Poster: Service Polymorphism: Enhancing Web Service Performance by Serving Clients Dissimilarly

Zhengquan Li, Zheng Song
Department of Computer and Information Science, University of Michigan – Dearborn, MI, USA
{zqli, zhesong}@umich.edu

2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS) | 979-8-3503-8605-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICDCS60910.2024.00143

***Abstract*—Modern applications often invoke web services to access remote data and functionalities. The current service-oriented paradigm is "one-size-fits-all," where App developers expect a single service to deliver satisfying Quality of Service (QoS) to all geographically and temporally dispersed clients. However, our empirical study reveals that despite the pervasive use of CDN and edge computing, many web services deliver significantly varied QoS to different users, resulting in some clients suffering from poor user experience. This paper introduces service polymorphism, a novel software paradigm that serves dispersed clients dissimilarly to improve their perceived QoS. Service polymorphism allows a client to maintain a list of equivalent services and invokes the one that offers the optimal QoS in the invocation context. The main challenge in supporting service polymorphism lies in minimizing the overhead for fine-grained QoS sensing. To address this challenge, we propose an edge-based QoS sharing mechanism that aggregates the context-specific QoS in edge servers, and allows clients to retrieve the QoS from local WiFi Access Points with minimized latency to decide the optimal service. Our evaluation shows that service polymorphism improves QoS significantly for 8 services out of 20, reducing their average latency by 231 ms (45%), tail latency by 80 ms (12%), and error ratio from 0.2% to 0%.**

***Index Terms*—Web Service, Latency, WiFi Access Points**

I. SOA: Current Practice and Problems

Service polymorphism is motivated by our in-depth study of Service-Oriented Architecture (SOA) applications on GitHub, analysis of service information provided by RapidAPI [1] (the world's largest web service repository), and empirical measurement of service QoS in different locations and times.

SOA Practice: One-size-fits-all: We searched GitHub for repositories that invoke RapidAPI services. Our search yielded 873 unique repositories, constrained by GitHub's limitations. Upon manual inspection of these repositories, we noticed a common trend: *all these applications hardcoded a pre-selected service for each required functionality*.

Wide Existence of Equivalent Services: RapidAPI offers collections of web services, such as Weather APIs, Flight data APIs, and Translation APIs. These collections consist of services with similar functionalities provided by different vendors that can be used interchangeably in Apps with minor adaptions. For instance, a language translation functionality can be fulfilled by either *Text Translation*, *NLP Translation*, or *Lecto Translation*. Overall, we found that 461 collections feature 3 or more services, indicating a significant presence of equivalent services.

No Service being Always QoS-optimal: We selected 20 sets of equivalent web services from RapidAPI and invoked these services for 3 days from 8 gloabl locations. We observed that the QoS of services, particularly latency, varies significantly in different locations and times. Using the measured QoS of translation services as an example, Fig. 1 and Fig. 2 shows the latency of three equivalent services at each location and time slot, respectively. These figures clearly show that the pre-selected service, *Lecto Trans*, performs well in most locations and times but also underperforms compared to its peers in some locations (e.g., Ohio and Cape Town) and during certain time slots (e.g., slots 10 and 35).

Key Summary of Our Study

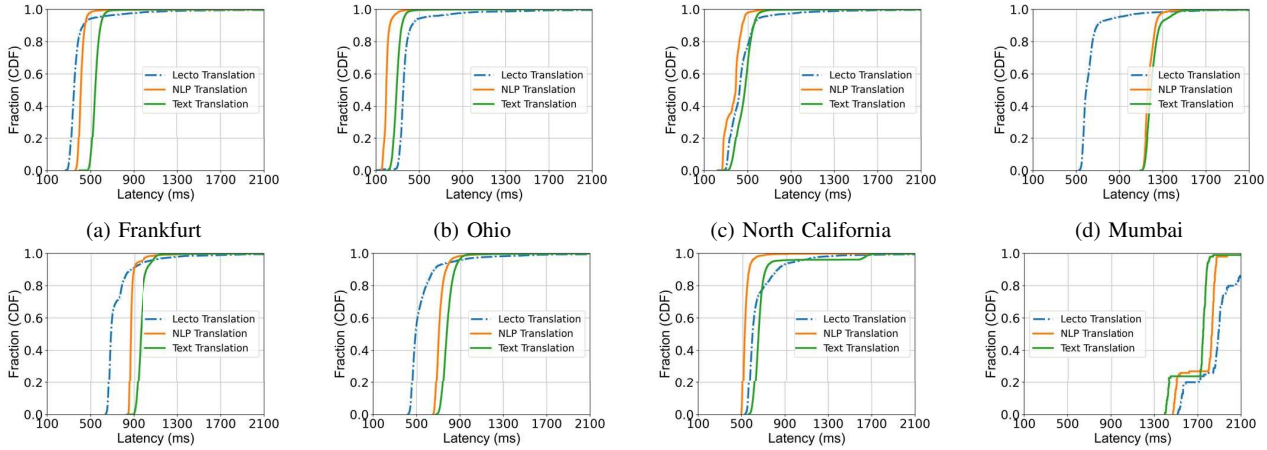
Many services suffer from fluctuating QoS over spatial/temporal contexts, rendering it *impossible to pre-select one service that consistently outperforms its equivalent peers in various contexts*. These observations motivate us to introduce service polymorphism as a replacement for the current "one-size-fits-all" programming paradigm of service-oriented applications.

II. Service Polymorphism

We introduce *service polymorphism*: a client automatically invokes web services in different contexts to guarantee optimal QoS. Instead of requiring each service to provide satisfactory QoS for all clients, a client maintains a list of equivalent services and dynamically decides which one to invoke based on their real-time QoS to provide the optimal QoS in current invocation context. To do so, the client needs to access the context-specific QoS statistics of all equivalent services.

Therefore, the main challenge for supporting polymorphism lies in *how to accurately assessing services' QoS with minimum overhead*. Previous studies purposed that the client individually send probe requests to each service either periodically or right before each service invocation, which incurs heavy overhead to both the clients and web servers. A promising solution to reduce QoS sensing costs is data crowd-sourcing. This method involves clients sharing their QoS sensing results with an edge server, which other clients can access the QoS. Typically, edge servers (e.g., Akamai) are deployed multi-hop away from end-users, with an average E2E communication

2575-8411/24/$31.00 ©2024 IEEE
DOI 10.1109/ICDCS60910.2024.00143



(a) Frankfurt (b) Ohio (c) North California (d) Mumbai

(e) Sydney (f) Tokyo (g) Sao Paulo (h) Cape Town

Fig. 1: Latency Distributions of Translation Services at Different Locations (dash lines representing the pre-selected services)

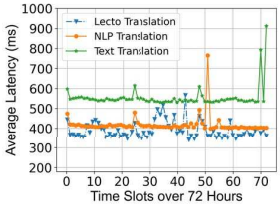


Fig. 2: Translation Services' Latency over Time

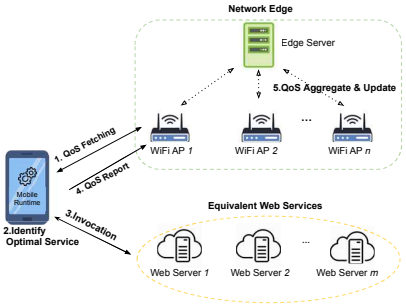


Fig. 3: Workflow of Service Polymorphism

latency of 20 to 30 ms [2]. To obtain timely service QoS, a client needs to communicate with the edge server before each service invocation. This extra latency overhead of QoS retrieval offsets the latency benefits of invoking the optimal service, potentially resulting in even higher latencies than these clients invoking a preselected service in some contexts.

To solve this challenge, we introduce a novel edge-based crowd-sourcing mechanism for fine-grained QoS sensing with minimized overhead. In particular, we integrate WiFi Access Points (APs), such as home routers, into the architecture of edge server QoS sharing. The clients share their service QoS sensing results to the edge server and retrieve the QoS data from the WiFi APs. These WiFi APs are ubiquitously positioned at the extreme edge of the network and only one hop away from the end users. Such proximity enables clients to access the shared QoS information with minimal latency prior to future service invocations.

Workflow: Before initiating a service request, the mobile runtime first fetches the QoS for all equivalent services from its associated WiFi AP. Based on the received QoS, the mobile runtime selects which service to invoke to ensure optimal QoS in the current context. After the invocation, the runtime reports its performance metrics (i.e., latency, successful or not) to the WiFi AP. The WiFi AP then forwards these performance results, along with a timestamp and location tag, to the edge server after removing any user identifiers. The edge server aggregates these performance results from the WiFi APs, calculates the QoS for each service, and disseminates this updated QoS back to WiFi APs in its vicinity.

III. Preliminary Experimental Results

We implemented service polymorphism as an Android library called Polymorphic Service Framework (PSF). We built a real testbed and deployed 20 pairs of Android Apps invoking the 20 sets of equivalent services used in our empirical study, one using pre-selected services which have the global optimal QoS and one using PSF. We compare the QoS performance of PSF and *Pre-selection* with 10 nearby clients present. We define *significant improvement* as PSF improves the performance by at least 20% over *Pre-selection*, *mild improvement* as 5% to 20%, and *insignificant improvement* as less than 5%.

Category	Average Latency Improv.		Tail Latency Improv.		Error Ratio Improv.
Significant (8)	44.89%	231.30ms	12.35%	80.31ms	0.2%
All (20)	17.59%	76.87ms	6.67%	38.72ms	0%

TABLE I: QoS Improvement of PSF over *Pre-selection*

According to our criteria, 8 sets (40%) are classified under the "significant improvement" category, one set falls into "mild", and 11 sets (55%) into "insignificant". Table I displays the average performance enhancements for both "significant" and all 20 polymorphic services, where PSF reduces average latency by 45% (231 ms) and 18% (77 ms), respectively.

References

[1] "Rapidapi - the next-generation api platform," https://rapidapi.com/, accessed: 2024-04-07.

[2] B. Charyyev, E. Arslan, and M. H. Gunes, "Latency comparison of cloud datacenters and edge servers," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 2020, pp. 1–6.