Edge Proxy Orchestration for HTTP-Based Service Continuity

Lorenzo Giorgi*, Carlo Puliafito†, Antonio Virdis† Enzo Mingozzi†
*T.A.I. Software Solution Srl, Pisa, Italy
†Department of Information Engineering, University of Pisa, Pisa, Italy

I. Introduction

Cloud computing provides computation via remote data centers. Edge computing extends it by deploying micro data centers (MDCs) across the cloud-edge continuum, closer to clients. This enables low-latency applications aimed at powering smart cities, such as augmented/virtual reality or autonomous driving. Edge computing is being standardized by ETSI as Multi-access Edge Computing (MEC).

In cloud-edge scenarios, clients often need to access services at different locations, for example due to mobility or for load redistribution. Ensuring seamless service continuity — transparent and robust access to services despite such changes — is essential. However, existing solutions often rely on ad hoc protocols or dedicated infrastructure, leading to complexity, limited transparency, or degraded performance.

While many service-continuity approaches assume direct client-service interactions, commercial cloud platforms (e.g., AWS, Azure) typically rely on a single proxy acting as the entry point to the cloud data center. This proxy acts as a transparent intermediary between the client and the pool of cloud services, managing load balancing, routing, access control, and telemetry collection. This trend is extending to cloud-edge systems. A single proxy entry point, however, is impractical due to the geographic distribution of edge servers. Recent proposals instead advocate distributed edge proxies. Each edge proxy may be decoupled from a specific MDC and rather execute as a forward proxy in close proximity to clients. In such a way, each proxy may potentially dispatch requests to any server in the cloud-edge continuum rather than to a subset of servers represented by a single MDC. This allows the system to be more flexible and muster the resources distributed in the continuum, to accommodate application requirements.

Following this trend, this work — which is the short version of [1] — leverages edge proxies as the core element of a service-continuity platform for the cloud-edge continuum. Fig. 1 depicts a mobility use case describing a real application benefiting from our solution. A drone traverses a city to deliver a parcel. Along its path, it may access the cloud-edge system to benefit from different types of service. Let us focus on a specific service, namely image filtering. The drone captures video frames of the surrounding environment and needs an image-filtering service, which we call λ, to analyze them. Access to the cloud-edge system is through distributed edge proxies, π_i, so that at each time the drone is connected only to one proxy in its proximity. At the beginning (see Fig. 1(**a**)), the drone connects to proxy π_1, which forwards service requests to λ hosted on *edge server 2*. Later, the drone moves far from its current entry point. As shown in Fig. 1(**b**), requests need to traverse a longer network path to reach proxy π_1. This may impair application performance. As a result, platform reconfiguration is needed to let the drone connect to π_2, which is closer to its new location (see Fig. 1(**c**)). Our example shows that requests eventually reach the same λ instance on *edge server 2*; however, the system orchestrator may decide to reallocate cloud-edge resources to meet system or application requirements.

Our solution is based on dynamic coordination of edge proxies. When a client needs to change from a source to a target proxy, an orchestrator updates proxy configurations. Specifically, it copies a per-client state on the target proxy, which lets the proxy recognize the client and know how to forward it requests. Besides, the orchestrator instructs the source proxy so that it can advertise the target proxy as the new one for the client. This advertisement is performed by leveraging the *Alternative Services* (AltSvc) extension of HTTP, which makes a client aware of service availability on a different network endpoint. The proposed approach does not require any ad-hoc network infrastructure nor protocol. Furthermore, it is transparent to applications and relies solely on standard HTTP extensions and orchestrated proxy behavior.

II. Overview of the Service-Continuity Solution

Our platform is designed around a distributed architecture that enables seamless service continuity for clients accessing cloud-edge services. Core to this solution is the use of edge proxies, which serve as entry points to the continuum for the clients. These proxies are managed by a central orchestrator that monitors client location and system load to dynamically assign each client to the most appropriate proxy.

When a client first logs into the system, the orchestrator authenticates it and assigns it to a nearby edge proxy. Two HTTP headers are included in the login response: an `Alt-Svc` header, pointing to the assigned proxy, and a `Set-Cookie` header, which contains an authentication token. This token allows to associate the client with a proxy-hosted, per-client state, which includes request-forwarding rules based on application requirements such as latency or throughput. As long as these requirements do not change, the client continues interacting with the system using the same proxy and cookie.

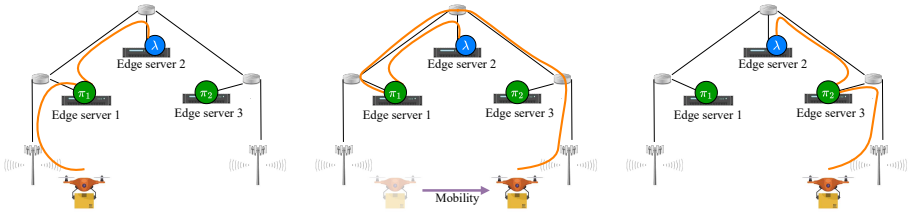

Fig. 1: The service-continuity use case of a drone for last-mile delivery.

Over time, the orchestrator may determine that a client should switch from a source to a target proxy, for example due to client mobility or for load redistribution. To manage this transition, the orchestrator updates the target proxy with the per-client state and configures the source proxy to include a new `Alt-Svc` header in its next response. This informs the client of the target proxy. As a result, the next client's requests reach the target proxy, though maintaining the same `Host` header field and cookie as before. As a result, this solution is totally transparent with respect to the application logic. As per the AltSvc mechanism, the client needs to open a new TCP connection and TLS session when it first contacts the target proxy. Once the transition is complete and the target proxy receives the first client request, the orchestrator finalizes the handover by cleaning up the per-client state on the source proxy and optionally deallocating service instances.

The implementation of our proposed platform, which leverages Envoy as proxying technology, is publicly available on GitHub at https://github.com/Unipisa/CEPHAS-platform.

III. Evaluation of Transaction Time

We show a simple experiment to validate our solution. We ran the test over the OpenStack cloud infrastructure of the University of Pisa. The testbed comprised five virtual machines (VMs), each having 2 virtual cores, 2 GB of RAM memory, and running Ubuntu 20.04 with Linux kernel 5.4: (i) two VMs run platform components; (ii) one VM runs the client; and (iii) two VMs are edge servers, each leveraging Docker to run Envoy proxy and service instance.

The goal of the test is assessing the impact of our solution on an application. We therefore performed long-run tests (80 minutes, 9600 transactions each) and measured the *transaction time* experienced by the application, which is the interval from the issuing of an HTTP request by the client to the reception of the related response. The considered application consists of a client performing a `GET` request every 0.5 s. The request firstly reaches the edge proxy of the client and is then forwarded to the backend service co-located with the proxy. The service container responds with "`hello from <host name>`".

In the experiment, the client moved back and forth between the two edge servers, getting closer to one and farther from the other, and changed edge proxy accordingly. We used *Linux Traffic Control* to emulate network delays and thus client mobility. Specifically, we set the round-trip delays to the edge servers to be 7 ± 1 ms and 26 ± 1 ms, for the closer and farther edge servers, respectively. The client moves every 20 minutes, thus causing three proxy changes per repetition.

We compared our solution against two alternatives: (i) *No Proxy Change* (NPC), where the client does not change proxy; (ii) *DNS*, the most used by ETSI MEC, where the client needs to resolve the DNS. We considered two variants — *DNS1* and *DNS60* — depending on the number of seconds, Time To Live (TTL), that the client maintains the DNS record in cache before asking again. For the AltSvc and DNS solutions, we considered two further variants, *Container* and *NoContainer*, whether or not the platform spawns a new service container besides changing proxy.

Fig. 2 presents the results of mean transaction time. In NPC, the client reaches the farther proxy for half of the experiment duration. DNS solutions pay the cost of establishing a new connection/session for each transaction. DNS60 has slightly higher mean transaction time since it takes longer to change proxy due to higher TTL. Our solution performs the best as it opens a new connection/session only when it changes proxy and allows the client to rapidly switch proxy. Figure 2 shows no significant advantage of the NoContainer variant. This is because only few transactions may benefit from the NoContainer variant, hence having no impact on the mean transaction time, which is calculated over thousands of transactions.

Acknowledgment

This work was supported by the European Union — NextGenerationEU — under the National Sustainable Mobility Center (Italian Ministry of University and Research Decree n. 1033—17/06/2022, Spoke 10); Partnership on "Telecommunications of the Future" through the Program "RESTART"; PRIN 2022 Project TWINKLE.

References

[1] L. Giorgi, C. Puliafito, A. Virdis, and E. Mingozzi, "Service continuity in edge computing through edge proxies and http alternative services," *IEEE OJCOMS*, vol. 5, pp. 7057–7074, 2024.

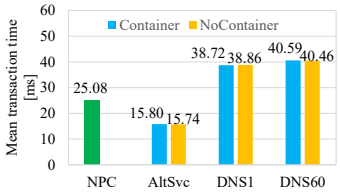

Fig. 2: Mean transaction time for the different solutions.