2021 IEEE/ACM 8th International Workshop on Software Engineering Research and Industrial Practice (SER&IP) | 978-1-6654-4476-7/20/$31.00 ©2021 IEEE | DOI: 10.1109/SER-IP52554.2021.00009

Can GraphQL Replace REST? A Study of Their Efficiency and Viability

Sri Lakshmi Vadlamani, Benjamin Emdon, Joshua Arts, and Olga Baysal
School of Computer Science
Carleton University
Ottawa, Canada
{sri.vadlamani, benjamin.emdon, joshua.arts, olga.baysal}@carleton.ca

***Abstract*—Representational State Transfer (REST) has traditionally been the standard web service architectural style for API creation. However, its popularity has been challenged with the introduction of GraphQL, an open source query language for APIs introduced by Facebook, in 2015. The latter has been quickly adopted by GitHub, Shopify, Airbnb, Twitter and more online portals are joining the list. In some instances, GraphQL has been adopted as an alternative architectural style or has been used in conjunction with REST.**

While GraphQL promises a considerable improvement over REST, much remains unexplored with respect to its efficiency and feasibility in its application. The goal of this paper is to determine viability of using GraphQL over REST for API architecture from quantitative and qualitative perspectives. A custom API client on GitHub is constructed to check on the response times and the corresponding magnitude of difference between REST and GraphQL. Thereafter, the paper surveyed employees of GitHub to understand software developers' educated opinion and perceptions about REST and GraphQL based on their practical experience with APIs. The results show that both API paradigms have their benefits and weaknesses, and one cannot replace the other, at least in the near future.

***Index Terms*—GraphQL, REST, efficiency, viability, adoption, developer perspective, survey, APIs.**

I. Introduction

The advancement of API protocols in the software industry has evolved over time. The demand for flexible and consistent API contracts has increased as more companies build public facing APIs. REST is currently the most common API protocol in the industry. REST-compliant web services make their resources available through specified HTTP endpoints. Each resource requires its own request. One of the major limitations of REST is that API users often require multiple related resources at a time, resulting in multiple round trip requests to fetch each resource. This limitation makes it difficult for third-parties to integrate with REST-compliant services efficiently, since integrator often needs to access multiple resources at a time. GraphQL, is an alternative API protocol, developed by Facebook in 2012; it has been more widely adopted, since the definition and implementation of GraphQL has been open-sourced in 2015. GraphQL offers a comprehensive API protocol which claims to address some of the issues with REST. One major advantage of GraphQL over REST is that API users are given the capability to, in a single request, request all and only the resources they need [1], [2].

While GraphQL promises a considerable improvement over REST, much remains unexplored with respect to its efficiency and feasibility in its application. This paper attempts to explore the benefits and weaknesses of both REST and GraphQL to ascertain if it is indeed possible for GraphQL to replace REST or to understand if they would have to co-exist, at least for now. More precisely, the goal of this paper is to determine the viability, or lack thereof, of using GraphQL over REST for API architecture, to ultimately make a recommendation on its use cases.

GitHub, the platform for open and closed source software collaboration, was one of the early adopters of GraphQL in 2016, while maintaining its REST capability in parallel [3]. Since then, GitHub has maintained one of the largest and most used GraphQL APIs in the industry.[1]

This paper analyzes REST and GraphQL from both quantitative and qualitative perspectives. The quantitative study focuses on comparing the efficiency of REST to GraphQL against four queries. While the qualitative study aims at identifying what software developers think about REST and GraphQL based on their practical experience with APIs. Moreover, the quantitative analysis refers to the academic evaluation, while the qualitative study is part of the industrial validation.

II. Background

A major design point of REST is that its implementation is only dependent on HTTP/1.1, i.e., any machine that supports HTTP/1.1 can effectively implement or communicate with a REST-compliant service [4], [5], [6], [7]. Moreover, REST specifies a clear separation of client and server. Servers expose resources through a URL. Clients request resources through an HTTP request. REST is a stateless protocol, meaning that the server and client are not aware of each others states. Also, REST requires that a client make the effort to determine the specifics of the resource or operation they hope to accomplish. A REST request consists of: a HTTP verb, a path to the resource, a header and a body.

[1]GitHub has granted this study access to survey GitHub employees and analyze its public APIs to further the industry's knowledge on this topic. A replication package of this survey and software developed to conduct quantitative analysis is available at https://github.com/Sri-Vadlamani/GraphQL-REST

An HTTP request always expects a response, therefore, providing an accept header in the request will tell the server the format of the response. A typical response consists of a status code, a header and a body. Some status codes are standard in HTTP/1.1, while other unused codes are usually adopted for implementation of specific use cases. REST takes full advantage of HTTP caching. HTTP caching allows resources to be copied to multiple places along the request path (e.g., local cache, proxy cache). If any of the caches along the request path are hit, it uses the copy to satisfy the request. Caching semantics are standardised in HTTP/1.1. HTTP's caching mechanism allows REST-compliant services to be highly available and performing when specified.

On the other hand, GraphQL is a schema language to describe an API, an API query language, and a server-side run-time for executing queries. Unlike REST, GraphQL is not tied to HTTP, and can exist as an API protocol over any client server transport protocol. Instead of specifying an intent with a HTTP verb, in GraphQL there are two types of interactions: *queries* and *mutations*. Queries provide a way to request data, while mutations provide a way to create, update, and destroy data. In other words, a GraphQL specification defines the schema language and the query language that provides a strongly typed hierarchical way to describe resources as types, which are connected through graph relationships [8]. At the root of the schema, there is a `schema` type, which typically has `Query` and `Mutation` types. The GraphQL's API query language is modeled after its result language JSON. A request specifies an intent to either query or mutate the data. In GraphQL, a query specifies the exact fields which it needs. Only the specified fields appear in the response, in the exact structure in which they were requested. The GraphQL's design allows clients to specify their own queries, thus letting a single API meet the needs of many different clients. The ability to request and receive only the data required allows GraphQL to avoid over-fetching and under-fetching, which is a common issue with REST. On the other hand, unlike REST, GraphQL does not have a built-in caching mechanism. Therefore, due to the nature of unpredictable dynamic queries, it is not trivial to devise a robust caching approach to GraphQL.

While GraphQL promises a considerable improvement over REST, much remains unexplored with respect to its efficiency and feasibility in its application. This work analyzes REST and GraphQL from both a quantitative and a qualitative perspectives. The quantitative aspects would focus on comparing the efficiency of REST to GraphQL in a practical setting. On the other hand, the qualitative aspects would focus on identifying what software developers think about REST and GraphQL based on their practical experience with APIs.

III. Related Work

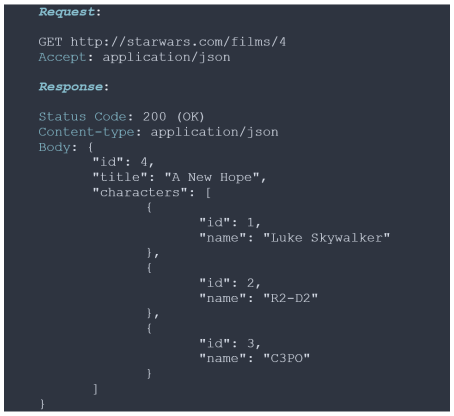
GraphQL is a relatively new in comparison with REST, therefore, there is neither a lot of literature analyzing GraphQL nor empirically comparing the gains of one web service architectural style over another. Brito et al [9] reviewed blog posts (grey literature) to understand the benefits and key characteristics of GraphQL, as perceived by practitioners. They conclude that GraphQL is more efficient in terms of reducing the size of JSON documents (in terms of number of bytes) by 99%. Hartig and Perez [10], [11] provide semantics for a formal query and thereafter analyze the language. Their results show that GraphQL is a relatively less complex language. Wittern et al. [12] assessed the feasibility of automatically generating GraphQL wrappers for existing REST(-like) APIs with Open API Specification (OAS) and based on this, they studied the gains achieved between GraphQL and REST. Furthermore, they discuss the challenges with creating such wrappers, such as data sanitation, authentication or dealing with nested queries. Vargas et al. [13] recognize that GraphQL could become a viable alternative to REST architectural style (which they believe is flawed and inefficient), however, they note that GraphQL schema implementation would need to be tested thoroughly prior to gaining more wider acceptance [14], [15]. Wittern et al. [16] also analyze specific GraphQL schemas with an aim to understand the strengths and weakness of this API in practice. They study the design of GraphQL interfaces in practice by analyzing two GraphQL schemas (one from 16 commercial GraphQL schemas and one picked from 8,399 schemas from GitHub projects). Furthermore, their work highlights the real possibility of security susceptibility of a majority of GraphQL APIs and provides certain ways to address these concerns.

Brito and Valente [9] conducted a controlled qualitative experiment, where 22 students, consisting of combination of graduate and undergraduates, were asked to implement using REST and GraphQL, eight (8) queries for accessing a web service. Their experiment shows that REST requires more effort, as compared to GraphQL, to implement remote service queries (9 minutes versus 6 minutes of median time). Moreover, their experiment has shown that the time and effort to implement a REST query increase with more complex endpoints and with several parameters. Furthermore, they show that implementing a GraphQL query is easier for participants with no previous experience with GraphQL. They have studied and contrasted the effort required and the perception of developers (students) while implementing remote queries on REST and GraphQL, and concluded that GraphQL outperforms REST.

However, the state-of-the-art research lacks studies that combine quantitative and qualitative analysis between REST and GraphQL. In this work, we try combining both quantitative analysis of the technology's performance (response time) and qualitative study conducted with the GitHub developers who are working with both API architectures.

IV. Methodology

To determine whether GraphQL was a viable API architecture over REST, the two architectures are compared using both quantitative and qualitative analyses. More specifically, this exploration culminated in three research questions (RQs) that objectively measure and compare the performance of REST and GraphQL, as well as allow us to understand their benefits



```
Request:

GET http://starwars.com/films/4
Accept: application/json

Response:

Status Code: 200 (OK)
Content-type: application/json
Body: {
        "id": 4,
        "title": "A New Hope",
        "characters": [
                {
                        "id": 1,
                        "name": "Luke Skywalker"
                },
                {
                        "id": 2,
                        "name": "R2-D2"
                },
                {
                        "id": 3,
                        "name": "C3PO"
                }
        ]
}
```

Fig. 1: A REST request and response.

and weaknesses by studying the perceptions of the GitHub experts.

A. Research Questions

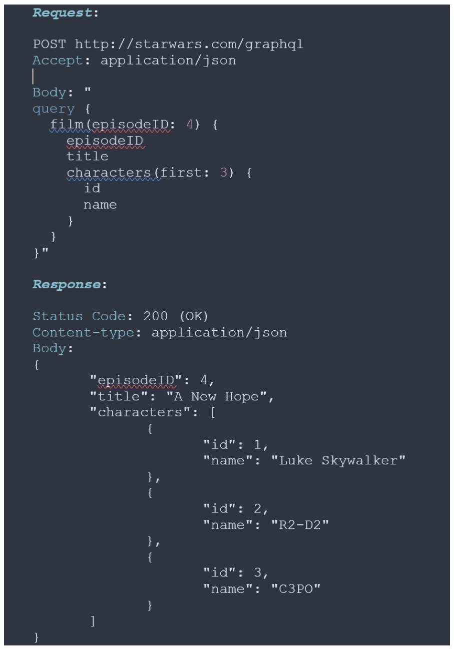
The primary research questions this paper explores are as follows:

- **RQ1: How do REST and GraphQL compare in terms of efficiency of a single request?** A single REST request can be compared to a single GraphQL request in terms of efficiency. A proxy for efficiency that is used here is the request response time.
- **RQ2: What are the benefits and weaknesses of REST and GraphQL?** REST and GraphQL are compared in terms of their benefits and weaknesses. This is done through a survey of a community of participants who have experience with both APIs.
- **RQ3: Which API is more likely to be adopted by developers?** The perspectives of experienced developers on either REST and/or GraphQL are gathered to understand which technology is likely to receive more adoption relative to its current adoption, or which technology is better for a particular reason.

B. Quantitative Analysis

REST and GraphQL can be quantitatively analyzed by directly comparing their efficiency. For this purpose, a scenario was developed where a REST API and a GraphQL API produced the same data, from the same source, in a single request. For such a scenario to be possible, a REST API would require a matching GraphQL API which provided the same resources. GitHub's public APIs [17] met this criteria and were used in this study to analyze the efficiency of REST compared to GraphQL in a single request.

For example, a REST API endpoint such as the one shown in Figure 1 was taken and was used to construct a GraphQL query, as shown in Figure 2, which produced a syntactically similar JSON response. To an API consumer, this response could be seen as an equivalent query. Both responses request the same data, and satisfy the same needs.



```
Request:

POST http://starwars.com/graphql
Accept: application/json

Body: "
query {
  film(episodeID: 4) {
    episodeID
    title
    characters(first: 3) {
      id
      name
    }
  }
}"

Response:

Status Code: 200 (OK)
Content-type: application/json
Body:
{
        "episodeID": 4,
        "title": "A New Hope",
        "characters": [
                {
                        "id": 1,
                        "name": "Luke Skywalker"
                },
                {
                        "id": 2,
                        "name": "R2-D2"
                },
                {
                        "id": 3,
                        "name": "C3PO"
                }
        ]
}
```

Fig. 2: A GraphQL request and response.

After developing a query to directly compare the efficiency of REST and GraphQL, efficiency was assessed in terms of request response time. Also, the amount of memory used by each architecture was considered, but since GraphQL requires a string query to be sent with every request, it would always use more memory in a single request scenario.

Furthermore, to benchmark requests made to GitHub's public REST and GraphQL APIs, the delta of the time right before a request was calculated, and the time right after the response was received was considered. When using request response time as a proxy for efficiency, multiple influential factors were considered: (i) time taken to transport a request, (ii) server computation and request handling, and (iii) current request load of an individual GitHub server.

To account for transport jitters and evaluate load on individual servers, repeating requests were made using a software, developed by one of the authors, to benchmark request response times. This software makes a number of repeating requests to a HTTP endpoint and has the ability to record each request's response time to a table. The source code to this software is under the MIT license, and is open sourced on GitHub [18].

Prior to analyzing the results of the performance, about 1–5% of the outliers were removed. Thereafter, in order to analyze the results, we conducted t-tests to compare the performance of APIs. The t-tests provide us with `t` and `p`-value. The idea was to compare the `p`-value to the (5%) significance level of α[2].

[2]If it is less than α, then the null hypothesis is rejected. On the other hand, if the result is greater than α, then we do not reject the the null hypothesis.

C. Qualitative Analysis

The two APIs were qualitatively analyzed by surveying 38 software developers, who were employees of GitHub Inc. [19] and had good working knowledge of both REST and GraphQL APIs. The survey included the following sections:

1) **Participant background**, documented the participants role and their level of expertise with REST and GraphQL,
2) **Benefits and weaknesses**, of the two APIs as reported by each participant
3) **Situational use cases**, a matrix type response section wherein participants reported whether REST, GraphQL, both, or neither API architecture were suitable for that specific situation,
4) **High level opinion** used Likert scale questions which asked the participants their perspective on a certain features of REST and GraphQL, and
5) **Anticipated adoption**: where each participant was asked to estimate the level of adoption of REST and GraphQL in 5 and 10 years into the future.

To determine the level of expertise each participant had with REST and GraphQL, each participant self reported their level of expertise using the scale developed by McBeath [20].

D. Survey Participants

The university's Ethics Review Board approval for the entire experiment was obtained prior to electronically distributing the survey to the GitHub's internal forum. Qualtrics[3], an online survey platform, was used to conduct the survey and data collection.

This survey study analyzes the responses received from 38 GitHub employees. We report the characteristics of the participants in our study.

1) **Main Role**: 90% of the respondents have identified themselves as engineers (34 out of 38 of the respondents).
2) **Years of Experience**: 82% have 5+ years of experience in working on web systems and/or their ecosystem (tooling, documentation, managing a team)
3) **Expertise with REST APIs**: 25 of the 38 respondents were either advanced users or highly accomplished, while 11 are intermediate users and the remaining 2 have indicated that they are apprentices.
4) **Expertise with GraphQL API**: Given that GraphQL is a relatively new, it is very much expected that the spread of expertise would be skewed towards the lower-scale; this is indeed reflected in the self-declaration by GitHub employees. Only 11 out of 38 have self-identified as advanced and/or accomplished users; 7 have noted that they are intermediate users and 11 have noted that they are apprentices; lastly, 9 out of the 38 are either beginners (5), or pursuing (3) or interested (1).

V. Results

We now present the results of the quantitative and qualitative studies.

[3]https://www.qualtrics.com/

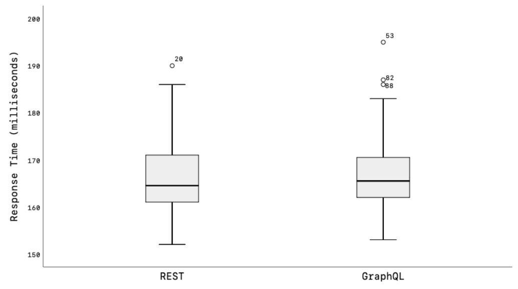


Fig. 3: Response times for `GET/user`.

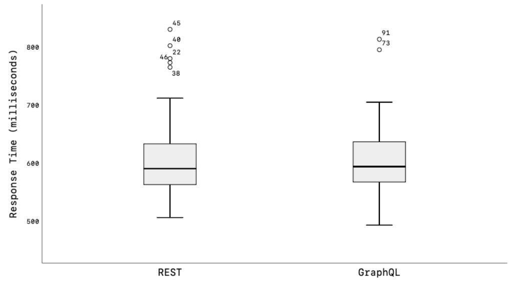


Fig. 4: Response times for `POST /repos/:owner/:repo/issues/:issue_number/comments`.

A. Quantitative Results

The REST requests and their equivalent GraphQL requests were executed, and their response times were measured and analyzed for comparison. For each request query, 50 requests were run to get their response times for each API architecture. It is noted that the *error bars* in Figure 3, Figure 4, Figure 5, and Figure 6 represent 95% confidence intervals. Table I summarizes the average response time of running the same query on GraphQL and REST APIs.

The REST endpoint `GET/user` (Query 1) and its equivalent GraphQL query yielded the response times seen in Figure 3. In essence, for the `GET/user` query, the response time was not significantly different between REST and GraphQL ($\mathtt{t}(88)=-0.482$, $\mathtt{p} = 0.631$).

The REST endpoint `POST /repos/:owner/:repo/issues/:issue_number/comments` (Query 2) and its equivalent GraphQL query yielded the response times seen in Figure 4. For this query, it is observed that the response time was not significantly different between REST and GraphQL, $\mathtt{t}(96)=0.278$, $\mathtt{p} = 0.781$.

The REST query for `GET/repos/:owner/:repo/issues/:issue_number` (Query 3) and its equivalent GraphQL query, yielded the response times as shown in Figure 5. For this query, the response times were significantly lower for GraphQL than for REST ($\mathtt{t}(94)=35.878$, $\mathtt{p} <$

TABLE I: Average response time (ms): REST & GraphQL.

Query	REST	GraphQL
Query 1	171.16	176.96
Query 2	627.00	606.34
Query 3	225.44	144.88
Query 4	338.16	388.46

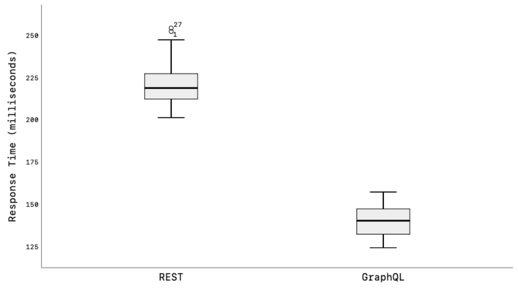


Fig. 5: Response times for `GET/repos/:owner/:repo/issues/:issue_number`.

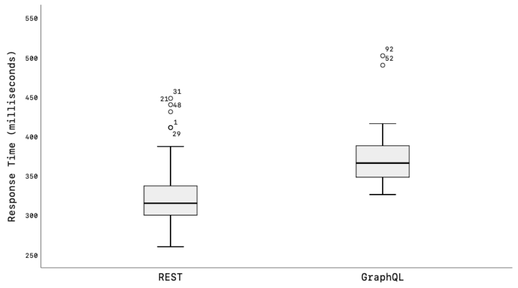


Fig. 6: Response times for `GET/repos/:owner/:repo/stargazers`.

0.001). In other words, GraphQL performed better than REST.

The REST endpoint `GET/repos/:owner/:repo/stargazers` (Query 4) and its equivalent GraphQL query, yielded the response times as reported in Figure 6. For this query, the response times were significantly lower for REST than for GraphQL ($t(91)=-5.600$, $p < 0.001$), i.e., REST performed better than GraphQL.

The quantitative analysis compared REST and GraphQL in terms of efficiency of a single request and the request response time was used as a proxy for efficiency. REST outperformed GraphQL in one scenario, GraphQL outperformed REST in another, and they performed very similarly in two scenarios. Based on the results of the quantitative analysis, there does not seems to be any statistically significant difference in performance between REST and GraphQL in terms of their efficiency.

B. Qualitative Results

Our qualitative study attempts to understand the developer perceptions about REST and GraphQL and is based on a survey with 38 GitHub employees, with expertise in REST and some level of familiarity and/or expertise with GraphQL. The survey asked the participants the perceived benefits and weakness of GraphQL and REST. In addition, the survey focused on efficiency of GraphQL in requesting resources, its maintainability and potential benefits for organizations that have REST APIs from adopting a GraphQL API. Lastly, the survey asked the participants on their expectation of the adoption of GraphQL over the next 5 years and also on their expectation for REST API.

The qualitative analysis showed that there is no clear winner between both REST and GraphQL APIs. Each of them had their benefits and weaknesses and neither can be declared as the winner. However, it was noted that GraphQL is a relatively new system and it would take sometime for it to be more widely accepted and adopted and only then it could be accurately compared with a more mature REST system.

GraphQL is obviously has some clear advantages when it comes to over-fetching of resources and can reduce the need for making round trip responses. Moreover, the survey participants noted that having a strong typed protocol language, a built-in type validation and having an inline documentation are benefits of GraphQL. Furthermore GraphiQL schema explorer tool provided by GraphQL is seen as an added advantage. Lastly, a minority of participants reported *"GraphQL can increase performance"* as a benefit. Notwithstanding the efficiency related advantages possessed by GraphQL the survey respondents not identified simplicity, ease of testing, maintainability, faster product development as major benefits of GraphQL. Interestingly, only a minority of developers thought that lack of caching mechanism was a major cause of concern or weakness of GraphQL.

On the other hand, the respondents have noted that learning GraphQL requires a major time commitment. One of the participant has noted that *"It's INCREDIBLY hard to learn. It's a pain . . . to test. People seem to think that GraphQL is intended to used to expose a multi-purpose API. At GitHub, at least, it adds a ton of unnecessary complexity, and they reinvented promises instead of upstreaming the minimum necessary batching logic into ActiveRecord. I wish GraphQL were only used to create page-specific queries for mobile."*. Another participant noted in the similar lines that GraphQL requires *"the mental overhead"*. Moreover, GraphQL is a relatively new system and knowledge about it is not widespread; one respondent noted that, it is *"difficult to migrate an existing project to use GraphQL instead of REST; hard to find devs who are already knowledgeable with it"*. This comment also notes that it is not very easy to migrate an existing project from an architectural style to GraphQL.

Furthermore, GraphQL is perceived to add complexity to a system. This is reflected in the comments of some of the survey participants. The consequences of this added complexity are reported as misusing the technology, not feeling completely confident in understanding the technology, and implementing solutions at the wrong layer of abstraction. Several responses highlight weakness of GraphQL that allude to how it adds complexity to a system.

- *"It is fundamentally confusing when applied to resources which are not actually graphs."*,
- *"It's often assumed in the talks, libraries etc, that you own the backend and client side of the GraphQL exchange. That is not always the case. In this case, you need to think about the interface from an external point of view just like REST. I don't believe enough folks do this."*

Furthermore, another major weakness of GraphQL is that it can cause N+1 problems without proving a way to solve them.

TABLE II: API architecture scenarios.

REST is Preferred	REST is NOT Preferred
• A mission-critical real-time API • A service that provides static content through an API • A small sized *public* web service project • A very small *internal* web service project	• API first product for millions of users (large size project) • Internal web service for other internal services to interact with • API for multiple different clients to use • A very large internal web service project • A medium sized *public* web service project • A medium sized *internal* web service project

Rest is preferred *means the number of people who voted for REST is greater than or equal to GraphQL and/or both REST and GraphQL, together.*
Rest is NOT preferred *means the opposite.*

This is echoed in the following response from a respondent, *"I'm not sure if it's just our app or all GraphQL but there are huge performance hits due to using GraphQL. We have many N+1's, boot time of the application is slowed drastically by GraphQL, and if GraphQL is supposed to fetch less resources than REST, our app isn't accomplishing that."*. This comments reflects on the performance issues of using such a large framework.

REST, on the other hand, enjoys the benefit of widespread knowledge, it has a mature ecosystem of libraries and tooling and given that it is in existence for quite sometime there are ample of available resources hence making it easy to learn. Also, REST uses status codes to convey a request status and since it is built upon HTTP, only a HTTP library is needed. For example, a respondent noted that the advantage of REST is that most projects already use it and hence there is no additional requirement to migrate to it (*"Most existing projects already use it and so no need to migrate."*).

The main weaknesses of REST is that REST API users might need to make multiple round trip requests and the responsibility is on the developer to ensure that REST is implemented correctly. Moreover, REST is a weakly typed and API users cannot avoid over-fetching. Interestingly, there seems to be little agreement on what REST is, for example, one respondent noted *"REST client-side tooling is haphazard, ranging from unopinionated to overly complicated/too far reaching...There is no one source-of-truth spec doc for REST. GraphQL's client-side tooling is more opinionated, resulting in more uniform usage and understanding of GraphQL. GraphQL's spec is published publicly, as a single source of truth."*. Furthermore, one participant mentioned versioning is one weakness of REST. REST APIs are difficult to deprecate because they are usually versioned-like dependencies. Unlike static dependencies which can be versioned independently of their clients, REST APIs are live, and changing behavior of a REST API breaks all clients that depend on it. For example, a respondent noted that *"The biggest downside for me with REST is the versioning."*.

Thereafter, the participants were given certain scenarios and were asked whether they prefer REST, GraphQL, both of them or neither, in each of those scenarios. Table II summarizes the scenarios where REST is preferred over either GraphQL

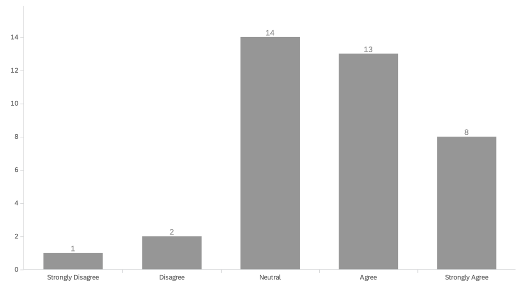


Fig. 7: GraphQL's efficiency, using a Likert scale.

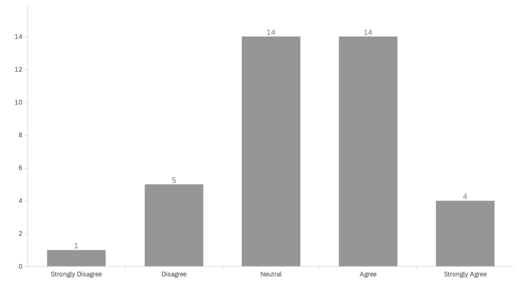


Fig. 8: Can organizations with public REST APIs benefit from adopting GraphQL?

or having both GraphQL and REST together and vice versa. It is evident from these results that REST API architectural style is preferred for small sized public and/or internal web service projects; also, REST is preferred when there is a mission critical real-time API involved. On the other hand, for medium- to large-scale projects and for projects with multiple clients developers prefer GraphQL and/or both REST and GraphQL together.

A majority (55%) of participants agreed that GraphQL APIs make it easier to efficiently request resources. On the other hand, about 36% of participants remained neutral. Figure 7 illustrates how participants reported their agreement with the statement about GraphQL's efficiency using a Likert scale.

Figure 8 shows how participants reported their agreement with a statement about whether organizations which have REST APIs could benefit from adopting a GraphQL API. 47% of participants agree with the statement that organizations with public REST APIs could benefit from adopting GraphQL. We can also observe that 36% of participants' opinion remains as neutral. Figure 9 reports the reaction of participants on the statement about whether GraphQL APIs are more maintainable than REST APIs. The survey indicated that 34% of participants disagree with the statement that GraphQL APIs are more maintainable than REST APIs; whereas, 42% of participants remained neutral on this subject.

Finally, the survey asked the respondents about how they anticipate the level of adoption of GraphQL in the next five-year horizon. The majority of respondents (57%) indicated that they believe GraphQL will receive more adoption in the next 5 years. Also, 26% reported that GraphQL would receive the same amount of adoption in the next 5 years. Whereas, about 11% of the respondents indicated that the adoption of

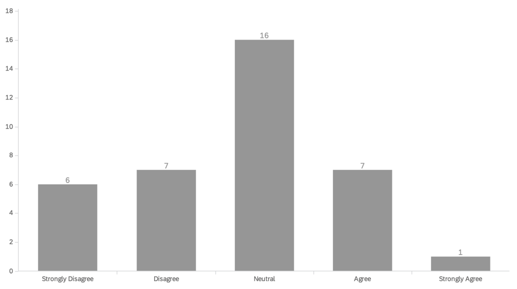


Fig. 9: Are GraphQL more maintainable than REST?

GraphQL would go down in the next five years. Similarly, the respondents were asked about their view on the adoption of REST in the next five years. It is interesting to note that about 65% of participants reported that they believe REST would continue to receive the same amount of adoption in the next 5 years; and, 21% reported that they believe that REST would receive more adoption in the next 5 years, whereas, only 8% indicated that the usage of REST APIs would go down in the next five years.

C. Answers to Research Questions

While we presented the overall results of the quantitative and qualitative studies, we now highlight and summarize the answers to our research questions.

Answer to RQ1: *How does REST and GraphQL compare in terms of efficiency of a single request?* When using request response time as a proxy for efficiency (RQ1), we found that REST outperformed GraphQL in one scenario, GraphQL outperformed REST in another scenario, and they performed very similarly in the other two scenarios. As a conclusion, in terms of efficiency of the single request, there is no clear winner between the two APIs.

Answer to RQ2: *What are the benefits and downsides of REST and GraphQL?* Table III and Table IV summarize benefits and weaknesses of REST and GraphQL, respectively. According to a majority of participants, REST is perceived to be easy to learn with lots of resources, on the other hand, most respondents found that GraphQL is difficult to learn and requires more time commitment, and that its knowledge base is limited. One explanation for this could be that GraphQL is still relatively new and its resources might not be fully developed. Notwithstanding, most respondents state that GraphQL reduces over-fetching and hence reduces the need for making round-trip requests; this is indeed the biggest pitfall with REST.

Answer to RQ3: *How do developers view REST compared with GraphQL?* Majority of participants (55%) reported that GraphQL APIs make it easier to efficiently request resources than REST APIs. 47% of participants agreed with the statement that organizations with public REST APIs could benefit from adopting GraphQL, while 36% of participants remained neutral. 34% of participants disagreed with the statement that GraphQL APIs are more maintainable than REST APIs, while 42% of participants remained neutral. A majority (57%) of the participants anticipate that GraphQL would receive more adoption over the next 5 years, while most (65%) of the participants anticipate that REST would receive about the same amount of adoption as it currently has over the next 5 years.

TABLE III: Top benefits and weakness of REST.

Top Benefits	Top Weakness
• Widespread knowledge (32) • Mature tooling ecosystem (29) • Easy to learn (27) • Uses status codes to convey request status (26) • Built on HTTP (23)	• APIs requires multiple round trip requests (34) • Onus on developer to ensure correct implementation (31) • Weakly typed (16) • Over-fetching (16) • Significant effort to document (16)

(The number in brackets represent the votes received.)

VI. Discussion

We now discuss key findings related to API efficiency, promise and pitfalls of GraphQL, as well as threats to validity.

A. APIs Performance

In the quantitative analysis of response times from a single REST and GraphQL request, one case was observed where REST outperformed GraphQL, one case was observed where GraphQL outperformed REST, and two cases were observed where they performed very similarly. These results do not conclusively demonstrate which API architecture is more efficient overall, and suggest that there are scenarios where each might be slightly preferable.

Although a sample of four request pairs is not a large enough to generalize to all GraphQL and REST APIs, it does suggest that GraphQL is sometimes more efficient than REST in a single request scenario. And, overall these results suggest that the two API architectures perform similarly in terms of efficiency in a single request.

The qualitative results indicated that GraphQL can benefit by reducing the need to make multiple round trip requests (under-fetching), and that reduces over-fetching. However, the qualitative findings are partially consistent with the quantitative findings where GraphQL performed similarly to REST in a single request. In the case, where REST would cause under fetching, GraphQL would significantly outperform multiple REST requests in terms of speed, assuming network speed is a significant factor. In the case of over-fetching, GraphQL would not necessarily outperform REST in terms of request speed, if the over-fetching is not significant.

B. GraphQL's Promise

In an over-fetching scenario GraphQL might not outperform REST, but it still performs similarly to REST in a single request comparison, and therefore would be considered a viable alternative to REST. Furthermore, GraphQL would be more efficient than REST in an under-fetching scenario. Assuming that under-fetching scenarios are common, then GraphQL would be a viable API architecture over REST.

From an application perspective, one should consider using REST or GraphQL when developing a medium to very large

TABLE IV: Top benefits and weakness of GraphQL

Top Benefits	Top Weakness
• Reduces over-fetching (27) • Strongly typed protocol language (26) • Reduced the need for making round trip requests (25) • Built-in type validation (24) • GraphiQL schema API explorer (23)	• Learning requires substantial time investment (29) • Does not have widespread knowledge base (26) • Potential to cause more problems without providing a guide to solve them (21) • Tooling ecosystem is not well developed (17) • Does note provide a way to calculate issue complexity (17)

(The number in brackets represent the votes received.)

sized public web service, or when developing a medium sized internal service. According to a majority of participant perspectives, one should consider GraphQL over REST if they want to make it easier to request resources efficiently. If under-fetching is a common occurrence in your API, then GraphQL would offer a more efficient architecture. One should consider GraphQL if they are willing to take the trade-offs of GraphQL over the trade-offs of REST.

C. GraphQL's Pitfalls

Despite these findings, participants considered GraphQL less frequently than REST in all scenarios. An explanation for this could include multiple factors unrelated to efficiency. One factor might be that knowledge about GraphQL is not yet widespread, whereas knowledge about REST is. Another factor might be that REST is a more mature API architecture with a strong ecosystem of tools and frameworks, whereas GraphQL's tooling and framework ecosystem is much newer and still maturing. These factors could be mitigated by increased adoption of GraphQL, which a majority (57%) of participants anticipate will occur over the next 5 years.

D. Threats to Validity

A limitation of this study is that we treat both REST and GraphQL APIs as black boxes and assume that they rely on the same logic. This assumption could be false in some cases, and would render our measurements irrelevant. However, it is very likely that GitHub models their REST API and GraphQL API using shared modeling, since it would require less code for them to maintain two interfaces which behave the same way. Furthermore, this study only analyzed only four REST and GraphQL query pairs. A more thorough examination of which API architecture performs better in a single request scenario, it is important to analyze a larger sample of REST and GraphQL query pairs. Another limitation of this study was that it focused on GitHub's API, which is a Ruby on Rails implementation. Other GraphQL APIs could be implemented differently and could potentially yield different results. A limitation on the qualitative study is that only a small subset of developers at GitHub were surveyed, and they do not represent a more general developer community. However, GitHub has a high level of expertise in this area, and the opinions of the GitHub experts who are familiar with both APIs should be valuable.

VII. Conclusion

This paper explores whether GraphQL is a viable alternative API architecture over REST by measuring and comparing the time performance efficiency in terms of request response; and, by assessing their benefits and weaknesses, and gathering developer perspectives on the two API architectures.

Acknowledgements

We thank GitHub and their 38 employees who participated in the study and helped in progressing research on this topic.

References

[1] L. Byron, "GraphQL: A data query language, Facebook Engineering," 2015. [Online]. Available: https://engineering.fb.com/core-data/graphql-a-data-query-language

[2] The Linux Foundation, "The Linux Foundation Announces Intent to Form New Foundation to Support GraphQL," 2018. [Online]. Available: https://www.linuxfoundation.org/press-release/2018/11/intent_to_form_graphql

[3] GitHub Engineering, "The GitHub GraphQL API," 2016. [Online]. Available: https://github.blog/2016-09-14-the-github-graphql-api

[4] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext transfer protocol – http/1.0, rfc 1945," 05 1996. [Online]. Available: https://www.rfc-editor.org/info/rfc1945

[5] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee, "Hypertext transfer protocol – http/1.1, rfc 2068," 01 1997. [Online]. Available: https://www.rfc-editor.org/info/rfc2068

[6] R. T. Fielding, "Architectural styles and the design of network-based software architectures. order no. 9980887, university of california, irvine," 2000. [Online]. Available: https://amturing.acm.org/award_winners/berners-lee_8087960.cfm

[7] T. Berners-Lee, "Sir tim berners-lee - a.m. turing award laureate, am turing," 2016. [Online]. Available: https://amturing.acm.org/award_winners/berners-lee_8087960.cfm

[8] GraphQL, "GraphQL Specification: June 2018 Edition," 2018. [Online]. Available: https://graphql.github.io/graphql-spec/June2018

[9] G. Brito and M. T. Valente, "Rest vs GraphQL: A Controlled Experiment," *arXiv preprint arXiv:2003.04761v*, 03 2020.

[10] O. Hartig and J. Perez, "An initial analysis of facebook's graphql language," *11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web (AMW)*, pp. 1–10, 2017.

[11] ——, "Semantics and complexity of graphql," *27th World Wide Web Conference on World Wide Web (WWW)*, pp. 1155–1164, 2018.

[12] E. Wittern, A. Cha, and J. Laredo, "Generating graphql-wrappers for rest (-like) apis," *International Conference on Web Engineering*, pp. 65–83, 2018.

[13] D. M. Vargas, A. F. Blanco, A. C. Vidaurre, J. P. S. Alcocer, M. M. Torees, and A. B. ane S. Ducasse, "Deviation testing: A test case generation technique for graphql apis," *11th International Workshop on Smalltalk Technologies (IWST)*, pp. 1–9, 2018.

[14] Yelp. Graphql intro - yelp. [Online]. Available: https://www.yelp.com/developers/graphql/guides/intro

[15] Apollo GraphQL. Apollo GraphQL, The Apollo Data Graph Platform. [Online]. Available: https://www.apollographql.com/

[16] E. Wittern, A. Cha, J. Davis, G. Baudart, and L. Mandel, "An empirical study of graphql schemas," *arXiv preprint arXiv:1907.13012*, 2019.

[17] GitHub. GitHub Public APIs. [Online]. Available: https://github.com/public-apis/public-apis

[18] J. Arts. A web-based api client. [Online]. Available: https://github.com/joshua-arts/web-api-client

[19] Wikipedia contributors, "GitHub — Wikipedia, The Free Encyclopedia," 2020, [Online; accessed 16-October-2020]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=GitHub&oldid=985080121

[20] J. McBeath, "Levels of expertise," 2018. [Online]. Available: http://jim-mcbeath.blogspot.com/2011/12/levels-of-expertise.html