JURNAL EMACS
e-ISSN: 2686-2573
[Engineering, MAthematics and Computer Science] Vol.8 No.1 January 2026: 95-104 DOI: 10.21512/emacsjournal.v8i1.15594

The Paradox of Web Service Composition and Load Balancing: Theoretical Compatibility vs. Business Reality

Maksymilian Iwanow

Faculty of Management, Computer Science and Finance,
Wrocław University of Economics and Business,
Wrocław, Poland 53-345
maksymilian.iwanow@ue.wroc.pl

Correspondence: maksymilian.iwanow@ue.wroc.pl

Abstract – *Web services have become a fundamental building block of modern IT infrastructures. They underpin both: internal system integration and the delivery of complex business functionalities – and as organizations continue to shift toward service-oriented or microservice-based architectures, managing service interactions well is no longer an option. Composition and load balancing have grown into critical concerns for any company serious about keeping production systems running reliably at scale. This paper explores where service composition and load balancing conceptually overlap – and, perhaps more interestingly, where they contradict each other. The tension becomes visible when these mechanisms are treated as multi-criteria optimization problems rather than as pragmatic, cost-driven business decisions. To examine this, the study combines a structured literature review with informal participant observation carried out in a real enterprise environment. What emerges from the analysis is that service composition – broadly, the integration of multiple sub-services to fulfill a specific functional goal – frequently runs into load balancing, which distributes incoming requests across service instances to make efficient use of available resources. In practice, though, this intersection tends to be poorly understood. Academic literature and industry practice alike treat both mechanisms inconsistently, often without acknowledging the tensions between them. This paper tries to cut through that ambiguity by combining a close reading of the literature with hands-on professional perspective. The result is a synthesis that places itself somewhere between theory and practice. In spite of definitive answers, the paper is also intended as a starting point for a broader conversation – one that feels overdue, given how many relevant questions remain underexplored.*

Keywords: *Web Service; Service Composition; Load Balancing; SOA*

I. INTRODUCTION

At its most basic, a web service works like a black box – it receives an input, does whatever it needs to do internally, returns a result to the caller (Felício et al., 2023). In many cases, a web service takes the form of a monolithic application, where a large codebase handles a wide range of business & technical processes (Mehta et al., 2024). As a codebase grows, it tends to accumulate complexity; at some point, even minor updates become slow and risky to push through. Breaking that logic apart into smaller, more focused sub-services (whether microservices or third-party vendor components) helps distribute responsibility and reduces the chance that a change in one place quietly breaks something elsewhere (Apel et al., 2018). It also becomes considerably easier to introduce new functionality or trace the source of a bug. Each component can serve as a focused, domain-driven unit with a well-defined and deliberately limited scope. Such services can be deployed independently, scaled on their own, tested in isolation. Breaking a monolith apart into a distributed system also tends to have a positive effect on CI/CD pipelines – teams can release more frequently and with less coordination overhead (Giamattei et al., 2024; Saavedra et al., 2025). This architectural shift also has organizational consequences. Agile teams can structure their work around individual services, with each service developed and maintained by a team that owns it "end to end" – without constant dependency on "what others are doing".

In practice, a single service almost never covers the full scope of what a complex business process demands. More often, several services need to work together – composed into something larger that can actually deliver on the functional requirement at hand (Claro et al., 2006). Service composition is essentially about building new combinations out of parts that already exist. Each sub-service brings its own piece of functionality, logic, by invoking components in a coordinated sequence, the system produces behavior that none of them could achieve

on their own. The output of one service feeds into the next, forming a dependency chain that – taken as a whole – delivers a higher-level value. Load balancing operates on a different axis. Rather than orchestrating, it concerns itself with how incoming requests are spread across available infrastructure – whether that means different clouds, data centers (Kamath & B, 2025). Its purpose is to avoid resource overload, improve overall system performance, and reduce response time.

The aim of this paper is to bring order to how service composition and load balancing are understood – both: from a technical standpoint and from the perspective of real business practice. In much of the existing literature, the two concepts blur into one another and are not clearly separated, particularly when the discussion gravitates toward multi-criteria optimization rather than the cost-driven, operationally grounded decisions that organizations actually face. Studies that address both concepts together in a precise and unambiguous way are rare, which is part of what motivated this work. The methodology combines a structured literature review with informal participant observation.

The rest of the paper is structured as follows. Section 2 describes the research methodology. Section 3 presents the findings from the literature review and offers a critical discussion of how both concepts are treated across the reviewed studies, including the contradictions that emerge from that analysis. Section 4 draws conclusions and points to directions that seem worth pursuing in future research.

II. METHODS

This study combines two methodological approaches: a structured literature review and informal participant observation. The rationale for this design is straightforward – understanding how service composition and load balancing are theorized in the literature and how they actually work out in a business environment.

First, a structured literature review (Kofod-Petersen, 2015) was conducted to examine how service composition and load balancing are treated in studies that explicitly recognize both concepts as primary research topics. Papers in which either notion appears only in passing, or as a secondary concern, were deliberately excluded. The review quickly surfaced a notable gap: there are relatively few studies that analyze service composition and load balancing together, particularly in the context of actual production environments. The focus was therefore placed on publications that clearly define both concepts and say something meaningful about how they relate to each other in practice – whether that means mutual reinforcement or conflict. This selective approach was an aware choice – casting the net wider would have added volume without adding much depth.

Following the contextual research framework proposed by (Kofod-Petersen, 2015), the review was guided by three research questions:

RQ1 (Problem): How are load balancing and service composition defined in the literature that explicitly addresses both topics?

RQ2 (Limitations): How do the definitions of load balancing and service composition intersect, overlap, constrain one another?

RQ3 (System & Implications): What insights does the literature provide regarding the joint application of both strategies in real-world, production business environments?

The remainder of the review protocol is structured as follows.

2.1 Data Sources

To ensure broad coverage while maintaining academic quality, four complementary data sources were used.

- Google Scholar (Falagas et al., 2025), offers wide interdisciplinary coverage of journals and conference proceedings.
- arXiv (Bagchi et al., 2024), included to identify emerging research and preprints in computer science and distributed systems.
- Web of Science (Mutlu Avinç & Yıldız, 2025), selected for its collection of peer-reviewed, high-quality publications.
- ScienceDirect (Tober, 2011), was selected as the established source for scientific and technical research.

Relying on a single database is a big weakness in structured reviews – relevant work gets missed, the picture that emerges is narrower than it should be. Using several sources, from indexed databases like Web of Science to repositories of emerging work like arXiv, makes it more likely that both well-established research and newer, still-developing directions end up in the final set of papers.

2.2 Time Frame

The review was scoped to study published between 2015 and 2025. This decade covers a period of considerable change in how distributed systems are built and operated – service-oriented architectures matured, cloud computing became the default assumption for many organizations, microservice-based designs moved from novelty to standard practice. Both service composition and load balancing sit squarely in the middle of that shift, which makes older literature less relevant than it might be in a more stable research area.

2.3 Search Strategy and Keywords

A consistent search strategy was applied across all databases, with queries restricted to article titles to ensure that both topics were central to the identified studies rather than peripheral mentions:

("load balance" OR "load balancing" OR "load-balance" OR "load-balancing") AND "service composition"

The search included several lexical variants of "load balancing" – this was necessary because the term is used differently across disciplines and research communities, a single phrasing would have missed a non-trivial portion of relevant work. Limiting searches to titles was a deliberate choice, consistent with selective strategies used in structured reviews when the goal is conceptual precision rather than exhaustive coverage. "Service composition" required no such variation – the term is stable and consistently used across the literature, so a single phrase was sufficient.

2.4 Study Selection and Quality Assessment

All publications retrieved within the defined time frame were collected first, after which duplicate records across databases were removed. What remained was then screened manually – the key criterion being whether a given paper actually engaged with both load balancing and service composition at a conceptual or definitional level, rather than just mentioning them in passing.

- Google Scholar returned four publications meeting the search criteria: (Dong et al., 2019; Hioual et al., 2017; Low et al., 2024; Wu, 2021).
- arXiv returned no relevant results within the specified period.
- Web of Science identified three publications, all of which overlapped with the Google Scholar results: (Dong et al., 2019; Hioual et al., 2017; Wu, 2021)
- ScienceDirect returned no relevant results within the specified period.

The small number of studies that made it through screening is itself telling – it confirms that research treating service composition and load balancing jointly, with any real conceptual clarity, remains scarce. Upon closer reading, all four publications were found to engage with both concepts as distinct but related frameworks for building complex services. This kind of manual validation step matters precisely because automated filtering cannot assess whether a paper is actually grappling with a concept or merely referencing it. The extracted findings are discussed in the following chapter.

2.5 Informal Participant Observation

Second, informal participant observation was employed to complement the literature review with insights from real business environments. This empirical perspective serves as a counterbalance to theoretical assumptions derived from academic sources.

A key element of this approach is the author's long-term professional experience as a Java web services developer in a large Polish bank. Over approximately five years, he worked across multiple projects and teams. In line with the methodological perspective described in (Glinka & Czakon, 2021; Kaczmarek, 2017), this form of unstructured participant observation does not rely on predefined hypotheses. Instead, it allows organizational practices, recurring patterns to emerge naturally through everyday professional activity.

The organization operates across a genuinely heterogeneous technological landscape – one end of the spectrum is occupied by heavily legacy core banking systems, while the other is built around modern, cloud-native microservice architectures. This kind of environment is analytically valuable because it forces the same fundamental challenges around quality, scalability, reliability to be solved in very different ways, depending on the context. Watching how teams approach service design, integration and load balancing under such varied technical and organizational conditions offers a perspective that a more uniform environment simply could not provide – particularly when the focus is on how complex services are actually composed within a large-scale ecosystem. The observation process itself unfolded across four stages.

- The first stage involved entering the organizational environment of a company that develops and maintains a heterogeneous portfolio of services – spanning multiple technologies and serving distinct business purposes. In practical terms, this meant going through a standard hiring process, including technical and HR interviews, followed by onboarding that covered the company's IT landscape, architectural principles, the overall structure of the application mesh.
- The second stage shifted toward observation. The focus here was on identifying recurring patterns and practices related to web service composition and load balancing as they actually appeared in day-to-day work. This was achieved through repeated exposure to different teams and their ongoing challenges – observing how services are created, integrated, composed, balanced in practice, what kinds of problems tend to surface along the way.
- The third stage moved from observation to participation. This involved hands-on

contribution to projects directly concerned with these mechanisms – writing application Java code, developing Infrastructure as Code (IaC) (Masoumzadeh et al., 2025) configurations, building supporting scripts. Working at this level provided a perspective that purely observational methods could not offer.

- The final stage was analytical. The insights gathered throughout the preceding stages were consolidated into informal notes and reflections, which were then used to draw broader conclusions about how load balancing and service composition are actually approached and evolved within a business context – capturing the gap between how these concepts appear in the literature and how they play out in practice.

III. RESULTS AND DISCUSSION

3.1 Findings from the Literature

RQ1 (Problem): How are load balancing and service composition defined in the literature that explicitly addresses both topics?

(Hioual et al., 2017) define service composition as an automated and dynamic process of assembling multiple reusable cloud services into a single composite service capable of fulfilling complex user requirements that no individual service could satisfy on its own. The process explicitly incorporates non-functional criteria, most notably cost and response time, and is framed as an essential phase of the cloud service lifecycle in multi-cloud environments. Load balancing, in the same work, is understood as the distribution of incoming requests across different clouds or service replicas, with the aim of preventing resource overload and bringing down response times. What makes this treatment interesting is that load balancing is not positioned as a purely infrastructural concern – rather, it is embedded within the composition process itself as a decision criterion.

(Dong et al., 2019) present service composition as a structured process of integrating multiple independent, atomic services into an execution sequence – one that satisfies both the user's functional requirements and global QoS constraints (Balakrishnan & Sangaiah, 2017). Services are formally described by their inputs, outputs and QoS attributes. Load balancing in this work is not treated as a standalone infrastructure mechanism. Instead, it is defined as the controlled distribution of incoming request flows across multiple feasible execution paths.

(Wu, 2021) defines service composition as the process of constructing an executable composite service by selecting and orchestrating concrete web services to replace workflow tasks – the selection must satisfy both functional requirements and non-functional QoS constraints. Load balancing, in the same work, is understood as the regulation of service workloads in a way that prevents demand from concentrating on a small subset of high-quality providers while others remain underutilized.

(Low et al., 2024) conceptualize service composition as a modular approach to building applications by integrating interoperable services; each service encapsulates a specific business function and exposes it through a standardized interface. Load balancing is defined as the systematic distribution of incoming requests across multiple servers, with the purpose of sustaining performance and availability within such composed systems. What is notable in this treatment is that load balancing is not positioned as a background infrastructure concern. When the two mechanisms are considered together, load balancing becomes an integral part of how a composed system actually operates.

RQ2 (Limitations): How do the definitions of load balancing and service composition intersect, overlap, or constrain one another?

(Hioual et al., 2017) reveal a tension that runs in both directions. Traditional composition approaches that optimize primarily for cost or response time tend to concentrate requests on a narrow subset of preferred services or clouds – which sooner or later produces load imbalance, degraded performance and reduced reliability. The opposite problem is equally real: enforcing load balancing without any regard for user preferences risks violating service-level expectations and undermining the perceived quality of the composite service. The practical consequence is that composition decisions in this work are explicitly constrained by load considerations. The composer cannot simply select the best-performing option – it must also account for which clouds are already overloaded, which are underutilized, and what trade-offs between cost, response time and system stability are actually acceptable in a given situation.

Authors in (Dong et al., 2019) argue that traditional composition methods – those built around finding a single optimal QoS path – inevitably concentrate demand on a narrow set of high-performing services. The problem compounds over time: load imbalance grows, performance deteriorates; under high request rates the risk of outright service failure becomes difficult to ignore. The response proposed in the article is to reframe composition itself as a load-aware problem. Rather than searching for one optimal path, the system must generate and evaluate multiple feasible paths – selected not only on the basis of QoS satisfaction but

also with reference to service processing capacity and actual request arrival rates.

(Wu, 2021) frames the intersection of these two mechanisms as a problem of competing constraints. Optimizing purely for QoS tends to concentrate demand on a small number of high-performing services, which leads to overload, reduced availability and a gradual erosion of long-term system robustness. Enforcing load balance without any QoS flexibility creates the opposite problem – composite service quality deteriorates and user constraints go unmet. The deeper issue, as the article makes clear, is that the two concepts are tightly coupled in a way that makes independent treatment of either one self-defeating. Composition decisions directly shape how load is distributed across services, and load conditions in turn determine what QoS levels are actually achievable.

(Low et al., 2024) captures a tension that sits in the middle of the discussion. Service composition brings genuine benefits – greater flexibility, reusability, the ability to integrate heterogeneous components – but that same diversity makes load distribution considerably harder to manage. Services can differ substantially in capacity, workload patterns, execution cost, which means a uniform balancing strategy rarely holds up under real conditions. Dynamic load balancing, the authors argue, fits more naturally into this picture because it can adapt to fluctuating workloads and uneven service performance rather than assuming stability that is unlikely to exist.

RQ3 (System & Implications): What insights does the literature provide regarding the joint application of both strategies in real-world, production business environments?

From a practical perspective, (Hioual et al., 2017) offer a concrete illustration of how both strategies can be applied together. The proposed model uses an agent-based architecture that combines multi-criteria decision-making techniques – specifically AHP and TOPSIS – with the Contract Net Protocol (Saoud et al., 2025). The result is a composition mechanism that can adapt to fluctuating workloads and heterogeneous cloud offerings while keeping resource utilization reasonably balanced across providers.

(Dong et al., 2019) ground their discussion in production-oriented environments – specifically cloud service platforms handling concurrent, high-volume request traffic. The work demonstrates that multipath QoS-aware composition can meaningfully improve system robustness by distributing requests across several execution paths rather than committing to a single one. Under growing demand, this approach maintains acceptable QoS levels.

(Wu, 2021) makes the point that production environments – cloud computing platforms and collaborative manufacturing settings among them – are inherently competitive and multi-user in nature. The proposed DCBC method shows how load balancing and composition can be operationalized together through proactive QoS adjustment, probabilistic service selection and workload-aware decision-making – yielding higher request fulfillment rates and acceptable QoS trade-offs even at scale.

(Low et al., 2024) discuss the joint application of service composition and dynamic load balancing in cloud-based and distributed production environments. Concrete examples such as Amazon Elastic Load Balancing and NGINX illustrate that adaptive, monitored load balancing is not optional in these settings – it is what makes it possible to meet SLAs and sustain service quality when demand fluctuates in ways that cannot be predicted in advance.

3.2 Summary of Key Patterns Across Literature

Across all four analyzed articles, the definitions of both concepts follow a one consistent pattern. Service composition is understood as the process of creating a new service by integrating multiple existing ones (with selection governed by non-functional metrics, that the composite service must satisfy to deliver an acceptable user experience). Load balancing is typically treated as a sub-mechanism within that process rather than as a standalone concern. Its role is to regulate how incoming requests are distributed across available service instances.

The underlying rationale for combining the two is straightforward. QoS-aware composition tends to repeatedly select the same high-performing services – and repeated selection eventually produces the overload it was trying to avoid. Load balancing introduces a degree of elasticity by enabling the system to route requests across different instances and execution paths. Importantly, it operates not at the level of composition logic itself but at the level of individual service instances in practice.

Figure 1 presents the general idea discussed in the analyzed articles. If a complex service call must integrate Functionality 1 and Functionality 2, it should first create a QoS-aware composition of the available services for both functionalities. After selecting services that meet the functional requirements while maintaining a high-quality level, the load-balancing mechanism chooses specific service instances. It may either select an instance for each service separately (e.g. A1 followed by Z2) or choose the entire instance composition at once (e.g. B2 and X1).

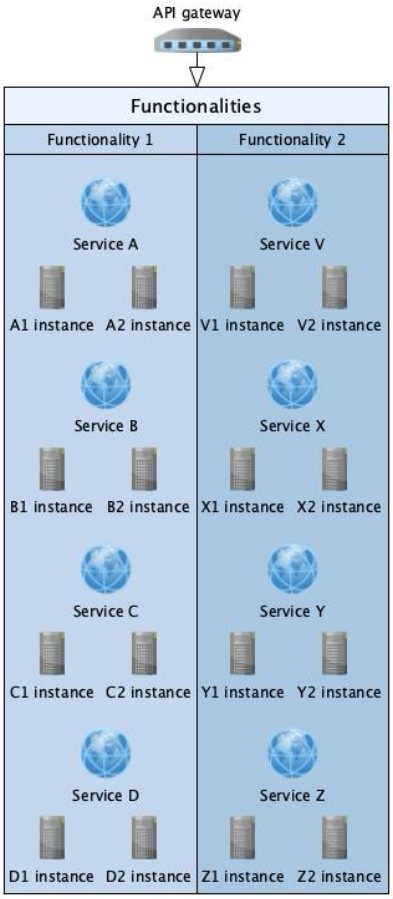

Figure 1. General conceptual relationship between web service composition and load balancing in the literature

3.3 Theoretical Assumptions vs. Observed Practice: A Critical Discussion

According to the reviewed studies, service composition is generally understood as the process of constructing a logical workflow by combining multiple independent services to fulfill a user-defined goal. Service selection based on non-functional requirements is typically framed as a multi-criteria optimization problem, with the objective of maximizing or balancing predefined QoS metrics at the level of the overall composition. Selecting services according to QoS parameters carries an implicit assumption – multiple services exist which provide equivalent functionality. For any given task, there must be more than one available service capable of performing the same operation, differing in non-functional characteristics such as: response time, cost, availability. In theoretical terms, this assumption is what makes optimization-based composition possible. Examined from the perspective of actual organizational environments, however, it raises concerns impossible to dismiss.

In production systems maintaining multiple & distinct services that perform the same business function is rarely feasible (this refers to separate services themselves, not different instances of a single service). Organizations are driven by cost efficiency and profit maximization. Developing, maintaining two or more fully independent services that provide identical functionality would lead to unnecessary duplication of effort and expense. From a business perspective, paying twice for the same capability is difficult to justify and unrealistic.

Even if such redundancy was hypothetically introduced, additional challenges would emerge at the technical level. For services to be interchangeable within a composition process, they would need to expose identical interfaces, as the system operates on the same data at the moment of service selection. This requirement extends beyond adopting the same communication paradigm (e.g. REST, SOAP (Andrianjaka et al., 2021; Juneau & Telang, 2022)). It also entails identical endpoint definitions, HTTP methods (Chodak et al., 2020), request parameters, payload structures, response schemas, exception-handling mechanisms. Achieving such strict interface equivalence across services developed by different teams within the same organization (or, more critically, by different external vendors) is highly unlikely in practice, again – unrealistic.

Additionally, functional equivalence at the interface level is insufficient if services depend on underlying data (most business cases). Fully interchangeable services would require access to the same state and historical data, often stored in relational or NoSQL databases and/or caches (Miyamoto et al., 2021). Sharing a common database across multiple independently developed services is regarded as an architectural total antipattern; it leads to tight coupling, undermines service autonomy, introduces severe risks related to consistency, scalability, fault isolation. But if services do not share data, it becomes unclear: how they even could correctly fulfill requests that rely on persistent or historical information.

These considerations lead to a broader question: what does web service composition mean in real-world production web systems? In practice, service composition is most often realized through an architectural component such as an API gateway/service dispatcher (Zhou et al., 2021). This component exposes a single endpoint to clients and orchestrates calls to downstream services. There is no *magical* composition mechanism; rather, the composer itself is simply another service responsible for routing logic.

Viewed from this perspective, the feasibility of dynamically selecting services based on QoS parameters becomes questionable. Such selection would require the execution of optimization algorithms at runtime – potentially involving historical measurements and complex

metaheuristics, before each client request is forwarded. Given typical latency constraints in microservice and distributed systems, as well as the computational cost of advanced optimization techniques (e.g. evolutionary multi-criteria algorithms (Odu, 2013; Suciu et al., 2013; Wang et al., 2019)), the overhead of such processing could outweigh its potential benefits. In many cases, the time and resources required to compute an optimal selection would exceed those of simpler routing strategies. This is the engineering overhead. Designing and implementing a dispatcher or gateway capable of handling such complex decision logic would increase system complexity.

Web service composition becomes practically meaningful when the notion of independent services is interpreted not as different services, but rather as multiple instances of the same service. In this interpretation, a single service implementation is deployed in several locations to improve availability and fault tolerance. Each instance exposes the same interface and follows the same service contract, which makes them interchangeable.

In this context, most of the reviewed papers describe load balancing as the mechanism responsible for selecting the appropriate service instance based on the current workload. It is typically presented as a component of the overall composition mechanism. This implies that once the composition is determined, the appropriate instance of each selected service is chosen. Such an interpretation raises a question: do web service composition and load balancing partially overlap, are they therefore inconsistent with each other?

From its definition, web service composition is the selection of an execution path. This means that before a request is dispatched to the first service, all subsequent services are already determined. Once the optimization algorithm selects the optimal path, it is executed without further changes across all service instances. In contrast, load balancers make decisions dynamically, just before each service invocation. In a traditional composition scenario, a composition algorithm might select a fixed sequence of service A, B, C instances, such as A1, B4, C9. A load balancer, however, would first choose an instance of service A (e.g. A2). After the request is processed by service A, it would then decide which instance of service B should be invoked next. Load balancers typically rely on well-known algorithms, such as round robin, to distribute requests among instances of the same service (Priya & Rajendran, 2024). This difference highlights a paradox. Service composition assumes a predefined execution path, while load balancing selects instances dynamically at runtime. Combining both mechanisms, predefining the path while choosing instances on the fly, appears contradictory (Figure 2).

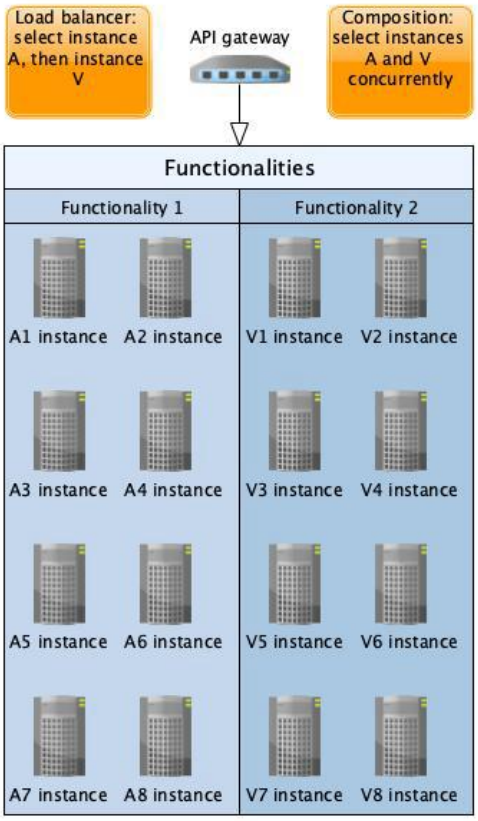

Figure 2. Conceptual contradiction between service composition and load balancing

IV. CONCLUSION

This paper aims to stimulate discussion on the coexistence of service composition and load balancing in both theoretical research and real production environments. The relationship between these two patterns remains insufficiently explored and conceptually unclear. This lack of clarity motivates the questions raised in this study. If service composition is to be treated as a practically applicable industrial problem, a clear distinction must be established between composition mechanisms and the load-balancing strategies with attention to their real business impact. Without considering cost-related factors, these problems risk remaining at an abstract level, without practical relevance.

This work offered a clear perspective on several questions. First, it examined how service composition and load balancing are defined in papers that address both concepts simultaneously, where the two approaches actually intersect in that literature. It also checked at how both mechanisms are described in terms of real-world applications – what the papers say about their use in production environments.

Despite of the literature, the study took on a set of more practice-driven questions. Why is it so rare, in real organizations, to maintain multiple distinct services performing the same business function; what drives that reality from both a technical and a business standpoint? What does the orchestration problem look like in live production settings, how does it shape the way load balancers and composers are actually understood and used? And why might selecting services based on historical QoS data be a questionable strategy when applied outside controlled research conditions?

The study also explored how the notion of "independent" services translates – or fails to translate – into business environments. Ultimately it addressed what may be the central paradox of the whole discussion: why, in practice, service composition and load balancing tend to work against each other, particularly when the moment of service instance selection is taken into account.

Future work may include experimental studies. These could involve the implementation of API gateways or dispatchers that incorporate heuristic-based service composition strategies, followed by a comparison with widely used load-balancing solutions in selected architectural settings. Such experiments would make it possible to evaluate system behavior under conditions of limited historical QoS data, as well as to assess the overhead introduced by collecting and processing such data.

AUTHOR'S CONTRIBUTION

Conceptualization, Methodology, Writing – original draft, Writing – review and editing.

REFERENCES

Andrianjaka, R. M., Hajarisena, R., Mihaela, I., Thomas, M., Sorin, I., & Raft, R. N. (2021). Automatic generation of Web service for the Praxeme software aspect from the ReLEL requirements model. *Procedia Computer Science*, *184*, 791–796. https://doi.org/10.1016/j.procs.2021.03.098

Apel, S., Hertrampf, F., & Späthe, S. (2018). Microservice Architecture Within In-House Infrastructures for Enterprise Integration and Measurement: An Experience Report. *Communications in Computer and Information Science*, 3–17. https://doi.org/10.1007/978-3-319-93408-2_1

Bagchi, C., Malmi, E., & Grabowicz, P. (2024). *Effects of Research Paper Promotion via ArXiv and X* (Version 2). arXiv. https://doi.org/10.48550/ARXIV.2401.11116

Balakrishnan, S. M., & Sangaiah, A. K. (2017). Integrated QoUE and QoS approach for optimal ser-vice composition selection in internet of services (IoS). *Multimedia Tools and Applications*, 22889–22916. https://doi.org/10.1007/s11042-016-3837-9

Chodak, G., Suchacka, G., & Chawla, Y. (2020). HTTP-level e-commerce data based on server access logs for an online store. *Computer Networks*, *183*, 107589. https://doi.org/10.1016/j.comnet.2020.107589

Claro, D. B., Hao, J.-K., & Albers, P. (2006). Web Services Composition. In *Semantic Web Services, Processes and Applications* (pp. 195–225).

Dong, X., Liu, Q., Lu, D., Ma, S., & Zheng, J. (2019). QoS-Aware Path Finding and Load Balancing in Service-Composition. *2019 International Conference on Networking and Network Applications (NaNA)*, 409–414. https://doi.org/10.1109/NaNA.2019.00077

Falagas, M. E., Paliogianni, P. M., Kontogiannis, D. S., Ragias, D., & Johnson, E. (2025). Google Scholar as a Resource for Systematic Reviews in Clinical Medicine. *Journal of Evaluation in Clinical Practice*, *31*(5), e70206. https://doi.org/10.1111/jep.70206

Felício, D., Simão, J., & Datia, N. (2023). RapiTest: Continuous Black-Box Testing of RESTful Web APIs. *Procedia Computer Science*, *219*, 537–545. https://doi.org/10.1016/j.procs.2023.01.322

Giamattei, L., Guerriero, A., Pietrantuono, R., Russo, S., Malavolta, I., Islam, T., Dînga, M., Koziolek, A., Singh, S., Armbruster, M., Gutierrez-Martinez, J. M., Caro-Alvaro, S., Rodriguez, D., Weber, S., Henss, J., Vogelin, E. F., & Panojo, F. S. (2024). *Monitoring tools for DevOps and microservices: A systematic grey literature review*. https://doi.org/10.5445/IR/1000166110

Glinka, B., & Czakon, W. (2021). *Podstawy Badań Jakościowych*. Polskie Wydawnictwo Ekonomiczne S.A.

Hioual, O., Boufaïda, Z., & Hemam, S. M. (2017). Load balancing, cost and response time minimisation issues in agent-based multi cloud service composition. *International Journal of Internet Protocol Technology*, *10*(2), 73. https://doi.org/10.1504/IJIPT.2017.085187

Juneau, J., & Telang, T. (2022). RESTful Web Services. In *Java EE to Jakarta EE 10 Recipes* (pp. 511–530).

Kaczmarek, Ł. (2017). Między survey research a obserwacją uczestniczącą: Rozdarcia metodologiczno-tożsamościowe w polskiej etnologii/antropologii kulturowej w XXI wieku. *Etnografia. Praktyki, Teorie, Doświadczenia*, *2*. https://doi.org/10.4467/254395379EPT.16.006.6485

Kamath, B. S., & B, M. K. (2025). Achieve a Dynamic and Reliable Web Service using Network Load Balancer on Cloud Platforms. *2025 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER)*, 483–489. https://doi.org/10.1109/DISCOVER66922.2025.11258938

Kofod-Petersen, A. (2015). *How to do a structured literature review in computer science*.

Low, W. K., Ramasamy, R. K., & Rajendran, V. (2024). Adaptive load balancing strategies in service composition for improved system performance. *Journal of Infrastructure Policy and Development*, *8*(13), 8967. https://doi.org/10.24294/jipd8967

Masoumzadeh, S., Saavedra, N., Maipradit, R., Wei, L., Ferreira, J. F., Varró, D., & McIntosh, S. (2025). Do Experts Agree About Smelly Infrastructure? *IEEE Transactions on Software Engineering*, *51*(5), 1472–1486. https://doi.org/10.1109/TSE.2025.3553383

Mehta, G., Pothineni, B., Parthi, A. G., Maruthavanan, D., Veerapaneni, P. K., Jayabalan, D., & Sankiti, S. R. (2024). Revisiting Monoliths: A Pragmatic Case for Transitioning from Microservices Back to Monolithic Architectures. *IJARCCE*, 328–337. https://doi.org/10.17148/IJARCCE.2024.131251

Miyamoto, M., Kawashima, R., & Matsuo, H. (2021). Transparent Relational Database Caching Based on Storage Engines Using In-memory Database. *2021 Ninth International Symposium on Computing and Networking Workshops (CANDARW)*, 444–448. https://doi.org/10.1109/CANDARW53999.2021.00082

Mutlu Avinç, G., & Yıldız, A. (2025). A bibliometric and systematic review of scientific publications on metaverse research in architecture: Web of science (WoS). *International Journal of Technology and Design Education*, *35*(2), 825–849. https://doi.org/10.1007/s10798-024-09918-1

Odu, G. (2013). Review of Multi-criteria Optimization Methods – Theory and Applications. *IOSR Journal of Engineering*, 1–14. https://doi.org/10.9790/3021-031020114

Priya, S. S., & Rajendran, T. (2024). Load balancing using improved weighted round robin algorithm in cloud computing environment. *International Journal of Cloud Computing*, *13*(5), 463–484. https://doi.org/10.1504/IJCC.2024.142205

Saavedra, N., Ferreira, J. F., & Mendes, A. (2025). *InfraFix: Technology-Agnostic Repair of Infrastructure as Code* (Version 2). arXiv. https://doi.org/10.48550/ARXIV.2503.17220

Saoud, A., Lachgar, M., Hanine, M., Dhimni, R. E., Azizi, K. E., & Machmoum, H. (2025). decideXpert: Collaborative system using AHP-TOPSIS and fuzzy techniques for multicriteria group decision-making. *SoftwareX*, *29*, 102026. https://doi.org/10.1016/j.softx.2024.102026

Suciu, M., Pallez, D., Cremene, M., & Dumitrescu, D. (2013). *Adaptive MOEA/D for QoS-based web service composition*. 73–84. https://doi.org/10.1007/978-3-642-37198-1_7

Tober, M. (2011). PubMed, ScienceDirect, Scopus or Google Scholar – Which is the best search engine for an effective literature research in laser medicine? *Medical Laser Application*, *26*(3), 139–144. https://doi.org/10.1016/j.mla.2011.05.006

Wang, C., Ma, H., Chen, G., & Hartmann, S. (2019). *A Memetic NSGA-II with EDA-Based Local Search for Fully Automated Multiobjec-tive Web Service Composition*. https://doi.org/10.1145/3319619.3321937

Wu, X. (2021). A Dynamic QoS Adjustment Enabled and Load-balancing-aware Service Composition Method for Multiple Requests. *KSII Transactions on Internet and Information Systems*, *15*(3). https://doi.org/10.3837/tiis.2021.03.005

Zhou, D., Chen, H., Cheng, G., He, W., & Li, L. (2021). SecIngress: An API gateway framework to secure cloud applications based on N-variant system. *China Communications*, *18*(8), 17–34. https://doi.org/10.23919/JCC.2021.08.002