# • Architecture Selection for API: Rest vs GraphQL

Arsalan Imran
*Masters in Science - Computer Science Bishop's University*
Sherbrooke, Quebec, Canada
AIMRAN22@ubishops.ca

Dan Luo
*Masters in Science - Computer Science Bishop's University*
Sherbrooke, Quebec, Canada
DLUO23@ubishops.ca

Guan Wang
*Masters in Science - Computer Science Bishop's University*
Sherbrooke, Quebec, Canada
WGUAN2223@ubishops.ca

*Abstract*—**Given the variety of architectural models that can be used, a frequent questioning among software development practitioners is: which architectural model to use? [1] The choice of API architecture can significantly impact the performance, scalability, and overall user experience of an application. The decision between Rest and GraphQL often poses a significant challenge. Also, different business area may need different performance demand regarding to apis. This project presents a comparative analysis of REST and GraphQL APIs, focusing on their performance in handling multiple concurrent users and various data request types. Using a dataset of 1,500 responses from both REST and GraphQL APIs across 15 business areas, the study explores key performance metrics such as throughput, response times, and error rates, offering insights into the scalability and reliability of each API architecture across different domains. Additionally, the study investigates how GraphQL and REST APIs can complement each other to enhance system performance and find the latest research and trends in REST and GraphQL architectures.**

***Keywords-API, REST, GraphQL, Performance Analysis, Scalability, Reliability***

## 1. Introduction

In the evolving landscape of web services, the choice between REST and GraphQL APIs significantly impacts the performance and scalability of applications. This paper investigates the comparative performances of REST and GraphQL, providing a data-driven insight to aid developers in selecting the most suitable API architecture for their needs.

In the continually evolving landscape of web services, the selection of an appropriate architectural model stands as a pivotal decision, profoundly impacting the performance, scalability, and overall user experience of an application. Among the myriad of architectural paradigms, the decision between REST and GraphQL often presents a significant challenge to software development practitioners. Moreover, the diverse demands across different business domains underscore the need for tailored performance considerations when selecting API architectures. This study endeavors to address this challenge by presenting a comprehensive comparative analysis of REST and GraphQL APIs, focusing on their performance in handling multiple concurrent users and diverse data request types. Through the examination of key performance metrics such as throughput, response times, and error rates, gleaned from a dataset encompassing 1,500 responses across 15 distinct business sectors, this research aims to provide valuable insights into the scalability and reliability of each API architecture within varying application contexts. Furthermore, the study explores the potential synergies between REST and GraphQL APIs and delves into the latest research and trends shaping their architectural landscapes. By shedding light on the nuanced performance characteristics of REST and GraphQL APIs, this study aims to equip developers with empirical evidence to inform their architectural decisions and navigate the complex terrain of modern web service development.

## 2. Background

This section provides a brief overview of web services and the architectural models REST and GraphQL. It presents the most significant characteristics of each architectural model to better understand the motivations behind our study. For a detailed presentation of GraphQL, readers are referred to its documentation [2], and for REST, we recommend the doctoral thesis that introduced this concept [3].

### A. Web Services

Web services are collections of protocols and standards used to exchange data between web systems. They enable software applications written in multiple programming languages and running on various platforms to exchange data over computer networks, such as the Internet. These services facilitate interoperability between system communications [4]. Several implementations, such as SOAP, REST, and GraphQL, provide solutions for this concept.

### B. REpresentational State Transfer (REST)

REST is an architectural style for implementing distributed systems. The style defines a set of constraints intended to improve performance, availability, and scalability, and it is based on a traditional client-server paradigm [3]–[5]. REST-based APIs are the ones that follow the constraints defined by the REST style. REST also defines a uniform interface for system components based on resource identification and dynamic data provision. In REST-based APIs, data is exposed by means of endpoints. Each endpoint returns data about one resource, and each resource has a predefined set of fields.

### C. GraphQL

In GraphQL, service data is exposed as a graph [10], defined by means of a schema. Each node of this graph/schema represents objects and contains fields. Each field has a name and a type. Edges appear when a field references another object. Clients access a GraphQL service through a single endpoint, which is used to submit queries. [7]

D. Compare REST and GraphQL

GraphQL offers several advantages over REST APIs. Firstly, it provides superior control over data fetching, enhancing performance by enabling clients to request only the necessary data. This feature reduces the payload size and improves efficiency. Additionally, GraphQL's ability to consolidate data from multiple sources in a single request reduces the number of HTTP calls required, thereby minimizing latency and enhancing scalability. Moreover, GraphQL's precise response mechanism alleviates bandwidth concerns, making it more suitable for resource-constrained environments such as small devices. Furthermore, its flexibility and rapid prototyping capabilities facilitate seamless adaptation to evolving UI requirements without necessitating extensive server-side modifications, thus accelerating development, and enhancing system agility. Overall, these attributes underscore GraphQL's efficacy in addressing modern application development challenges and its potential to drive innovation in distributed systems architecture.

While GraphQL offers compelling advantages, traditional RESTful APIs remain advantageous in certain contexts. REST's flexibility in handling complex queries by establishing multiple endpoints allows for efficient data retrieval and fine-tuning of specific queries. In contrast, GraphQL's susceptibility to performance issues with large queries demands careful optimization. Moreover, for simpler tasks or applications with predictable data usage patterns, REST may offer a more straightforward and efficient solution, as GraphQL's additional complexity, including types, queries, and resolvers, can potentially increase response times and development overhead. Therefore, in scenarios where simplicity and performance are paramount, RESTful APIs continue to be a pragmatic choice, highlighting the importance of considering the specific requirements and characteristics of each application when selecting the appropriate API architecture.

## 3. Methodology

The study employed JMeter to conduct load testing on 150 REST and 150 GraphQL endpoints across 15 different categories. Metrics such as average response time, throughput, and error rates were collected under varying load conditions to evaluate each API's performance.

This study evaluated the performance of RESTful and GraphQL APIs across various user loads and application categories. Performance metrics included throughput, response time, and data transmission rates (sent and received KB/sec). We used a dataset that captured these metrics for APIs servicing 1, 20, 50, 100, and 1000 concurrent users. Data were aggregated by API type and analyzed per application category, including Animal, Art & Design, Business, Calendar, Development, Email, Finance, Game, Movie, Music, Security, Social, Transport, and Weather.

A Random Forest regression model was trained to predict the performance metrics based on API type, user load, and application category. The importance of these features was evaluated using the model's feature_importances_ attribute (formula 2). Correlation matrices for each API type were generated to understand the relationship between different metrics.

For a Random Forest with B trees, the importance of feature $j$ is:

$$Imp_j = \frac{1}{B}\sum_{b=1}^{B} Imp_{j,b} \qquad (2)$$

- $Imp_{j,b}$ : Importance of feature $j$ in tree $b$

## 4. Results

The performance analysis indicated that RESTful APIs generally provided higher throughput and data transmission rates across most application categories (shown in Figure 1). This was particularly noticeable for high user loads, where RESTful APIs maintained superior performance compared to GraphQL. (shown in Figure 4)

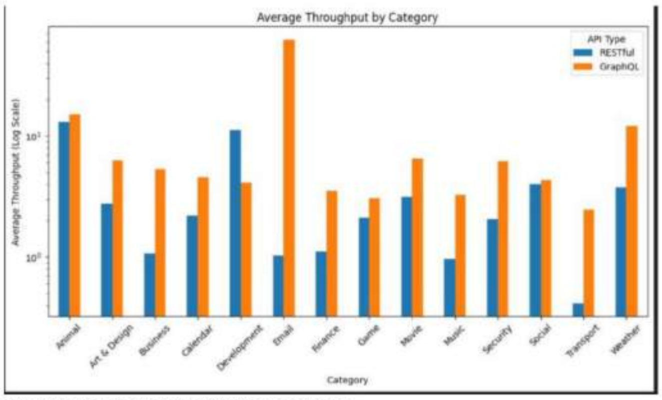


**Figure 1: Throughput Comparison**

As shown in Figure 2, GraphQL APIs demonstrated shorter response times in categories likely to require complex queries, such as Email, Development, and Social. The Random Forest model corroborated these findings, with the most important feature for predicting performance being the API type (feature importance of 0.6716), followed by the number of concurrent users (feature importance of 0.1394).

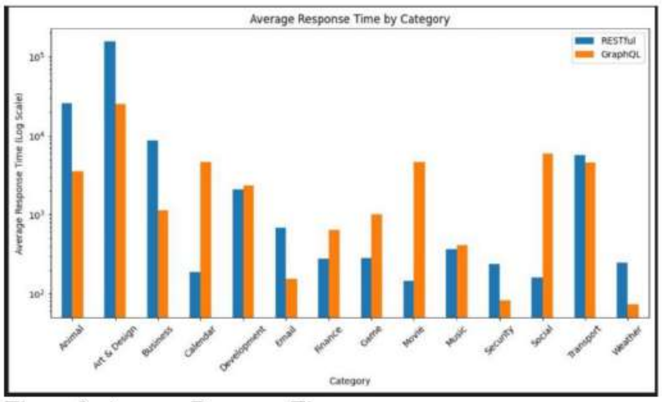


**Figure 2: Average Response Time**

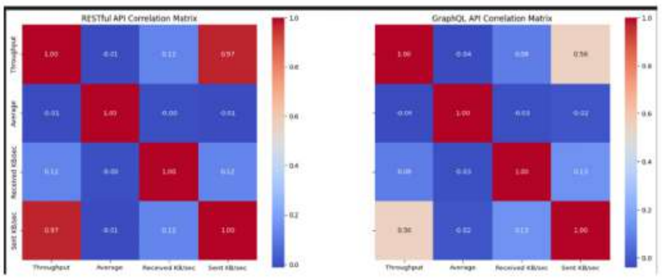


**Figure 3: Correlation graph of Throughput, Average Response Time, received KB/Sec, sent KB/Sec**

The correlation analysis for RESTful APIs (shown in Figure 3) showed a strong positive relationship between throughput and sent KB/sec (correlation coefficient of 0.97), whereas for GraphQL, the strongest correlation was between sent KB/sec and received KB/sec (correlation coefficient of 0.56). These results suggest that while RESTful APIs may excel in raw data handling, GraphQL APIs could offer advantages in scenarios demanding quick, tailored responses with less concern for data load.

In conclusion, our study demonstrates a clear distinction in the suitability of RESTful versus GraphQL APIs depending on the specific requirements of an application. RESTful APIs are preferable for scenarios that demand high throughput and data transmission, while GraphQL APIs are advantageous when the need for fast, efficient data retrieval and shorter response times is paramount.

Our results indicated that REST APIs generally offered higher throughput, whereas GraphQL APIs provided more stable response times and lower error rates.

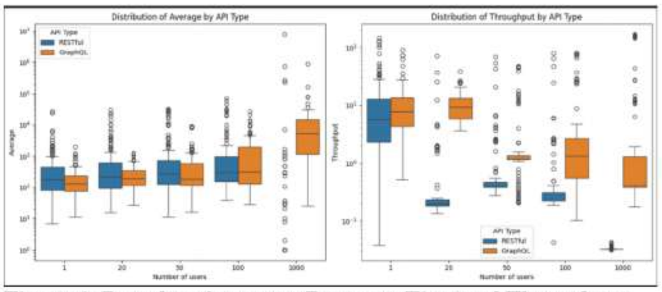
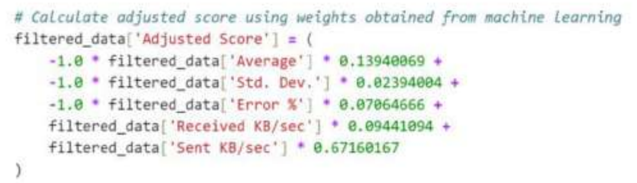


**Figure 4: Box plot of Average Response Time and Throughput**

Our Random Forest model, optimized using GridSearch, provided a quantitative measure of feature importance for predicting API performance. The feature importances, representing the relative contribution of each metric to the predictive accuracy of the model, were as follows: average response time (0.1394), standard deviation of response time (0.0239), error percentage (0.0764), received KB/sec (0.0944), and sent KB/sec (0.6716). These importance values suggest that the amount of data sent per second is the most significant predictor of throughput performance, highlighting the critical role of data transmission efficiency in API performance.

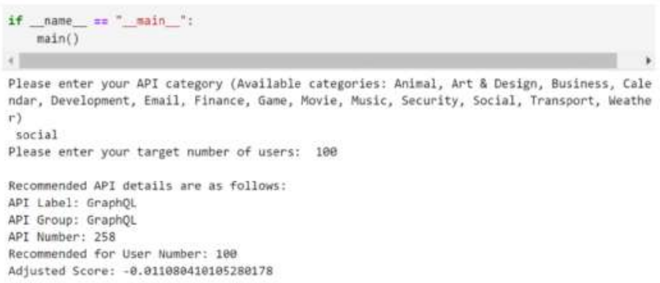
The model achieved a test mean squared error (MSE) of 0.0007211, indicating a high level of accuracy in its predictions. Utilizing the feature importances as weights, we derived an 'Adjusted Score' for each API to comprehensively evaluate performance, combining all considered metrics. The formula for the 'Adjusted Score' is as shown int Figure 5:



```
# Calculate adjusted score using weights obtained from machine learning
filtered_data['Adjusted Score'] = (
    -1.0 * filtered_data['Average'] * 0.13940069 +
    -1.0 * filtered_data['Std. Dev.'] * 0.02394004 +
    -1.0 * filtered_data['Error %'] * 0.07064666 +
    filtered_data['Received KB/sec'] * 0.09441094 +
    filtered_data['Sent KB/sec'] * 0.67160167
)
```

**Figure 5: Formula for the 'Adjusted Score'**

This score provided a single performance figure, reflecting the nuanced interplay between throughput, reliability, and data transmission rates. The negative coefficients for average and standard deviation of response time indicate that higher values negatively impact the score, emphasizing the desirability of lower response times and more consistent performance.
According to the model and function we build; we can predict and give the recommend Api for specific business area and user numbers. For instance, if there is a social media company who want to select Api and general have 100 users in the same time, we will suggest them choose the GraphQL api. (shown in Figure 6)



```
if __name__ == "__main__":
    main()
```

```
Please enter your API category (Available categories: Animal, Art & Design, Business, Cale
ndar, Development, Email, Finance, Game, Movie, Music, Security, Social, Transport, Weathe
r)
 social
Please enter your target number of users:  100

Recommended API details are as follows:
API Label: GraphQL
API Group: GraphQL
API Number: 258
Recommended for User Number: 100
Adjusted Score: -0.011080410105280178
```

**Figure 6: recommended Api for social category and 100 users**

## 5. Conclusion

This study concludes that the choice between REST and GraphQL should depend on the specific requirements of the application. For applications requiring high throughput, REST is preferable. However, for applications where efficient data retrieval and reliability are critical, GraphQL presents a more advantageous solution.

## References

[1] REST or GraphQL? A Performance Comparative Study Matheus Seabra Universidade Federal do Pará Belém, PA, Brasil matheusvieiracoelho@gmail.com Marcos Felipe Nazário Universidade Federal do Pará Instituto Evandro Chagas Belém, PA, Brasil marcosnazario@iec.gov.br Gustavo Pinto Universidade Federal do Pará Belém, PA, Brasil gpinto@ufpa.br

[2] Facebook Inc., "GraphQL specification (draft)," https://facebook.github.io/graphql/draft/, 2015, [accessed 02-April-2019].

[3] R. T. Fielding, “Architectural styles and the design of network-based software architectures,” Ph.D. dissertation, University of California, 2000.

[4] R. Mizouni, M. A. Serhani, R. Dssouli, A. Benharref, and I. Taleb, “Performance evaluation of mobile web services,” in 9th European Conference on Web Services (ECOWS), 2011, pp. 184–191.

[5] R. T. Fielding and R. N. Taylor, “Principled design of the modern Web architecture,” ACM Transactions on Internet Technology (TOIT), vol. 2, no. 2, pp. 115–150, 2002.

[6] O. Hartig and J. P´erez, “An initial analysis of Facebook’s GraphQL language,” in 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web (AMW), 2017, pp. 1–10.

[7] REST vs GraphQL: A Controlled Experiment Gleison Brito∗, Marco Tulio Valente∗∗ASERG Group, Department of Computer Science (DCC), Federal University of Minas Gerais, Brazil {gleison.brito, mtov}@dcc.ufmg.br