XXI Escola Regional de Redes de Computadores (ERRC). DOI: 10.5753/errc.2024.4560

Artigo Completo/Full Paper

RouteBastion: Architecting a Broker for VRP APIs

Pietro Vieira • pietrovieira.aluno@unipampa.edu.br
Universidade Federal do Pampa (UNIPAMPA)

Bruno Dalmazo • dalmazo@furg.br
Universidade Federal do Rio Grande (FURG)

Diego Kreutz • diegokreutz@unipampa.edu.br
Universidade Federal do Pampa (UNIPAMPA)

Rodrigo Brandão Mansilha • mansilha@unipampa.edu.br
Universidade Federal do Pampa (UNIPAMPA)

Abstract. The Vehicle Routing Problem (VRP) is a complex issue that requires advanced algorithmic problem-solving techniques and computational power. Major tech companies have developed cloud-based APIs to address this issue, offering solutions varying in cost, capabilities, and performance. However, selecting the right one for a given context is a challenging task, requiring balancing trade-offs, understanding long-term impacts, and adapting to future changes. RouteBastion, a Software as a Service (SaaS) platform, aims to unify and simplify the use of VRP-related APIs. Its modular, scalable architecture allows users to compare, select, and switch between APIs as their needs evolve. RouteBastion provides seamless integration, dynamic API selection, and decision-making, making it an intelligent middleware that enhances the flexibility and accessibility of vehicle route optimization. The paper will explore the system architecture and performance considerations to ensure RouteBastion can adapt to evolving industrial requirements.

Palavras-chave: Problema de Roteamento de Veículos • PRV • Arquitetura de Software • Software como Serviço • Sistemas Distribuídos

Keywords: Vehicle Routing Problem • VRP • Software Architecture • Software as a Service • Distributed Systems

1 Introduction

The Vehicle Routing Problem (VRP) is a complex optimization problem that has gained significant attention due to its real-world applications and the opportunities brought by emerging technologies such as artificial intelligence and IoT[1, 2]. Efficiently managing a fleet of vehicles in real-time under various constraints can enhance logistics and customer satisfaction. However, solving the VRP is classified as NP-hard, making it computationally challenging for large instances, especially under short time constraints[1].

To address the complexity of route optimization, large-scale technology companies provide cloud-based APIs solutions, using advanced heuristic algorithms and scalable cloud infrastructure. The selection of an appropriate API creates opportunities for specialized brokers to manage the arbitration process, orchestrating services to ensure they meet specific business requirements [3, 4]. However, optimizing service-level agreements (SLAs) while balancing performance, cost, and customization remains a challenging task [5].

This paper introduces RouteBastion, a Software-as-a-service (SaaS) middleware solution designed to unify multiple VRP APIs into a single platform. RouteBastion abstracts the complexity of comparing and integrating different APIs by providing a seamless, centralized service where users can dynamically select the most suitable API for their needs. The platform's architecture is built on principles of modularity and scalability, ensuring it can accommodate the constantly evolving landscape of route optimization services.

The paper presents a problem analysis (Section 2), introduces the architecture proposal (Section 3), evaluates the proposed architecture (Section 4), and concludes with a summary of findings and implications for future research (Section 5).

2 Problem Analysis

As stated in Section 1, the VRP is a complex challenge in academia and industry due to its complex real-world applications. Companies such as Google and Microsoft offer VRP optimization services through APIs, namely Google Optimization API and Azure Maps Route Service. However, selecting the most suitable API for a use case remains a significant problem. The RouteBastion Broker addresses this issue by offering a unified middleware that simplifies and automates interaction

Pietro Vieira is a Student at the Graduate Program in Software Engineering (PPGES) at UNIPAMPA. Bruno Dalmazo is an Assistant Professor at FURG. Rodrigo Mansilha and Diego Kreutz are Professors at PPGES/UNIPAMPA.

with various VRP APIs. Its key differentiator lies in its intelligent decision-making mechanism, which leverages multiple metrics to select the optimal API for each request dynamically. It removes the burden of manual comparison and aligns the chosen API with the customer's specific needs.

2.1 Requirements

RouteBastion's practical applications include optimizing vehicle itineraries (with or without restrictions such as time windows, cargo capacity, and fuel consumption) and querying the current status of optimizations. To support these applications, we outline the functional and non-functional requirements in Tables 1 and 2, respectively. Most requirements were derived from analyzing existing VRP API endpoints to ensure a similar client interface with minimal changes, while others focus on core RouteBastion operations.

Table 1. Functional Requirements.

ID	Requirement
FR01	As an External Software System (ESS), I want to register and manipulate (query, update, delete) all my limitations, such as the available budget and security/performance/availability concerns, in order to tailor my needs for selecting a Cloud Provider.
FR02	As an ESS, I want to request an optimization by sending waypoints and vehicles involved on the route, to start a new optimization.
FR03	As an ESS, I want to cancel an optimization by sending an optimization identifier, in order to cancel a running optimization.
FR04	As an ESS, I want to query my running optimizations and their progresses, in order to get to know the progress of my optimizations
FR05	As an ESS, I want to query my previous completed optimizations, in order to access historical data.
FR06	As an ESS, I want to be able to opt-out the Selection Algorithm by choosing a pre-defined VRP API based on the available ones, in order to use an API that suits my needs.
FR07	As the Broker API, I should periodically request the status of an optimization and memoize the result in order to reduce the system overall latency.
FR08	As the Broker API, I should balance the workload between cloud providers, in order to not be blocked from making new requests to a specific cloud provider.
FR09	As the Broker API, I should expose a list of available Cloud Providers, in order to be transparent with ESS's.
FR10	As the Broker API, I should periodically run the Selection Algorithm and memorize the results, in order to know the best available VRP APIs providers to reduce the latency of requests.
FR11	As the Broker API, I should queue pending optimization requests if the selected cloud provider is busy (i.e. the maximum of requests have been achieved).

2.2 Related Work

Some works present solutions to VRP API usage [6–8], however, in the context of cloud service/provider selection, previous research has introduced brokerage-

Table 2. Non Functional Requirements.

ID	Requirement
NFR01	The system should ensure data integrity and source integrity (i.e., non-repudiation) to provide accountability.
NFR02	The system should ensure confidentiality.
NFR03	The system should ensure high availability through measures that minimize downtime for operations such as deployment and error recovery.
NFR04	The system should have low latency to handle a high number of requests.
NFR05	The system should be easy to maintain and extend.
NFR06	The system should be scalable to handle a high number of requests while maintaining high throughput and low latency.
NFR07	The system should be elastic, reducing resource consumption while handling low throughput of requests.
NFR08	The system should be fault-tolerant by using multiple replicas to ensure reliability for ESSs.
NFR09	The system should be interoperable to cope with different clients and VRP API providers.

based architectures that aim to address the complexity of choosing optimal cloud services for users. Sundareswaran et al. (2012) [9] proposed a cloud brokerage system, helping users select services tailored to their needs based on several factors such as cost, security, and performance. Their approach involves an indexing technique to efficiently manage large numbers of cloud service providers and a service selection algorithm that ranks providers according to user requirements.

Achar and Thilgam (2014) [10] proposed a broker-based architecture to address the challenge of choosing a suitable cloud provider from multiple available options. Their system introduces a cloud broker that evaluates and ranks providers based on factors such as cost, availability, and security. This architecture incorporates a Service Measurement Index (SMI), which helps differentiate cloud providers by their performance in various categories.

Rădulescu et al. (2016) [11] proposed a decision-making framework that addresses the complexity of weighting and ranking criteria for selecting cloud providers. Their framework combines the Decision-Making Trial and Evolution Laboratory (DEMATEL) method and Analytical Network Process (ANP) method, forming a hybrid approach (DANP) to better handle the interdependencies between various criteria, such as cost, performance and security.

Mukherjee et al. (2020) [12] studied the usage of the Harris Hawks Optimization (HHO) algorithm for cloud provider selection, comparing it's performance to techniques such as Teaching-Learning-Based Optimization (TLBO), and Jaya algorithms. This work highlights the growing importance of multi-criteria decision-making methods in cloud provider selection, a concept that aligns with the goals of RouteBastion in dynamically selecting the best VRP API based on a range of metrics.

Similarly, Baranwal and Vidyarthi (2014) [13] presents a framework for selecting the best service providers using a Ranked Voting Method. This framework addresses the challenge of choosing a cloud service provider by ranking providers based on Quality of Service (QoS) metrics, such as cost, availability and security.

Similar to the previously cited papers, RouteBastion aims to tackle the challenge of selecting the optimal Cloud Service Provider by acting as a unified service broker. It dynamically chooses the most suitable provider based on factors such as cost-effectiveness and performance, aligning with existing approaches for cloud service provider selection. However, it distinguishes itself by serving as a broker specifically for VRP APIs, a topic not addressed in the literature, to the best of our knowledge.

2.3 Related Systems

While not identical to RouteBastion, various commercial systems aim to provide a solution to the VRP problem, but also provide full functionality of a logistics enterprise software, being those:

- Onfleet: Onfleet [1] is a logistics management platform that provides route optimization, driver tracking, and delivery analytics. It integrates with multiple external systems and focuses on optimizing delivery routes for businesses. Although it doesn't aggregate multiple VRP APIs such as RouteBastion, it addresses the same need for efficient routing and logistics management but through a standalone platform.
- OptimoRoute: OptimoRoute [2] is another SaaS solution that focuses on route optimization, handling constraints similar to delivery windows, vehicle capacities, and driver schedules. It is tailored for small to medium-sized logistics businesses and provides an all-in-one routing solution. However, different from RouteBastion, it does not provide the capability to dynamically select different VRP APIs.

3 RouteBastion

While traditional solutions such as Google Cloud's or Azure's VRP APIs focus on providing a singular API interface for optimization, RouteBastion is architected to act as an intelligent broker that dynamically selects the most suitable API from multiple providers based on a variety of metrics, such as cost, performance, and input size limits. This API selection adds a layer of flexibility that is uncommon in existing systems, which generally require manual selection or are tied to a single provider. For such, we present an overview of RouteBastion architecture using the C4 model notation [14] in Figure 1. RouteBastion interacts with two main entities:

- **External Software System**: This represents any server-side application that interacts with RouteBastion to request route optimizations or query the status of ongoing optimizations. The external system relies on RouteBastion to route these requests to the most appropriate cloud provider based on dynamic factors such as cost and performance.
- **Cloud Provider**: These are VRP optimization API providers such as Google Cloud and Azure, which provide the computational resources and algorithms to solve the Vehicle Routing Problem. RouteBastion connects with these providers, leveraging their APIs to perform route optimizations for external software systems.

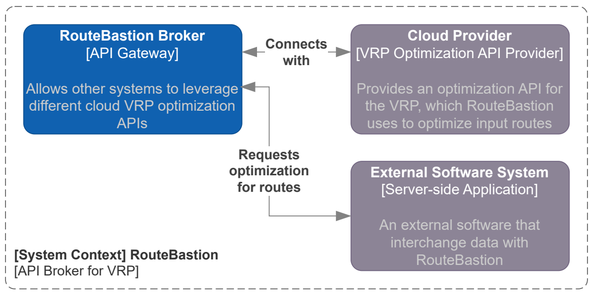
RouteBastion Broker
[API Gateway]
Allows other systems to leverage different cloud VRP optimization APIs
Connects with
Cloud Provider
[VRP Optimization API Provider]
Provides an optimization API for the VRP, which RouteBastion uses to optimize input routes
Requests optimization for routes
External Software System
[Server-side Application]
An external software that interchange data with RouteBastion
[System Context] RouteBastion
[API Broker for VRP]

Figure 1. System Context Diagram.

At the heart of this architecture is the RouteBastion Broker, which serves as the API gateway and manages the selection and routing of optimization requests to the cloud providers. The dynamic cloud provider selection algorithm poses unique challenges, as it must balance performance, cost, and other metrics without introducing significant overhead.

The container diagram, shown in Figure 2, delves deeper into the internal components of the RouteBastion Broker system, highlighting the service itself and databases that power the platform. The key components of the architecture include:

- **Broker API**: The Broker API is the main container, which acts as a stateless API gateway, handling authentication/authorization, request validation and

[1] https://onfleet.com/

[2] https://optimoroute.com/

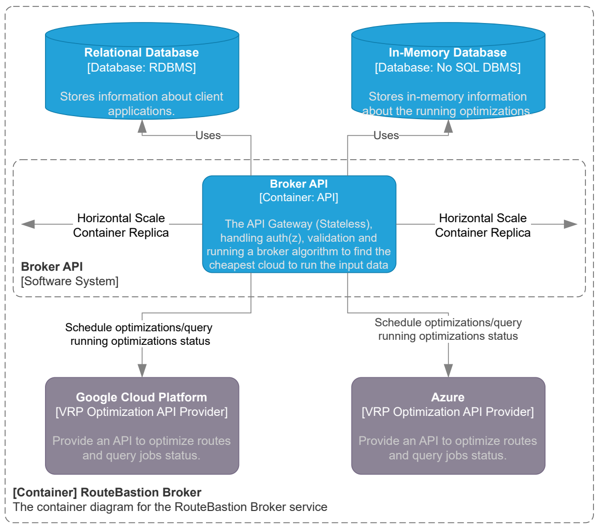
Relational Database
[Database: RDBMS]
Stores information about client applications.
In-Memory Database
[Database: No SQL DBMS]
Stores in-memory information about the running optimizations
Uses
Uses
Broker API
[Container: API]
The API Gateway (Stateless), handling auth(z), validation and running a broker algorithm to find the cheapest cloud to run the input data
Horizontal Scale Container Replica
Horizontal Scale Container Replica
Broker API
[Software System]
Schedule optimizations/query running optimizations status
Schedule optimizations/query running optimizations status
Google Cloud Platform
[VRP Optimization API Provider]
Provide an API to optimize routes and query jobs status.
Azure
[VRP Optimization API Provider]
Provide an API to optimize routes and query jobs status.
[Container] RouteBastion Broker
The container diagram for the RouteBastion Broker service

Figure 2. Container Diagram

running a broker algorithm that dynamically selects a cloud provider. The Broker API is horizontally scalable, meaning that additional replicas can be deployed to handle increased loads, ensuring the system can process high volumes of route optimization requests simultaneously.

- **Relational Database**: This component stores persistent data, such as information about client applications (e.g., API tokens for authorization), usage statistics, and other metadata required for managing the system's operations.
- **In-Memory Database**: This component is used to store in-memory data about running optimizations. It allows the system to quickly access real-time information, such as the current status of optimizations, which is essential for monitoring and querying the ongoing optimization processes.
- **Cloud Providers**: These are external services such as Google Cloud and Azure, which provide the actual optimization algorithms and computational power to solve the VRP. The Broker API schedules optimization jobs with these cloud providers and retrieves job statuses, allowing it to present the results back to the external software systems.

In summary, while existing systems provide powerful APIs for solving the VRP, RouteBastion differentiates itself by serving as a broker with intelligent, dynamic selection capabilities and a scalable, efficient architecture that optimizes for both performance and flexibility. This introduces new technical challenges, such as state synchronization and decision-making at scale.

4 Evaluation

4.1 Methodology

We use the Architecture Tradeoff Analysis Method (ATAM) [15], which is a structured evaluation technique used to assess the quality attributes of a software architecture. It helps stakeholders identify and prioritize architectural decisions, their effects on the system, and how well the architecture meets various quality attributes such as performance, scalability, modifiability, security, and usability. The process begins with establishing the context of the architecture, engaging stakeholders, and defining quality attribute scenarios. The final stages involve identifying sensitivity points, trade-offs, risks, and prioritizing risks for further investigation or refinement.

4.2 Architecture Analysis

4.2.1 Business Goals and Key Quality Attributes

RouteBastion aims to provide a reliable option to enterprises for selecting the best Cloud Provider for solving Vehicle Routing Problem considering their needs and restrictions. The main business goals of our solution are as follows.

- **Cost-effectiveness**: Selecting the most affordable API for route optimization from multiple cloud providers.
- **SLA Confidence**: Enhance customer assurance by consistently meeting or exceeding SLA commitments, thereby reinforcing our reputation for reliability and responsiveness in service delivery.
- **Maintainability**: Easy updates and expansions as new cloud providers or APIs become available.

We define the following key quality attributes to effectively measure if our systems matches the above goals.

- **Latency**: Measure the overhead on the elapsed time (i.e., delay) from request to response.
- **Scalability and Elasticity**: Measures if the system accommodates fluctuating demand for route optimizations.
- **Availability**: Measures whether a system is operational and accessible when required.
- **Reliability**: Measures whether a system consistently performs correctly without failure over time

4.2.2 Architectural Scenarios

Some potential scenarios that stress different quality attributes include:

- **Scenario 1 - API selection based on cost-effectiveness**: The broker must choose the cheapest available API without degrading performance.
- **Scenario 2 - Scalability under high load**: The RouteBastion Broker receives a large number of route optimization requests concurrently.
- **Scenario 3 - Cloud provider failure**: One of the cloud providers is unavailable, and the system must switch to another provider without significant delays.
- **Scenario 4 - Adding a new API provider**: Integrating a new cloud provider into RouteBastion without significant downtime or code changes.

4.2.3 Key Architectural Components and Trade-offs

We discuss how our architecture aligns with the quality attributes, outlining how each system component fulfills a key attribute.

Broker API The Broker API is stateless, horizontally scalable, and capable of handling multiple parallel requests through multithreading. This is achieved by using proven technologies such as the Go language. The design supports key quality attributes (KQAs) like scalability, elasticity, and availability, but requires careful management to prevent race conditions under high loads.

Databases RouteBastion's architecture uses a Relational Database Management System (RDBMS) (e.g., PostgreSQL) for persistent storage and an In-Memory DBMS (e.g., Redis) for caching. While the In-Memory DBMS reduces the load on the RDBMS, this combination supports key quality attributes (KQAs) such as latency, scalability, and availability, at the cost of increased code complexity to manage consistency and scaling as workloads grow.

Protocol We chose the protocol for service-to-service communication between the Broker API, cloud providers, and client services, prioritizing performance (throughput, latency, scalability) at the cost of maintainability. For instance, gRPC's compact binary format (Protocol Buffers) is highly efficient, minimizing overhead compared to REST, which typically uses JSON, resulting in greater throughput per second [16]. We compensate for the impact on maintainability by developing appropriate SDKs and documentation.

Cloud Provider Integration The architecture uses multiple cloud providers for route optimization, ensuring availability and reliability. However, this approach introduces latency and complexity in decision-making and may require extra handling to ensure consistent results across providers.

4.3 Discussion

Optimizing for the cheapest API can sometimes compromise performance, as switching between options can cause delays and slower response times. The system is horizontally scalable, but maintaining consistency between temporary and persistent states increases complexity. Proper coordination is necessary to avoid data consistency issues and race conditions. The architecture is flexible enough to accommodate new cloud providers and APIs.

4.3.1 Sensitivity Points and Risks

- **Database Bottlenecks**: While the In-Memory DBMS helps offload some traffic from the RDBMS instance, both databases must be carefully scaled to handle concurrent access under heavy loads.
- **API Decision Logic**: The decision-making logic in the Broker API for selecting the cheapest cloud provider is a critical point of sensitivity. Poor optimization in this algorithm could lead to performance degradation and bias towards a single Cloud Provider.
- **Communication Complexity**: While a performance-oriented protocol like gRPC enhances internal communication, its binary format and the need for precise interface definitions (Protobufs) may introduce development and debugging complexity, especially for developers more familiar with REST.

5 Conclusion and Future Work

In this paper, we present the architecture of RouteBastion, designed to provide an efficient and scalable solution by integrating various cloud-based APIs for VRP. Based an Architecture Tradeoff Analysis Method (ATAM), we belive RouteBastion is a robust and future-proof SaaS offering in the vehicle routing domain. Future work will focus on the Decision-Making Algorithm, the concrete implementation and evaluation (including security, usability, and performance) of RouteBastion as a whole, as well as case studies.

Declarations

Funding

This study was financed in part by: (i) FAPERGS (23/2551-0000773-8, 22/2551-0000841-0), and (ii) UE iTec/FURG (iTec-62).

References

1 Bogyrbayeva, A. *et al.* Machine Learning to Solve Vehicle Routing Problems: A Survey. *IEEE Transactions on Intelligent Transportation Systems*, v. 25, n. 6, p. 4754–4772, 2024. DOI: 10.1109/TITS.2023.3334976.

2 Shahin, R. *et al.* A survey of Flex-Route Transit problem and its link with Vehicle Routing Problem. *Transportation Research Part C: Emerging Technologies*, v. 158, p. 104437, 2024. ISSN 0968-090X. DOI: https://doi.org/10.1016/j.trc.2023.104437. Available from: https://www.sciencedirect.com/science/article/pii/S0968090X23004278.

3 Chauhan, S. S. *et al.* Brokering in interconnected cloud computing environments: A survey. *Journal of Parallel and Distributed Computing*, v. 133, p. 193–209, 2019. ISSN 0743-7315. DOI: https://doi.org/10.1016/j.jpdc.2018.08.001. Available from: https://www.sciencedirect.com/science/article/pii/S0743731518305719.

4 Khorasani, N. *et al.* Cloud Broker: A Systematic Mapping Study. *IEEE Transactions on Services Computing*, v. 17, n. 5, p. 2989–3005, 2024. DOI: 10.1109/TSC.2024.3442541.

5 Gyani, J.; Ahmed, A.; Haq, M. A. MCDM and Various Prioritization Methods in AHP for CSS: A Comprehensive Review. *IEEE Access*, v. 10, p. 33492–33511, 2022. DOI: 10.1109/ACCESS.2022.3161742.

6 MuÃ±oz-Villamizar, A. *et al.* Integration of Google Maps API with mathematical modeling for solving the Real-Time VRP. *Transportation Research Procedia*, v. 78, p. 32–39, 2024. 25th Euro Working Group on Transportation Meeting. ISSN 2352-1465. DOI: https://doi.org/10.1016/j.trpro.2024.02.005. Available from: https://www.sciencedirect.com/science/article/pii/S2352146524000589.

7 Chowlur Revanna, J. K.; Al-Nakash, N. Y. B. Impact of ACO intelligent vehicle real-time software in finding shortest path. *Software Impacts*, v. 19, p. 100625, 2024. ISSN 2665-9638. DOI: https://doi.org/10.1016/j.simpa.2024.100625. Available from: https://www.sciencedirect.com/science/article/pii/S2665963824000137.

8 Cavecchia, M.; Alves de Queiroz, T.; Iori, M., *et al.* An Optimization-Based Decision Support System for Multi-trip Vehicle Routing Problems. *SN Computer Science*, v. 5, p. 225, 2024. DOI: 10.1007/s42979-023-02540-3. Available from: https://doi.org/10.1007/s42979-023-02540-3.

9 Sundareswaran, S.; Squicciarini, A.; Lin, D. A Brokerage-Based Approach for Cloud Service Selection. *IEEE International Conference on Cloud Computing*, v. 5, 2012.

10 Achar, R.; Thilagam, P. S. A Broker Based Approach for Cloud Provider Selection. *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, v. 1, 2014.

11 Zoie, R. C. *et al.* A decision making framework for weighting and ranking criteria for Cloud provider selection. *International Conference on System Theory, Control and Computing (ICSTCC)*, v. 20, 2016.

12 Mukherjee, P. *et al.* HHO Algorithm for Cloud Service Provider Selection. *In*: 2020 IEEE International Women in Engineering (WIE) Conference on Electrical and Computer Engineering (WIECON-ECE). Dec. 2020. P. 324–327. DOI: 10.1109/WIECON-ECE52138.2020.9397936.

13 Baranwal, G.; Vidyarthi, D. P. A framework for selection of best cloud service provider using ranked voting method. *IEEE International Advance Computing Conference (IACC)*, v. 1, 2014.

14 Brown, S. *The C4 model for visualising software architecture*. Sept. 2024. Available from: c4model.com. Visited on: 5 Oct. 2024.

15 Kazman, R. *et al.* The Architecture Tradeoff Analysis Method. *IEEE International Conference on Engineering of Complex Computer Systems*, v. 4, 1998.

16 Aydemir, F.; Basçiftçi, F. Performance and Availability Analysis of API Design Techniques for API Gateways. *Arabian Journal for Science and Engineering*, 2024.