DebtCom: Technical Debt-Aware Service Recomposition in SaaS Cloud

Satish Kumar, Student Member, IEEE, Tao Chen, Member, IEEE, Rami Bahsoon, Member, IEEE, 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 paper, 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—Service composition, technical debt, software adaptation, optimization.

1 INTRODUCTION

SERVICE composition has emerged as a standard way of building software application by composing multiple existing component 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 on SaaS is the realization of multi-tenancy, where multiple tenants are simultaneously served by the same composite service on based on shared resources [2]. However, the workload of composite service can be changed rapidly during execution, causing dynamic behaviours of the composite services. On one hand, increasing workload can rise over-utilization for the component services within a composite service, which in turns, would negatively affect the Quality of Service (QoS) and violate the Service Level Agreement (SLA) [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?

1. Tenants denote the end-users in SaaS Cloud

While service recomposition (or reconfiguration) in the SaaS 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 prepared for a largely increased workload for 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. 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]. The 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 tailor a time-series prediction method, namely ARFIMA model, into the debt model for predictably learning future debt.
- We propose a new concept and model, namely service debt, that explicitly map the metaphor of technical debt in the contexts of service composition. Such a model is capable of quantifying both good and bad debt.
- 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 to equip DebtCom with the ability to decide whether to trigger recomposition, 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 evaluate DebtCom on a service system in 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 paper is a significant extension of our previous work [7], in which we have documented a preliminary version of the technical debt model for service composition in SaaS. Specifically, our extensions include:

- We present an in-depth discussion and reasons that motivate our design of the service debt model, which significantly extends our prior work [7].
- We propose a debt-aware trigger for recomposing service such that the service debt model and time-series prediction serve as an integrated driver. The approach also supports controllable trade-off between short-term advantages and long-term benefits.
- We develop a holistic debt-aware framework for recomposing services in SaaS, namely DebtCom.
- We extend the experimental evaluation to larger scale service composition scenario of up to 10 abstract services, each of which has 100 component services and with over 10 different Docker containers.

The remaining of this paper are organized as follows. Section 2 presents the background of technical debt and a motivating scenario. Section 4 describes the time-series perdition method of service workload in DebtCom. Section 5 specifies the proposed notion of service debt, which is the core of DebtCom. Section 6 illustrates the debt-aware triggers of recomposition in DebtCom, grounded on the time-series prediction and service debt model. Section 7 explains the architecture and implementation details of DebtCom. Section 8 discusses the experimental results, particularly with respect to the state-of-the-art. Section 9, 10 and 11 specifies the threats to validity, related work and conclusion, respectively.

2 PRELIMINARIES

2.1 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-term benefits, aiming to make informed decisions based on when (or whether) the technical debt can be paid off [12].

3 TECHNICAL DEBT IN SERVICE COMPOSITION

In service composition, 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].
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 benefits in terms of improving instant utility but not geared for long-term benefits or future value creation. 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 values and avoiding unnecessary recomposition. 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 at time $t_1$ and take into account the possibility of generating future values in current service composition plan, as shown in Figure 1. Moreover, it may be possible to intentionally incurred the technical debt, which is accumulated over time, for a period (e.g., $t_1, ..., t_n$), because at each point in time (e.g., $t_1, t_2, t_3, ..., t_n$), the debt may be increased or decreased due to dynamic changes in requests workload generated by users in the SaaS cloud. We may accept such time-varying debt in a way to consider the future demand for scaling-up the service capacity that transforms the accumulated debt into future value. As a result, technical debt-aware decision 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 service recomposition decisions that produce weak composite service; which fails to process incoming requests workload in a dynamic 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 as shown in Figure 2. 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.

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.

4 TIME-SERIES PREDICTION OF 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 estimation of recomposition.

In DebtCom, we use Autoregressive Fractionally Integrated Moving Average model (ARFIMA) [26], a widely used time-series model that guarantee the prediction accuracy when a time-series contains long memory pattern, to predict the workload of each component service.

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:

\[
\Phi(B)(1 - B)^d X_t = \Theta(B)\varepsilon_t
\]

where \((1 - B)^d\) is the fractional differencing operator and the fractional number \(d\) is the memory parameter, such that \(d \in (-0.5, 0.5)\). \(\Phi(B) = 1 - \phi_1 B - \phi_2 B^2 - \ldots - \phi_p B^p\) is the autoregressive polynomial of order \(p\) and \(\Theta(B) = 1 + \theta_1 B + \theta_2 B^2 + \ldots + \theta_q B^q\) 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 \(\varepsilon_t\) represent the white noise process.

The prediction process in DebtCom is written s 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 Geweke 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.

5 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.

5.1 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.

5.2 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 equation 2.

\[
Principal = E \times C_{cpu}
\]

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.
5.3 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 \( y \)-th component service for the \( x \)-th 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 equation 1, as shown below:

\[
Int(CS_{xy})_{m,n} = \begin{cases} 
\sum_{t=m}^{n} ((T - W_t) \times C) & \text{if } W_t \leq T \\
\sum_{t=m}^{n} (W_t - R_{SLA}) \times P & \text{otherwise}
\end{cases}
\]

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], 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.

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 equation 5:

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

6 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 need.

6.1 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:

\[
R(CS_{xy}) = W_t \times C_{tenants}
\]

\[
C(CS_{xy}) = W_t \times C
\]

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 \( x \)th timesteps, denoted as \( U_n \), can be calculated as:

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

where \( x \) 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.

6.2 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 3.3, 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:

\[
D_{m,n} = \begin{cases} 
D_{bad} & \text{if } U_n > U_{n+k} \text{ and } D_{m,n} < D_{n,n+k} \\
D_{good} & \text{otherwise}
\end{cases}
\]

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
Algorithm 1: Debt-Aware Recomposition Trigger

1. **Input:** $W_c, W_j, P_n, S$
   */ $W_c$: 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. $D_{m,n} \leftarrow \text{calcuImateDebt}(P_n, W_c)$
   */ using equation (2), (3) and (4) */

5. $U_n \leftarrow \text{CalculateUtility}(P_n, W_n, D_{m,n})$
   */ using equation (5), (6) and (7) */

6. for $j \leftarrow n + k$ to $n + 1$
   7. $D_{n,j} \leftarrow \text{calcuImateDebt}(P_n, W_j)$
   8. $U_j \leftarrow \text{CalculateUtility}(P_n, W_j, D_{n,j})$
   9. if ($U_n > U_j$ and $D_{m,n} < D_{n,j}$) then
      10. /* using equation (8) */
      11. if $j == n + k$ then
      12. $D_{bad} \leftarrow 1$
      13. end
      14. if $j == n + 1$ then
      15. $S'_{demand} \leftarrow \frac{W_j}{j-n}$
      16. $P_j \leftarrow \text{optimize}(S, S'_{demand})$
      17. end
      18. else
      19. if $j == n + k$ then
      20. /* No recomposition needed */
      21. $D_{good} \leftarrow 1$
      22. else
      23. $S'_{demand} \leftarrow \frac{W_j}{j-n}$
      24. $P_j \leftarrow \text{optimize}(S, S'_{demand})$
      25. break;
      26. end
   27. return $P_j$

$n$ and time $n + k$. When $U_n < U_{n+k}$ and $D_{m,n} > D_{n,n+k}$, 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 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.

6.3 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 $j (n < j \leq k)$ and service debt up to that timestamp can be estimated, denoted as $U_j$ and $D_{n,j}$, respectively (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 7-8. 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 18-20) 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 13-16). 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 8.8, we experimentally examine the sensitivity of DebtCom to $k$ in terms of both the utility and running time.

7 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 Figure 4. Briefly, in the following, we discuss the general interaction between the components of these levels.
7.1 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 an 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, Service 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 Figure 5, 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 at Section 5. 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 determines whether to trigger the recomposition based on the the approach discussed in Section 6.

7.2 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., Apache ODE) 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 composition plan.

7.3 Back-end Process and Data Repository Level

Here, as discussed in Section 4, 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]).

8 Experimental Evaluation

In the experiments, 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?

8.1 Experimental Setup

For experimental purpose, we developed a real-world e-commerce system, 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 real-world WSDream dataset [23]. The relevant setups of the subject service systems have been shown in Table 1, 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.

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 8.8. Note that we use a sampling interval of 1 second in our experiments.

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

### 8.2 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 actual optimization mechanism is also the same.

### 8.3 Metrics

The evaluations of DebtCom have made use of the following metrics:

- **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 Equation (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). Similar 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.

### 8.4 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 2. Considering that the general workload varies between around 35 and 60 requests per second, the MAE and RMSE are in fact relatively low 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, Figure 5 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 pattern of trace of, e.g., the spike between 4000 and 5000 second 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.

### 8.5 RQ2: Results of DebtCom against Baseline

To investigate RQ2, we compare DebtCom with the Baseline approach as discussed in Section 8.2. In particular, we assess their utility, service debt and operation cost for recomposing the services under the testing period of the workload.

Figure 6 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:

<table>
<thead>
<tr>
<th>Debt Type</th>
<th>DebtCom</th>
<th>Baseline</th>
<th>Debt Impacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Good Debts</td>
<td>1417</td>
<td>1132</td>
<td>Improving overall service utility</td>
</tr>
<tr>
<td>Total Bad Debts</td>
<td>983</td>
<td>1268</td>
<td>Degrades overall service utility</td>
</tr>
</tbody>
</table>

Figure 7: Service debt, operation cost and utility achieved by Baseline over all timesteps

Figure 8: Service debt, operation cost and utility achieved by DebtCom over all timesteps
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 Figure 7 and 8, 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 second, because of the fact that the workload fluctuation till that point is relatively light. However, following the spike between 4000 and 5000 second 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 whether to recompose that takes the long term benefits into account.

In Table 3, 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.

8.6 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 Figure 10, 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 Figure 9, similar conclusion can be drawn. In particular, the ability of prediction in Proactive makes it robust to the spike after 4000 second 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.

8.7 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 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 Figure 11, 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 10ms 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.

8.8 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 Figure 12. 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.

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 correction2 and calculate the η2 as the effect size, which can be interpreted following the guidance by Tomczak and Tomczak [30]. The results reveal that the sensitivity of DebtCom to k value is statistically

2. We use .05 as the significance level, which becomes .005 after correction.
(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

Figure 9: Utility and service debt achieved by Passive, Proactive and DebtCom over all timesteps

Figure 10: Accumulated utility and accumulated debt achieved by Passive, Proactive and DebtCom over all timesteps

Figure 11: Running time

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.

9 Threats to Validity

**Threats to construct validity** is used to determine whether the adopted 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, service debt, good/bad debt count and running overhead.

**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 assessed 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 experimental evaluation, we developed a real-world ecommerce system as a testing environment, with up to 10 abstract services, each of which has possible 100 component services and 10 dockers. Further, DebtCom has been evaluated on the real-world WSDream dataset [23] and FIFA98 workload trace [25].

10 RELATED WORK

Over decades, different approaches have been presented for reconfiguring the composite service with QoS requirements [24] [5] [6] [31] [32] [36] [37]. Among others, most of the work is focused 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 an algorithm that continuously monitors the QoS of composite service and triggers the service recomposition once the SLA is violated. The optimization relies on Genetic Algorithm. Chen et al. [36] [37] 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 respond to changes. As we have shown, acting proactively to likely changes would create extra benefits and improve the overall utility. The DebtCom, on the other hand, proactively triggers recomposition based on time-series prediction of the workload.

Indeed, 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.

11 CONCLUSION AND FUTURE WORK

In this paper, we propose a technical debt aware framework, namely DebtCom, 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 DebtCom 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.

In future work, we will study the problem of optimizing service composition plan using technical debt analysis. In particular, we will examine how technical debt metaphors deals with service constraints (e.g., notably hard and soft service constraints) and strengthening the service composition to reach consistent optimality throughout execution.

REFERENCES