Client-Specific Homogeneous Service Composition at Runtime for QoS-Critical Tasks

Zhengquan Li[1], Long Cheng[2], Zheng Song[1]

[1] Dept. of Computer and Information Science, University of Michigan at Dearborn
{zqli, eejoylim, zhesong}@umich.edu
[2] School of Computing, Clemson University
lcheng2@clemson.edu

Abstract. Web services are widely used in modern software, providing diverse data and functionalities. Some data and functionalities are critical to an application's execution and user experience, posing strict requirements on the Quality of Service (QoS) of their delivery (e.g., latency and reliability), which services often fail to meet. Previous studies show that composing homogeneous services, i.e., simultaneously invoking multiple services providing the same functionalities and returning the first response, can improve latency and reliability. However, this approach increases the workloads on cloud servers and causes additional network traffic, limiting its deployment at scale. Noting that services provide varying QoS across different locations, we introduce an approach that dynamically composes homogeneous services for each client, achieving the desired QoS for critical services while minimizing invocation costs. Specifically, our method probes the QoS of all homogeneous services for a client, then calculates an optimal composition strategy that meets QoS requirements at the lowest cost. The evaluation results show that our approach significantly improves the QoS invoking a single service (enhancing reliability to 100%, reducing average latency by 7% and tail latency by 35%) while incurring 50% less cost than static homogeneous composition, making it a useful tool for service-oriented applications.

Keywords: Quality of Service · Service Composition · Latency Optimization

1 Introduction

Web services are widely used in modern software, providing diverse data and functionalities essential for various applications. Some of these data and functionalities are critical to an application's execution and user experience, such as real-time data processing in financial transactions or live updates in social media platforms. Given several functionally-equivalent services, developers face the problem of how to meet the stringent QoS requirements of their applications, in terms of latency, tail latency, and reliability.

The current state-of-the-practice approach is to select and invoke a skyline service (see Fig. 1 up left), i.e., a service that is not dominated by any other functionally-equivalent services. However, according to service statistics collected by service marketplaces [7] and an empirical study on developers' comments invoking web services [2], even skyline services sometimes fail to meet QoS requirements. To further improve QoS for critical tasks, previous works [3, 9] have explored the composition of homogeneous services, which involves simultaneously invoking multiple equivalent services and using the first response to continue the application's execution (Fig. 1 bottom left). While effective in enhancing QoS, this method significantly increases the number of service invocations, causing higher workloads on web servers and more network traffic. The additional invocation cost makes this approach impractical for large-scale deployment.

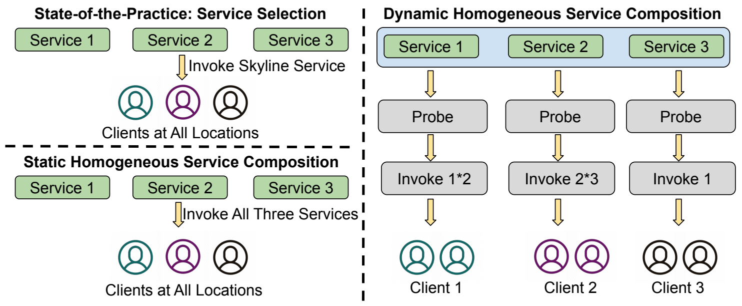


Fig. 1: Approaches for Satisfying the QoS Requirements of Critical Tasks

Motivated by that most services exhibit significantly different QoS at various locations [12], this paper introduces a more cost-efficient homogeneous service composition approach, which customizes service composition for end users at runtime. As demonstrated by the right sub-figure in Fig. 1, for clients at different locations, our approach first probes the QoS of all homogeneous services, and calculates a client-specific composition strategy that best satisfies the developers' QoS requirements within a predefined invocation cost budget.

The main contributions of our paper are as below:

- We introduced an approach that customizes service composition strategies for individual clients for both QoS improvement and cost efficiency. To support this idea, we developed a QoS estimation model for composition strategies, which ensures selecting a QoS-optimal strategy with less probing cost.
- The trace-based evaluation indicate that 1) our QoS estimation model is more accurate than all baseline approaches; 2) on average, our approach improves reliability from 99% to 100%, reduces mean latency and tail latency by 7% and 35% respectively, while incurring 50% less cost than static homogeneous service composition.

2 Related Works and Motivation

In this section, we summarize approaches for meeting the QoS requirements of critical tasks and the motivation of our approach.

2.1 Meeting the QoS Requirements of Critical Tasks

The problem of QoS-based web service selection and composition has consistently received significant attention over the past two decades [1]. As service-oriented architecture remains the primary method for accessing remote data and functionalities, critical tasks such as VR/AR, autonomous driving, and financial transactions demand stringent QoS requirements. Below, we introduce current solutions aimed at meeting these requirements.

Skyline Service Selection. Among a set of equivalent services, a service is considered a skyline service if no other service is better in all QoS attributes simultaneously. Developers can either hard code a pre-selected skyline service into their application [1] or rely on service gateways to select a service with optimal real-time QoS for composition [6]. This approach assumes that a selected skyline service can always satisfy the soft QoS requirements of an application. However, this is not true for emerging applications with hard QoS requirements.

Static Homogeneous Service Composition. Homogeneous service composition has been explored in various contexts to enhance QoS. For example, some studies [9] invoke multiple microservices in a way of speculative parallel fashion to improve the system reliability and execution time for applications in IoT environments. [3] parallely invoke multiple cognitive web services to improve accuracy (e.g., face recognition accuracy). Despite their benefits, these approaches simply invoking all services specified by static configurations, which lack flexibility and can lead to significant operational costs at runtime.

2.2 Motivation

A large-scale measurement of service QoS [12] revealed significant variations in service performance across different invocation contexts, such as locations and times. This motivated our approach of dynamically composing homogeneous services for individual users. To find the efficient service combination for each user, it requires accurately estimate the resulting QoS for each combination. Existing method [4,10] uses the minimum average latency among services as the composited latency, which is inaccurate: We observed that the average latency of a speculative parallel invocation can be even lower, as the fastest service may sometimes experience long tail latency, and the slower service might return the result sooner.

3 HomoService: Client-specific Service Composition

This section introduces our approach to composing homogeneous services for individual clients, termed as "HomoService". In particular, HomoServic first probes the QoS of all homogeneous services for a client, then calculates an optimal composition strategy that meets QoS requirements at the lowest cost.

3.1 Problem Formulation for Finding Optimal Composition

Given the service invocation cost and the limited budget of the application provider, it is essential to maximize QoS benefits in a cost-effective way when composing homogeneous services. We formulate this composition problem as follows: 1) Using a soft/hard QoS model, the developer's requirements for reliability and latency are treated as hard and soft constraints, respectively. Let $\hat{C}$, $\hat{R}$, $\hat{L}$, and $\hat{T}$ represent the developer's per-invocation budget, minimum reliability, desired average latency, and desired tail latency. 2) Let $\mathcal{I} = i = 1, 2, 3, \ldots, I$ denote the set of homogeneous services, and $\mathcal{S} = s = 1, 2, 3, \ldots, S$ represent all possible composition strategies. For a strategy s, L_s, R_s, T_s, and C_s denote its latency, reliability, tail latency, and cost. The optimal composition problem can be formulated as:

$$\begin{aligned} &s = \arg\max \frac{L_s + \hat{L}}{L_s} * \frac{T_s + \hat{T}}{T_s} \\ &\text{subject to: } \forall s \in \mathcal{S}; R_s \geq \hat{R}; C_s \leq \hat{C} \end{aligned} \tag{1}$$

, where $\frac{L_s+\hat{L}}{L_s} * \frac{T_s+\hat{T}}{T_s}$ represents the index of latency satisfaction, whose value range is $(1, +\infty)$. The latency satisfaction index considers both the average latency and tail latency. If both the desired values for tail latency and average latency are achieved (i.e., $L_s = \hat{L}$, $T_s = \hat{T}$), the latency satisfaction index reaches 4. A larger index means higher satisfaction of the overall latency, i.e., average and tail latency.

As we only consider the speculative parallel invocation, there are only $S = 2^I$ combinations of strategies, i.e., whether each service is in a strategy. We can treat Equation 1 as a 0-1 knapsack problem, i.e., given a set of homogeneous services, each with a certain latency, reliability, and cost, determine which services to include in the composition so that the reliability requirement is fully satisfied, the total cost is less than or equal to a given limit, and the latency satisfaction index is as small as possible. Various methods, such as dynamic programming, and exhaustive search, can be used to solve this 0-1 knapsack problem. These methods explore possible service combinations, evaluating them based on latency, cost, and reliability constraints to find the best option. Accurate estimation of QoS parameters (cost C_s, reliability R_s, latency L_s, and tail latency T_s) is crucial to determining the optimal composition.

3.2 Composition QoS Estimation Model

We introduce the limitation of current models for composition QoS estimation, and how we address it with a fine-grained way.

Existing Approaches for Estimating Composition QoS For a service $i \in \mathcal{I}$, we use c_i, r_i, and l_i to denote its cost, reliability, and latency. Given

a set of homogeneous services $\mathcal{H}_s$ of a composition strategy s, their speculative parallel invocation succeeds when any service succeeds, and fails when all constituent services fail. Existing approaches [4, 10] estimate the QoS for their speculative parallel invocation as: 1) $C_s = \sum_{i \in \mathcal{H}_s} c_i$; 2) $R_s = 1 - \prod_{i \in \mathcal{H}_s} (1 - r_i)$; 3) $L_s = \min(l_i), \forall i \in \mathcal{H}_s$. We found that although the cost and reliability estimations are accurate, the latency is not. The average latency of a speculative parallel invocation can be even lower, as the fastest service may sometimes experience long tail latency, and the slower service might return the result sooner. Furthermore, these methods do not consider tail latency estimation. As a result, these approaches are insufficient for accurately estimating composition latency.
Our Approach for Estimating Composition Latency To more accurately estimate the latency of composition, we treat service latency as a distribution rather than a single value. The distribution is calculated from the recorded QoS data of the service at runtime. In particular, this paper adopts the Shifted Exponential Distribution [8] to model individual service's latency. We employ a piece-wise function, $F_i(x)$, to represent the Cumulative Distribution Function (CDF) of service i's latency, where x denotes a latency value.

$$F_i(x) = \begin{cases} 0, & x < t \\ 1 - e^{-\frac{1}{m-t}(x-t)}, & x \geq t \end{cases} \tag{2}$$

The distribution parameters, t and m, correspond to the service's minimum latency and average latency, respectively. In contrast to other service latency distributions (e.g., Erlang and Pareto Distribution) that require estimating parameters from the entire set of invocation samples, the distribution we choose is simple and only requires to acquire one additional latency statistical parameter, the minimum latency, in addition to the average latency.

After modeling each homogeneous service latency, we calculate the resulting latency distribution for a composition. Recall that we use $l_i, \forall i \in \mathcal{H}_s$ to denote the latency of a service i. We use $\mathbf{P}\big(L_s \leq x\big)$ to denote the probability of the latency L_s of a composition strategy s is less than x. Assuming $l_i, \forall i \in \mathcal{H}_s$ are independent, we have:

$$\begin{aligned} \mathbf{P}\big(L_s \leq x\big) &= \mathbf{P}\big(\min(l_i) \leq x\big), \forall i \in \mathcal{H}_s \\ &= 1 - \mathbf{P}\big(\min(l_i) > x\big), \forall i \in \mathcal{H}_s \\ &= 1 - \prod_{i=1}^{\mathcal{H}_s} \mathbf{P}\big(l_i > x\big) \\ &= 1 - \prod_{i=1}^{\mathcal{H}_s} (1 - \mathbf{P}\big(l_i \leq x\big)) \end{aligned} \tag{3}$$

We can calculate the resulting latency CDF $\mathbf{P}\big(L_s \leq x\big)$ by using $P(l_i \leq x) = F_i(x)$, where $F_i(x)$ can be given by Eq. 2. After calculating the latency distribution, we can further calculate other statistics of interest, such as the average latency (i.e., L_s) or tail latency (i.e., T_s) in a closed-form expression, which other service distributions models cannot achieve.

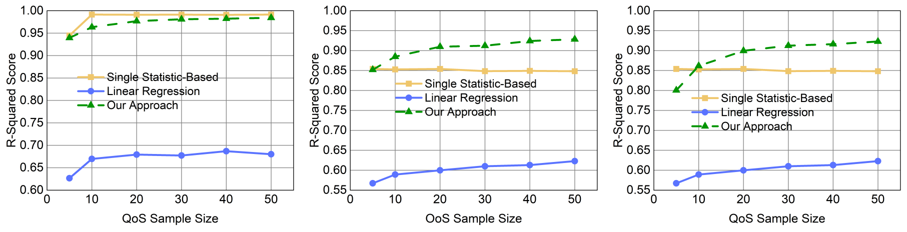
4 Evaluation

In this section, we evaluate HomoService by addressing two key questions: 1) How accurate is our QoS estimation model? 2) How much does HomoService improve QoS and reduce invocation costs?

To answer these questions, we collected service QoS of multiple tasks from different locations worldwide. Specifically, we selected six types of tasks: weather forecasting, IP-to-location, face detection, language translation, flight data retrieval, and hotel data retrieval, to cover typical service usages scenarios [11]. For each task, we selected three homogeneous services, giving priority to those with lower subscription costs and higher popularity. We developed a Python program to invoke these services every 40 seconds from five global locations—Frankfurt, Tokyo, Sydney, Mumbai, and Michigan—over continuous three days. This process collected approximately 4,000 samples per service set, totaling 90 service invocation trace sets. We used this dataset to evaluate both the accuracy of the QoS estimation model and the performance improvements achieved through HomoService.

4.1 Performance of our QoS Estimation Model

We measured the estimation accuracy of our model on average latency and tail latency (90^{th}, 95^{th} percentile latency). We compared our approach with the following three baseline approaches: 1) Average Latency Based [4, 10], which estimates the latency of service composition as the minimum of the average latencies of all invoked services; 2) Single Statistic-Based, which further estimates tail latency as the minimum tail latency of individual services. For example, for three services with tail latencies T_1, T_2, T_3, the composition's tail latency is $T = \min(T_1, T_2, T_3)$; and 3) Linear Regression [5], which is trained using 80% of 90 sets of invocation traces and predicts average and tail latencies. In the above example, the linear regression model takes T_1, T_2, T_3 as inputs and outputs the estimated resulting tail latency for a composition strategy that combines three services.



(a) Average Latency (b) 90% Tail Latency (c) 95% Tail Latency

Fig. 2: Sample Size's Impact on Modeling Fitness for Service Composition

We compared the performance of these methods with varying sample sizes, incrementing from 5 to 50 in steps of 10. We randomly selected the required number of samples from each trace and repeat the procedure 400 times. As shown in Fig. 2, when the sample size reaches 30, our approach showed much better accuracy than other approaches, especially for tail latency. Overall, our method more accurately estimates composition latency, particularly with limited samples.

4.2 QoS Benefits and Cost Reduction of Using HomoService

After estimating QoS for a single composition strategy, we show how to optimally compose homogeneous services to maximize QoS improvement considering the developer-specified QoS parameters.
QoS Parameters Configuration For using HomoService, parameters like service invocation cost, required reliability, latency, and sample size are crucial. We developed a random budget generator to evaluate our solution under various cost constraints. The budget range is defined from the cost of the QoS optimal service (minimum) to a maximum of K times the highest service cost, with K initially set at 2. Costs were sourced from the service corresponding Rapid page, normalized to integers, and used to generate budgets. We targeted a reliability of 99.99% and aimed for latencies 15% better than the QoS optimal service, with a set sample size of 50 for each task using the composition.

	Reliability	**Cost**	**Latency ms (Avg. | Tail)**
Paradigm	Skyline/Static/Dynamic	Skyline/Static/Dynamic	Skyline/Static/Dynamic
India	0.99/1.00/1.00	356/645/467	841/711/744 | 3094/1392/1773
Japan	0.99/1.00/1.00	356/645/467	500/432/448 | 2882/1111/1500
Australia	0.99/1.00/1.00	356/645/469	783/730/743 | 3379/1683/2059
Germany	0.99/1.00/1.00	356/645/468	671/680/701 | 1389/1119/1250
US	0.99/1.00/1.00	356/645/468	661/550/591 | 2100/1621/1804
Avg.	**0.99/1.00/1.00**	**356/645/468**	**691/620/645 | 2568/1385/1677**

Table 1: Service QoS Performance at Five Locations Worldwide

QoS Improvement and Costs Comparison We compared our dynamic approach with other two baselines: 1) Skyline, which invokes the service with the best average latency across five locations, and 2) Static Homogeneous Service Composition, which hardcodes and simultaneously invokes three homogeneous services. We ran 400 Python simulations and summarized the average results in Table 1, showing the QoS performance of three different invocation paradigms at five global locations. Findings include: 1) dynamic composition boosts QoS at each location, enhancing reliability to 100% and reducing average latency by 7% and tail latency by 35% compared to invoking skyline services; 2) Overall, dynamic composition costs 31% more than Skyline, whereas static composition costs 80% more but only slightly improves latency (3% average, 11% tail). In summary, HomoService offers the most cost-effective improvement in service reliability, latency, and budget efficiency.

5 Conclusion

In this paper, we introduced an approach that dynamically composes homogeneous services for each client, achieving the desired QoS for critical services while minimizing invocation costs. Unlike static homogeneous service composition, which invokes all services to improve QoS at high costs, our approach customizes the optimal composition strategy to balance QoS benefits and invocation costs for each end user.We collected QoS data across various tasks and locations and conducted extensive evaluations based on this dataset. The results demonstrate the effectiveness and practicality of our approach, offering a valuable solution for service-oriented application developers.

Acknowledgement

This research is supported by NSF through grant 2104337. The authors also want to thank Ms. Eejoy Lim for her contribution to this work.

References

1. Alrifai, M., Skoutas, D., Risse, T.: Selecting skyline services for QoS-based web service composition. In: WWW'10. pp. 11–20 (2010)
2. Baravkar, S., Zhang, C., Hassan, F., Cheng, L., Song, Z.: Decoding and answering developers' questions about web services managed by marketplaces. In: 2024 IEEE International Conference on Software Services Engineering. IEEE (2024)
3. Bhatia, A., Li, S., Song, Z., Tilevich, E.: Exploiting equivalence to efficiently enhance the accuracy of cognitive services. In: IEEE CLOUDCOM'19. pp. 143–150
4. Hiratsuka, N., Ishikawa, F., Honiden, S.: Service selection with combinational use of functionally-equivalent services. In: 2011 IEEE International Conference on Web Services. pp. 97–104. IEEE, Washington DC, USA (2011)
5. Mann, G., Sandler, M., Krushevskaja, D., Guha, S., Even-Dar, E.: Modeling the parallel execution of black-box services. In: HotCloud (2011)
6. Moussa, H., Gao, T., Yen, I.L., Bastani, F., Jeng, J.J.: Toward effective service composition for real-time soa-based systems. SOCA **4**, 17–31 (2010)
7. rapidAPI: RapidAPI - the next-generation API platform (2015), `https://rapidapi.com/hub`
8. ReliaWiki: The exponential distribution (2017), `https://reliawiki.org/index.php/The\_Exponential\_Distribution`
9. Song, Z., Tilevich, E.: Equivalence-enhanced microservice workflow orchestration to efficiently increase reliability. In: 2019 IEEE International Conference on Web Services (ICWS). pp. 426–433. IEEE (2019)
10. Song, Z., Tilevich, E.: Win with what you have: QoS-consistent edge services with unreliable and dynamic resources. In: 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS). pp. 530–540. IEEE (2020)
11. Wu, Z., Madhyastha, H.: Understanding the latency benefits of multi-cloud webservice deployments. ACM SIGCOMM Computer Communication Review **43**, 13–20 (04 2013). https://doi.org/10.1145/2479957.2479960
12. Zheng, Z., Zhang, Y., Lyu, M.R.: Investigating QoS of real-world web services. IEEE transactions on services computing **7**(1), 32–39 (2012)