Data-anchored multi-cloud microservices placement: a greedy approach

Carmine Colarusso
Dept. of Engineering
University of Sannio
Benevento, Italy
ccolarusso@unisannio.it

Ida Falco
Dept. of Engineering
University of Sannio
Benevento, Italy
i.falco@studenti.unisannio.it

Eugenio Zimeo
Dept. of Engineering
University of Sannio and CINI
Benevento, Italy
zimeo@unisannio.it

***Abstract*—In a multi-cloud environment, the placement of microservices is crucial and may be subject to data constraints. Data could be anchored to some environments due to regulatory compliance, data sovereignty issues, or performance optimization. This enforces specific placement strategies to obtain good performances and scalability. In this extended abstract, we propose the adoption of the WL-A model expanded with the inclusion of anchors for implementing a data-centric placement strategy. The results show how limited data movements improve performance in terms of response times.**

***Index Terms*—service architecture, microservices, multi-cloud, placement, performance.**

I. Introduction

In distributed systems, especially on large-scale ones such as smart cities, the amount of data generated and managed grows exponentially. This phenomenon is due to the growing number of sensors, actuators, and services offered by cities. This is leading to significant improvements in the computing and storage facilities, with large investments on in-memory computing. This shift is introducing a new problem in distributed applications: when a large amount of data is collected by some nodes in the system, performing computation on these data by moving them to the execution environments of the interested services is infeasible. However, even in the case of a smaller amount of data, moving them could be difficult due to possible policy constraints, such as data privacy.

Smart waste management bins that gather data to optimize collection routes, smart meters for energy management that monitor consumption, and hospitals that need to store and keep private patients data are some examples of applications that may need to prevent transferring personal data among different authorities.

In this work, we propose a technique that exploits the concept of communication intensity to maximize the locality between services and data stores that compose a distributed application, with the constraint of data stationarity. The idea is to anchor microservices to data by assuming that the latter can not be moved in some cases. To get information about communication intensity, we exploit the Workload Application (WL-A) Model [1] (a runtime graph model of a distributed system) that captures system microservices' interactions with their intensity, extended to consider also the interactions between services and DBMSes.

The proposed approach includes and extends the Data Gravity [2] concept, i.e., the phenomenon where large datasets attract applications, services, and other data, making it more efficient to process data close to where it is stored. As data accumulates, it creates a "gravitational" pull that encourages the colocation of applications and services to reduce latency, minimize data transfer costs, and improve performance. Different from Data Gravity, which assumes a static-defined attraction force, adopting the WL-A model captures the dynamic nature of modern applications by changing the link intensity according to a specific workload.

In an ideal context with virtually unlimited hardware resources, the optimal solution for reducing communication costs would involve "annihilating" all services and data into a single environment. However, this is unfeasible in real-world settings due to hardware limitations, requiring a careful approach in microservice placement taking into account data anchoring, i.e., we aim at reducing communication costs towards anchored data and related microservices considering the resource constraints imposed by the environment hosting the data. This extended abstract presents a preliminary solution based on greedy algorithms and the related results.

II. Data-anchored placement

Starting from the WL-A graph model (considering microservices and databases as nodes and interaction between them as weighted directed edges, the weights are related to a reference workload), we introduce a new graph component called "anchor".

Definition 1. Given a directed weighted graph $G = (V, E, w)$, where V is the set of vertices, E is the set of edges, and $w : E \to \mathbb{R}$ is a weight function that assigns a weight to each edge, an *anchor* a is defined to incorporate a subset of leaf nodes and edges linked to a certain execution environment. The anchor a can be formally defined as a tuple $(L, E_L, W_{incident}(L))$. $L \subseteq V$ is the set of leaf nodes incorporated into the anchor. $E_L \subseteq E$ is the set of edges incident to any node in L: $E_L =$

This work is framed within the AVANT (dAta and infrastructural serVices for the digitAl coNTinuum) project by Engineering Ingegneria Informatica s.p.a.

$\{(p,l) \in E | l \in L\}$. $W_{incident}(L)$ is a function that maps each parent node p to a sum of weights of edges starting from p and incident to nodes in L, calculated as:

$$W_{incident}(L) = \left\{ p \leftarrow \sum_{(p,l) \in E_L} w(p,l) \mid p \in P(L) \right\} \quad (1)$$

where $P(L) = \{p \in V \mid (p,l) \in E \text{ for some } l \in L\}$ is the set of parent nodes connected to the leaf nodes in L. In this way, the edges directed to anchors are merged by the parent.

Leveraging this definition, we created a graph that extends the WL-A model to include *anchor* elements, i.e., nodes that work as attractors and can not be moved from their hosting environments.

A. Greedy Algorithm for Community Detection

Using the intensity between nodes, we apply a greedy algorithm to insert the microservices with the strongest attractive force into the *anchor* community. Starting from this newly constrained WL-A graph, the algorithm assigns nodes to form communities, beginning with the *anchor* nodes.

The algorithm's initial step assigns an *anchor* to each community:

$$C_a = \{a\} \; \forall a \in A, \quad (2)$$

where A is the set of anchors. Until all nodes in V are assigned, it searches, for an edge $e = (v,c)$, directed from v to c, where c is a node that belongs to a community while v does not belong to a community. The greedy algorithm searches for the e with the highest value of w and assigns v to the community of c:

$$C_a \leftarrow C_a \cup \{v\}. \quad (3)$$

At the end of this first step, if there are any leaf microservices, they are assigned to the community they most heavily invoke (indicated by directed edges from the community to the leaf node). The result is a set of communities $\{C_a \mid a \in A\}$, where each community C_a includes an anchor node and other nodes attracted by it.

III. Experimentation

To test our approach, we used the TrainTicket[1] benchmark application, consisting of 36 services and 20 databases. We computed the WL-A model by submitting a reference workload (W_r) with 20 concurrent users to the application deployed in a single execution environment (a VM with 48 vCPUs, 120GB of RAM, and 400GB of disk storage). Then, we collected and processed the execution traces with the pipeline of the previous work [1] to obtain the graph model. This graph contains the weighted interaction among services and also considers DBMSes. Hence, we divided the databases into 4 groups; each group is an *anchor* a to deploy into a dedicated virtual machine (VM); this way, each machine will host a single C_a. VMs are configured with 12 vCPUs, 32GB of RAM and 100GB of disk storage.

Fig. 1 shows the extended WL-A model with the *anchors* and their relationships with services using Eq.(1). The color of the nodes refers to the community C_a linked to a given *anchor*, obtained by applying our greedy algorithm.

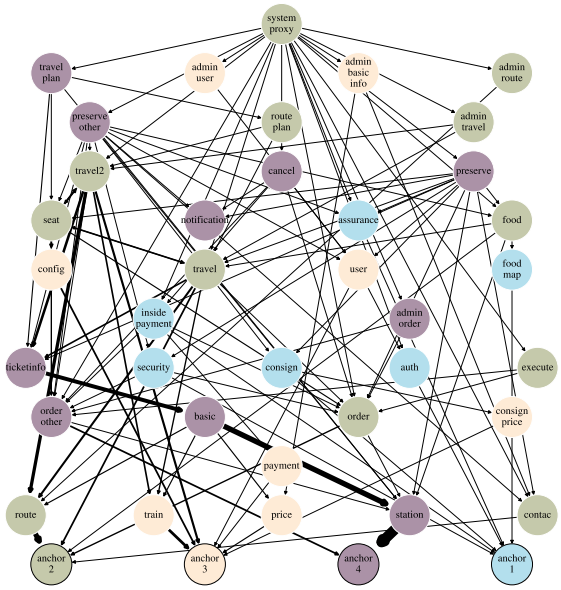

Fig. 1. Extended WL-A Model and its clustering based on the greedy community detection algorithm.

To evaluate our greedy-based placement proposal (P_G), a comparison was made with a placement proposal that, using the same anchors, randomly distributed the related services (P_R). We injected (W_r) into the application deployed according to both placement proposals, and then we collected the response times of endpoints.

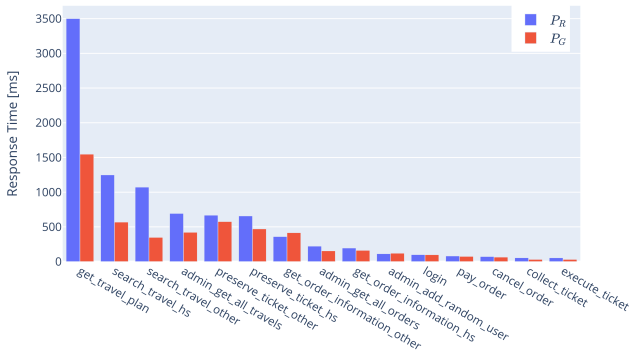

Fig. 2. Median response times of the slowest 15 endpoints.

Fig. 2 shows the median response times of some of the slowest operations for P_R and P_G. In almost every case, there is an improvement in response times in the case of intensity-driven anchor-based placement. The overall improvement is $\sim 25\%$ considering all the operations. About the slowest three endpoints, the improvement is higher than $\sim 54\%$ endpoint.

[1]https://github.com/FudanSELab/train-ticket

REFERENCES

[1] C. Colarusso, A. De Caro, I. Falco, L. Goglia, and E. Zimeo, "A distributed tracing pipeline for improving locality awareness of microservices applications," *Software: Practice and Experience*, vol. 54, no. 6, pp. 1118–1140, 2024.

[2] D. McCrory, "Data gravity – in the clouds." https://datagravitas.com/2010/12/07/data-gravity-in-the-clouds/. Accessed: 2024-05-24.