s point. We therefore conclude that:

Answering RQ1: The workload prediction of ARFIMA in DebtCom is sufficiently accurate, as the deviation is small and the pattern of trace can be generally captured.

E. RQ2: Results of DebtCom *Against* Baseline

To investigate **RQ2**, we compare DebtCom with the Baseline approach as discussed in Section VIII-B. In particular, we assess their utility, service debt and operation cost for recomposing the services under the testing period of the workload.

Fig. 4 shows the accumulated utility, service debt and operation cost of both approaches. As we can see, in contrast to Baseline, DebtCom performs significantly better on reducing the accumulated debt while keeping less accumulated operation cost. This has in turn, leading to considerably better result on the accumulated utility. Notably, the improvement of DebtCom on the utility and debt can be achieved with even less operation cost. To conduct a more detailed review, in Figs. 5 and 6, we illustrate the utility, service debt and operation cost measured at each timestep. It is clear to see that DebtCom outperforms Baseline on every timestep in terms of the operation cost. As for the service debt and overall utility, DebtCom is only slightly better before the point of 4000 s, because of the fact that the workload fluctuation till that point is relatively light. However, following the spike between 4000 and 5000 s points, the superiority of DebtCom becomes much more obvious, as the service debt and utility are both significantly improved for the long term. All the above evidence the ability of DebtCom to handle sudden changes, especially for making decision of wether to recompse that takes the long term benefits into account.

In Table III, we compare the number of good/bad service debt achieved by DebtCom and Baseline. To achieve a fair comparison for Baseline, we assess its debt only on the timesteps that DebtCom has checked whether the current debt is good or bad, and thereby the total number of debt to be compared is equivalent. As can been seen, Baseline produce more bad debt than the good ones, i.e., 1132 against 1268; while DebtCom achieves 1417 good debt, which is around 44% more than the 983 bad debt. This is a significant improvement, as higher number of good debt implies that DebtCom requires less number of recomposition, as each composition plan is more sustainable, thanks to the awareness of debt enabled by our service debt model. To conclude, we can summarize that:

Answering RQ2: DebtCom performs significantly better than Baseline on the utility, service debt and operation cost. The benefits cover not only the accumulated results, but also the outcome of each individual timestep. In addition, DebtCom is more robust to the sudden spike in the workload, providing much more good debt than the bad ones, as it takes the long term benefits into account when recomposing the services in SaaS cloud. Noteworthily, the benefit of DebtCom can be produced with less cost/number of recomposition, as each composition plan is more sustainable.

F. RQ3: Effectiveness of Workload Prediction and Debt-Aware Trigger in DebtCom *Against State-of-The-Art*

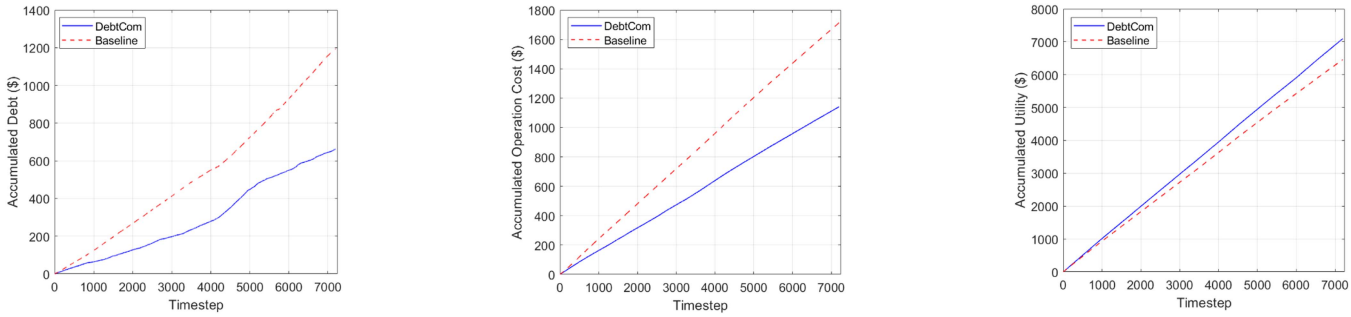
To assess the effectiveness of workload prediction and debt-aware trigger, which are the core components in DebtCom, we compare the results between Passive and Proactive, as well as those between Proactive and DebtCom. From Fig. 8, we see that Proactive achieves better utility with less service debt than that of Passive. This indicates that the workload prediction is indeed beneficial to the service composition. It is also obvious that DebtCom outperforms Proactive on both metrics, which proves that the debt-aware trigger, supported by the service debt model and workload prediction, create greater benefit in the long term.

When observing the result for each timestep, as shown in Fig. 7, similar conclusion can be drawn. In particular, the ability of prediction in Proactive makes it robust to the spike after 4000 s point, but the fact that it only aims for short time benefit has caused a few suddenly increased debt (sudden drop on the utility). In contrast, DebtCom does not suffer such issue, thanks to the service debt model. As a result, we conclude that:

Answering RQ3: Both the workload prediction and debt-aware trigger in DebtCom are effective in reducing the service debt, leading to better utility of service composition in SaaS cloud in contrast to the state-of-the-art methods.

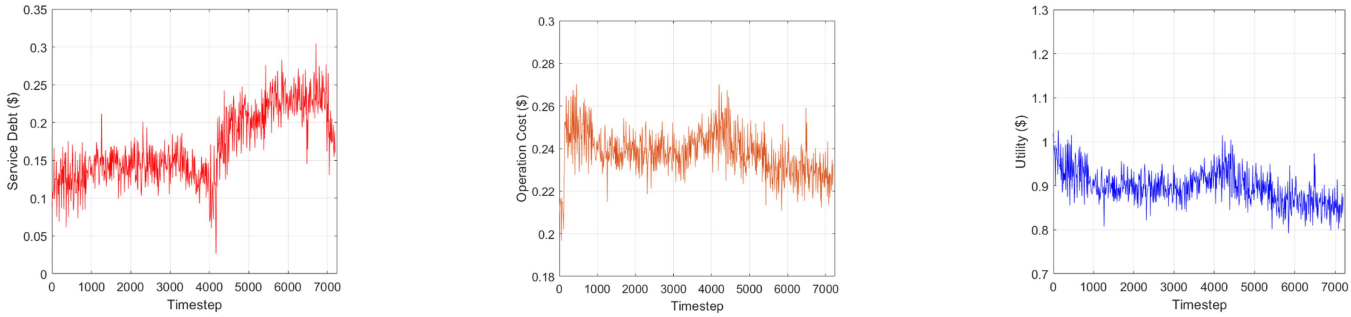
G. RQ4: Running Overhead of DebtCom

To understand **RQ4**, we evaluate the running overhead of the decision making process in DebtCom, including both the reasoning process of service debt and the optimization process



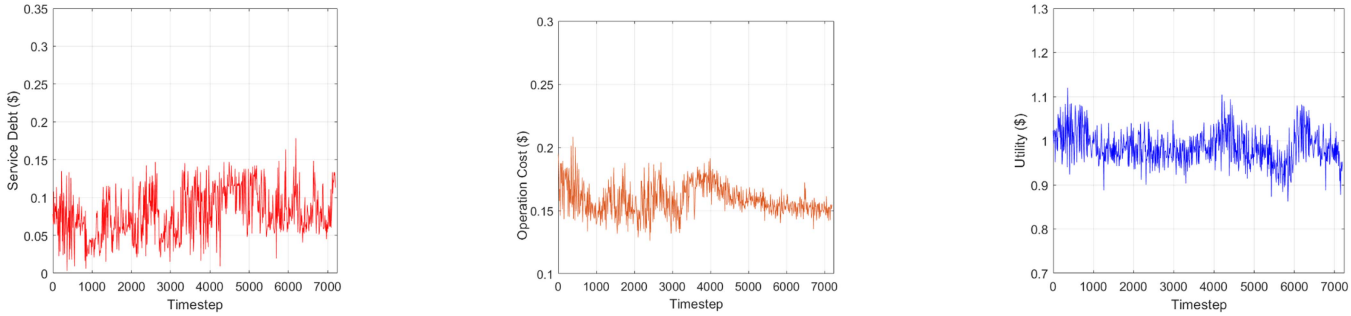
(a) Accumulated debt (b) Accumulated operation cost (c) Accumulated utility

Fig. 4. Accumulated debt, accumulated operation cost and accumulated utility achieved by `DebtCom` and `Baseline` over all timesteps.



(a) Service debt (b) Operation cost (c) Utility

Fig. 5. Serivce debt, operation cost and utility achieved by `Baseline` over all timesteps.



(a) Service debt (b) Operation cost (c) Utility

Fig. 6. Service debt, operation cost and utility achieved by `DebtCom` over all timesteps.

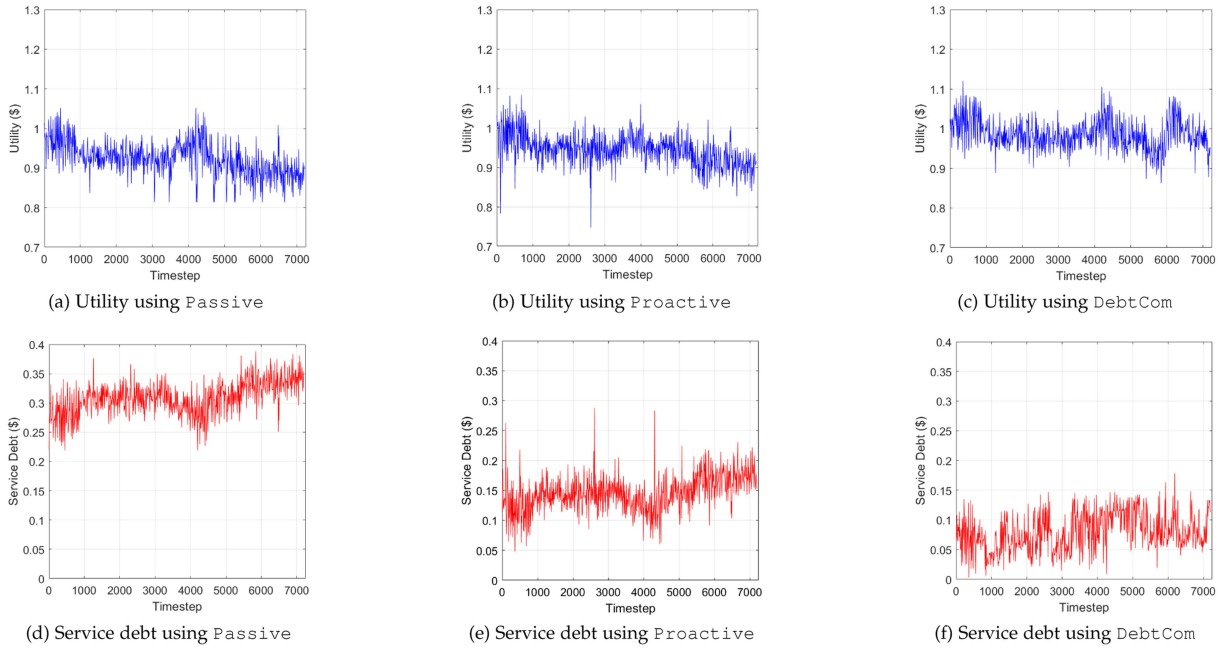
that find the actual composition plan. We proceed such by comparing `DebtCom` with `Baseline`. To this end, we run both approaches for 30 times and report on the minimum, average and maximum time required.

As shown in Fig. 9, we see that `DebtCom` clearly runs faster than `Baseline`. Although the margin differs in the scale of milliseconds, it is worth noting that the need of recomposition can be rapid in service composition at SaaS cloud, which implies a matter of 10 ms faster can be seen as a considerable improvement. For **RQ4**, our answer is that:

Answering RQ4: `DebtCom` runs considerably faster than `Baseline` when recomposing services in SaaS cloud.

H. RQ5: Sensitivity of `DebtCom` *to* k *Value*

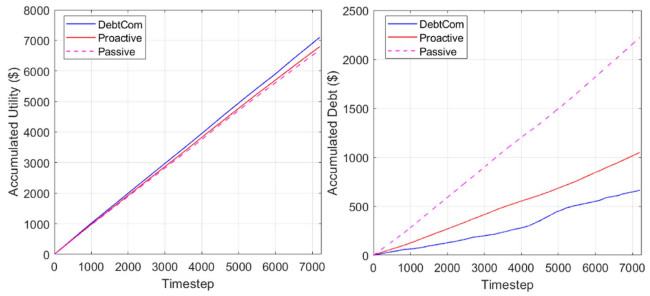
To answer **RQ5**, we compare the utility and running time of `DebtCom` under five different k values that represent different preference between short-term advantages and long-term benefits. In particular, we repeat 120 runs for assessing running time and 7200 timesteps for the utility. The boxplots of the results are shown in Fig. 10. As can been seen, `DebtCom` can be indeed sensitive to the k value, in which $k = 5$ tends to be the optimal setting, but neither the utility nor the running time exhibit clear monotonic trace. The results also suggest that both too small and too large k could be harmful, as they failed to gain long-term benefits and accept too much bad debt, respectively.



(a) Utility using `Passive` (b) Utility using `Proactive` (c) Utility using `DebtCom`

(d) Service debt using `Passive` (e) Service debt using `Proactive` (f) Service debt using `DebtCom`

Fig. 7. Utility and service debt achieved by `Passive`, `Proactive` and `DebtCom` over all timesteps.



(a) Accumulated utility (b) Accumulated debt

Fig. 8. Accumulated utility and accumulated debt achieved by `Passive`, `Proactive` and `DebtCom` over all timesteps.

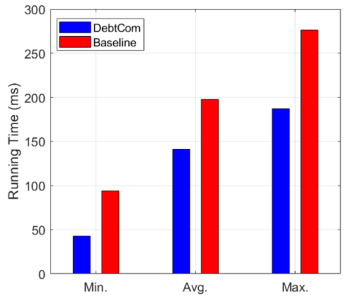
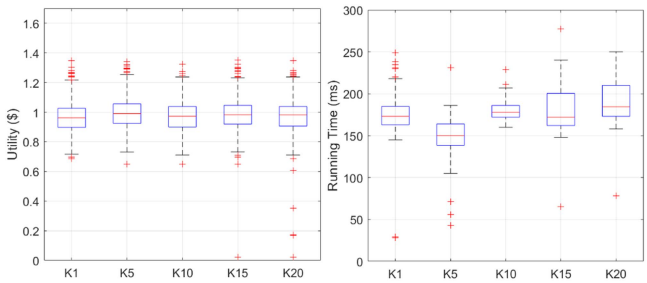


Fig. 9. Running time.

Clearly, although the case of $k = 5$ is better than the others on both metrics, their margins tend to be relatively small. Therefore, to confirm the statistical significance on the sensitivity of `DebtCom` to k value, we perform Kruskal Wallis test with Bonferroni correction[4] and calculate the η^2 as the effect size, which can be interpreted following the guidance

[4]We use. 05 as the significance level, which becomes. 005 after correction.



(a) Utility
$p < .001$; $\eta^2 = .013$ (small)

(b) Running time
$p < .001$; $\eta^2 = .294$ (large)

Fig. 10. Sensitivity of `DebtCom` to k values in terms of utility and running time.

by Tomczak and Tomczak [30]. The results reveal that the sensitivity of `DebtCom` to k value is statistically significant on both utility and running time, with $p < .005$ and non-trivial effect size. In conclusion, the answer for **RQ5** is that:

> **Answering RQ5:** `DebtCom` is indeed sensitive to the k value in terms of both utility and running time, with statistical significance and non-trivial effect size. In particular, both sets of sensitivity exhibits non-monotonic traces, which implies potentially complex trade-off between short-term advantages and long-term benefits.

I. RQ6: Comparison of Service Response Time

Since the response time can be a critical QoS attribute to consider, we also compare all the approaches under such a metric. As shown in the box-plots of Fig. 11, when comparing the response time, `DebtCom` has a smaller variance than

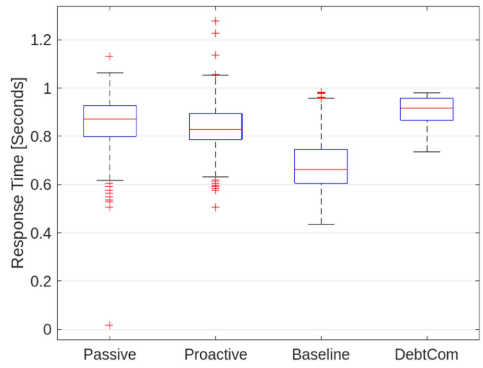


Fig. 11. Comparing response time QoS metric.

Passive and Proactive, and also achieves better response time. However, Passive and Proactive approaches have a better mean response time but violate the SLA.[5] Further, Baseline outperforms other three approaches but it accumulates higher operating cost than DebtCom as shown in Fig. 4(b).

Answering RQ6: Overall, DebtCom achieves stable response time on the lower operating cost when compare other state-of-the-art approaches have higher variance.

J. Summary

In answering all these research questions, we investigated the effectiveness of the DebtCom framework by evaluating the core components, such as the prediction model and service debt model. We have also used a variety of metrics to make a fair comparison between the approaches. In particular, we examined the importance of workload prediction in DebtCom by comparing the passive approach that cannot make a predictive recomposition decision and only react whenever an SLA violation is detected. The results discussed in RQ3 show that DebtCom outperforms the passive approach on all the metrics. Further, we used the same workload prediction model in the Proactive approach and DebtCom. After that, we examined what brings the predictive model by integrating it into the service debt model for making a proactive debt-aware decision for recomposing the service. From the results in Figs. 7 and 8, we observed that DebtCom produced higher service utility and accumulated less debt than the Proactive approach. Similarly, in RQ2 and RQ4, we assess the DebtCom performance against the Baseline approach and running overhead by executing these approaches several times on the same experimental setup. Apart from that, we examined the sensitivity of DebtCom controlled by a numerical value k, which is the number of forecasted timesteps monitored by DebtCom for making proactive recomposition decisions. We estimated multiple timesteps and identified the best case value of k as discussed in RQ5.

[5]One second is the maximum service response time defined in the SLA.

TABLE IV
COMPARISON TO RELATED WORK

Parameters	[5]	[6]	[24]	[33]	[34]	DebtCom
QoS Attributes (e.g., Th, RT and Cost etc.)	√	√	√	√	√	√
Proactive Recomposition	×	×	×	√	√	√
Economic-Driven Perspective	×	×	×	×	×	√
Framework	×	×	√	×	√	√

IX. Threats to Validity

Threats to construct validity are used to determine whether the metrics can undoubtedly reflect what we aim to measure. In this paper, we set up our experiments with a broad range of metrics for evaluating different aspects of DebtCom, including accuracy, utility (e.g., generated revenue), operation cost, amongst others.

Threats to internal validity can be mainly related to the value of the parameters for the DebtCom. Particularly, the setup has been designed in a way that it produces good trade-off between the quality of services composition and the recomposition overhead. Further, threats to internal validity could be related to the randomness of the results obtained from different runs. Indeed, the actual optimization of recomposition plan is achieved by using our prior work [22], which relies on stochastic algorithm. To mitigate such, we repeat all the experiments across different timesteps and runs. We have also assess the sensitivity of DebtCom to the k value, which determines the preference between short-/long-term benefits, based on statistical analysis and effect sizes.

Threats to external validity can be associated with the testing environment and the dataset that are used in this experiment. To improve generalization of the results, we developed a real-world ecommerce system as a testing environment, with up to 10 abstract service, each of which has possible 100 component services and 10 dockers. Further, DebtCom has been evaluated on the reald-world WSDream dataset [23] and FIFA98 workload trace [25].

X. Related Work

Over decades, different approaches have been presented for reconfiguring the composite service with QoS requirements [5], [6], [24], [31], [32], [36], [37]. Among others, most of the work is based on passive recomposition. For example, Lin et al. [24] described a multi-steps algorithmic approach; in which an expand region algorithm is proposed that identifies the reconfiguration region of faulty services, each of which would be recomposed locally. However, recomposition is triggered upon the detection of SLA violation. Canfora et al. [31], [32] present algorithm that continuously monitors the QoS of composite service and triggers the service recomposition once the SLA is violated. To address QoS constraints, Ren et al. [42] studied QoS uncertainty and service behavior in the dynamic environment, where constraint-satisfied service composition is formulated as Markow Decision Process (MDP) and solved using a Q-learning algorithm. Chattopadhyay et al. [43] presented a graph-based

abstraction refinement approach that explores the substantially smaller space for constructing a complete service composition graph. First, they created a set of abstractions and corresponding concrete service refinements to form the service groups. Afterwards, a dependency graph is generated to compose the service by selecting a representative service from each group.

Beyond deterministic optimization, evolutionary optimization based approaches have been presented by Chen et al. [36], [37], they seed the multi-objective evolutionary algorithm with pre-selected solutions to optimize service composition. The recomposition is triggered in every timestep, aiming to maintain optimality throughout the lifecycle of service composition. However, the passive nature of triggering recomposition has limited the ability to proactively response to changes. As we have shown, acting proactively to likely changes in DebtCom would create extra benefits and improve the overall utility.

Indeed, workload prediction enables proactive service recomposition. In literature, many cloud workload prediction methods have been discussed [44], [45], [46], [47].for example, Calheiros et al. [44] utilized the ARIMA method for predicting the HTTP requests workload on the cloud application. However, Deep Learning-based prediction models such as Recurrent Neural Network (RRN) perform poorly on the time-series data containing long-term memory patterns and high volatility [45]. Further, there is work that have been conducted to achieve proactive recomposition of services [28], [33], [34], [35]. For example, Dai et al. [33] presented a self-healing approach for service composition. They rely on performance prediction to trigger recomposition. Aschoff et al. [34] presented a ProAdapt framework for proactive adaptation of service composition due to changes in composite service. They used the exponential weighted moving average (EWMA) that models the response time of service operation, which would then trigger recomposition when likely degradation of response time is detected. Nevertheless, those approaches do not take the concept of technical debt into account. This means that they do not have the ability to reason about short-term and long-term benefits, blurring the potential effectiveness of each recompositions less sustainable. Indeed, as we have shown, explicitly considering technical debt in DebtCom help to achieve much better utility in the long-term, which make the composition plan more sustainable. In Table 4, we provide the comparative summary of closely related works.

XI. Conclusion

In this paper, we propose a technical debt aware framework, namely DetbCom, for service recomposition in SaaS Cloud. In particular, we transform the notion of technical debt metaphor to form a new model called service debt, which fits explicitly within the context of service composition. Such a service debt model, together with time-series prediction using ARFIMA, forms a utility model based on which an algorithm is designed to make decision about when to trigger recomposition. Experiments have been conducted under a large scale service system based on real-world dataset and workload trace. The results confirm the superiority of DetbCom over the state-of-the-art, such that better overall utility can be obtained with less number of recomposition required, which implies that each composition plan becomes more sustainable.

References

[1] Y. Chen, J. Huang, C. Lin, and X. Shen, "Multi-objective service composition with QoS dependencies," *IEEE Trans. Serv. Comput.*, vol. 7, no. 2, pp. 537–552, Apr.-Jun. 2019.

[2] Q. He et al., "QoS-Aware service selection for customisable multi-tenant service-based system: Maturity and approaches," in *Proc. EEE Int. Conf. Cloud Comput*., 2012, pp. 566–573.

[3] T. Chen and R. Bahsoon, "Self-adaptive and online QoS modeling for cloud-based software services," *Trans. Softw. Eng.*, vol. 43, no. 5, pp. 453–475, May 2017.

[4] Y. Yan, P. Poizat, and L. Zhao, "Repair vs recomposition for broken service compositions," in *Proc. 8th Int. Conf. Service-Oriented Comput.*, 2010, pp. 152–166.

[5] Y. Li, Y. Lu, Y. Yin, S. Deng, and J. Yin, "Towards QoS-Based dynamic reconfiguration of SOA-Based application," in *Proc. IEEE Asia-Pecific Serv. Comput. Conf.*, 2010, pp. 107–114.

[6] F. Boudries, S. Sadouki, and A. Tari, "A bio-inspired algorithm for dynamic reconfiguration with end-to-end constraintes in web service composition," *Service Oriented Comput. App.*, vol. 13, no. 3, pp. 251–260, 2019.

[7] S. Kumar, R. Bahsoon, T. Chen, and R. Buyya, "Identifying and estimating technical debt for service composition in SaaS cloud," in *Proc. IEEE Int. Conf. Web Serv*., 2019, pp. 121–125.

[8] F. Buschmann, "To pay or not to pay technical debt," *IEEE Soft.*, vol. 28, no. 6, pp. 29–31, Nov./Dec. 2011.

[9] E. Tom, A. Aurum, and R. Vidgen, "An exploration of technical debt," *J. Syst. Soft.*, vol. 86, no. 6, pp. 1498–1516, 2013.

[10] W. Cunningham, "The WyCash portfolio management system," in *Proc. Conf. Object-Oriented Program. Syst. Lang. Appl.*., 1992, pp. 29–20.

[11] P. Avgerious, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing technical debt in software engineering," *Dagstuhl Rep.*, vol. 6, no. 4, pp. 110–138, 2016.

[12] N. S. R. Alves, T. S. Mendes, M. G. Mendonca, R. O. Spinola, F. Shull, and C. Seaman, "Identification and management of technical debt: A systematic mapping study," *Inf. Soft. Tech.*, vol. 70, pp. 100–121, 2016.

[13] A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "The financial aspect of managing technical debt: A. systematic literature review," *Inf. Soft. Tech.*, vol. 64, pp. 52–73, 2015.

[14] R. Marinescu, "Assessing technical debt by identifying design flaws in software systems," *IBM J. Res. Dev.*, vol. 56, no. 5, pp. 9–13, 2012.

[15] C. Seaman et al., "Using technical debt data in decision making: Potential decision approaches," in *Proc. 3rd Int. Workshop Manag. Tech. Debt*, 2012, pp. 45–48.

[16] M. Foucault, M. A. Storey, X. Blanc, J. R. Falleri, and C. Teyton, "Gamification: A game changer for managing technical debt? A design study," 2018, *arXiv:1802.02693*.

[17] Y. Guo, R. O. Spinola, and C. Seaman, "Exploring the costs of technical debt management- a case study," *Emp. Soft. Eng.*, vol. 21, no. 1, pp. 159–182, 2016.

[18] Z. Codabux and B. J. Williams, "Technical debt prioritization using predictive analytics," in *Proc. IEEE/ACM 38th Int. Conf. Soft. Eng. Comp*., 2016, pp. 704–706.

[19] W. Snipes, B. Robinson, Y. Guo, and C. Seaman, "Defining the decision factors for managing defects: A technical debt perspective," in *Proc. 3rd Int. Workshop Manag. Tech. Debt*, 2012, pp. 54–60.

[20] T. Chen, R. Bahsoon, S. Wang, and X. Yao, "To adapt or not to adapt? Technical debt and learning drivent self-adaptation for managing runtime performance," in *Proc. IEEE/ACM Int. Conf. Perform. Eng.*, 2018, pp. 48–55.

[21] J. Pasley, "How BPEL and SOA are changing web services development," *IEEE Int. Comput.*, vol. 9, no. 3, pp. 60–67, May/Jun. 2005.

[22] S. Kumar, R. Bahsoon, T. Chen, K. Li, and R. Buyya, "Multi-tenant cloud service composition using evolutionary optimization," in *Proc. IEEE 24th Int. Conf. Parallel Dist. Syst*., 2018, pp. 972–989.

[23] Z. Zheng, Y. Zhang, and M. R. Lyu, "Investigating QoS real-world web services," *IEEE Trans. Serv. Comput.*, vol. 7, no. 1, pp. 32–39, Mar. 2014.

[24] K. J. Lin, J. Zhang, Y. Zhai, and B. Xu, "The design and implementation of service process reconfiguration with end-to-end QoS constraints in SOA," *Service Oriented Comput. App.*, vol. 4, no. 3, pp. 157–168, 2010.

[25] M. Arlitt and T. Jin, "A workload characterization study of the 1998 world cup web site," *IEEE Net.*, vol. 14, no. 3, pp. 30–37, May/Jun. 2000.

[26] K. Liu, Y. Q. Chen, and X. Zhang, "An evaluation of ARFIMA (autoregressive fractional integral moving average) programs," *Axioms J.*, vol. 6, no. 2, 2017, Art. no. 16.
[27] J. Veenstra and A. I. Mcleod, "Package 'arfima' version-1.7–0," 2018. [Online]. Available:https://cran.r-project.org/web/packages/arfima/arfima.pdf
[28] X. Sun et al., "A fluctuation-aware approach for predictive web service composition," in *Proc. IEEE Int. Conf. Serv. Comput.*, 2018, pp. 121–128.
[29] F. Bliemel, "Theil's forecast accuracy coefficient: A clarification," *J. Mark. Res.*, vol. 10, no. 4, pp. 444–446, 1973.
[30] M. Tomczak and E. Tomczak, "The need to report effect size estimates revisited. an overview of some recommended measures of effect size," *Trends Sport Sci.*, vol. 21, no. 1, pp. 19–25, 2014.
[31] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani, "QoS-Aware replanning of composite web services," in *Proc. IEEE Int. Conf. Web Serv.*, 2005, pp. 121–129.
[32] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani, "A framework for QoS-Aware binding and re-binding of composite web services," *J. Syst. Soft.*, vol. 81, no. 10, pp. 1754–1769, 2008.
[33] Y. Dia, L. Yang, and B. Zhang, "QoS-Driven self-healing web service composition based on performance prediction," *J. Comp. Sci. Tech.*, vol. 24, no. 2, pp. 250–261, 2009.
[34] R. Aschoff and A. Zisman, "QoS-Driven proactive adaptation of service composition," in *Proc. Conf. Service-Oriented Comput.*, 2011, pp. 421–435.
[35] J. Li, D. Ma, X. Mei, H. Sun, and Z. Zheng, "Adaptive QoS-aware service process reconfiguration," in *Proc. IEEE Int. Conf. Serv. Comput*., 2011, pp. 282–289.
[36] T. Chen, M. Li, and X. Yao, "On the effects of seeding strategies: A case for search-based multi-objective service composition," in *Proc. Genet. Evol. Comput. Conf*., 2018, pp. 1419–1426.
[37] T. Chen, M. Li, and X. Yao, "Standing on the shoulders of giants: Seeding search-based multi-objective optimization with prior knowledge for software service composition," *Inf. Soft. Tech.*, vol. 114, pp. 155–175, 2019.
[38] M. A. Hauser and R. M. Kunst, "Forecasting high-frequency fi-nancial data with the ARFIMA-ARCH model," *J. Forecast.*, vol. 20, pp. 501–518, 2001.
[39] T. Bollerslev and J. H. Wright, "High-frequency data, frequency do-main inference, and volatility forecasting," *Rev. Econ. Statist.*, vol. 83, pp. 596–602, 2001.
[40] J. Contreras, R. Espinola. F. J. Nogales, and A. J. Conejo, "ARIMA models to predict next-day electricity prices," *IEEE Tran. Pow. Sys.*, vol. 18, pp. 1014–1020, Aug. 2003.
[41] M. Pathirange, S. Perera, I. Kumara, and S. Weerawarana, "A multi-tenant architecture for business process execution," in *Proc. IEEE Int. Conf. Web Serv.*, 2011, pp. 264–271.
[42] L. Ren, W. Wang, and H. Xu, "A reinforcement learning method for constraint-satisfied services composition," *IEEE Trans. Serv. Comput.*, vol. 13, no. 5, pp. 886–800, Sep./Oct. 2020.
[43] S. Chattopadhyay and A. Banerjee, "QoS constrained large scale web service composition using abstraction refinement," *IEEE Trans. Serv. Comput.*, vol. 13, no. 3, pp. 529–544, May/Jun. 2020.
[44] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using ARIMA model and its impact on cloud applications' QoS," *IEEE Trans. Cloud Comput.*, vol. 3, no. 4, pp. 449–458, Dec. 2015.
[45] Z. Chen, J. Hu, G. Min, A. Y. Zomaya, and T. E. Ghazawi, "Towards accurate prediction for high-dimensional and highly-variable cloud workloads with deep learning," *IEEE Trans. Para. Dist. Syst.*, vol. 31, no. 4, pp. 923–934, Apr. 2020.
[46] Y. Lu, L. Liu, J. Panneerselvam, X. Zhai, X. Sun, and N. Antonopoulos, "Latency-based analytic approach to forecast cloud workload trend for sustainable datacenters," *IEEE Trans. Sust. Comput.*, vol. 5, no. 3, pp. 308–318, Jul.-Sep. 2020.
[47] J. Bi, H. Yuan, and M. Zhou, "Temporal prediction of multiapplication consolidated workloads in distributed clouds," *IEEE Trans. Auto. Sci. Eng.*, vol. 16, no. 4, pp. 1763–1773, Oct. 2019.

Satish Kumar received the PhD degree from the School of Computer Science, University of Birmingham, Birmingham, U.K. He is currently a research fellow with the School of Computing, University of Leeds, Leeds, U.K. His research interests include cloud computing, Edge computing, Internet-of-Things, distributed software systems and service computing. For further information, please visit https://kumar-satish.github.io

Tao Chen received the PhD degree from the School of Computer Science, University of Birmingham, U.K. He has broad research interests related to intelligent software engineering, including but not limited to search-based software engineering, performance engineering, self-adaptive software systems and data-driven software engineering. As the lead author, his work has been published in the top Software Engineering journals and conferences, such as *IEEE Transactions on Software Engineering*, *ACM Transactions on Software Engineering and Methodology*, *IEEE Transactions on Services Computing*, ICSE, FSE, and ASE.

Rami Bahsoon received the PhD degree in software engineering from the University College London, London, in 2006, investigating stability of software architectures. He is currently a reader with the School of Computer Science, University of Birmingham, Birmingham, U.K. His investigations have also looked at self-aware software architectures, economics-driven software architectures, and technical debt management in cloud software engineering.

Rajkumar Buyya (Fellow, IEEE) is a Redmond Barry distinguished professor and director with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne, Australia. He has authored more than 625 publications and seven text books. He is one of the highly cited authors in computer science and software engineering worldwide (h-index=118, g-index=245, 73,200+ citations). He served as the founding editor-in-chief of the *IEEE Transactions on Cloud Computing* and currently serving as editor-in-chief of Software: Practice and Experience.