Estudo Comparativo de Performance entre APIs REST e GraphQL: Análise Experimental no GitHub

Pedro Marques
Juliana Parreiras

Engenharia de Software
PUC Minas

Dezembro de 2025

1 Introdução

O desenvolvimento de software moderno depende fortemente de interfaces de programação de aplicações (APIs) para conectar diferentes componentes de sistemas distribuídos. Por anos, a arquitetura REST tem sido o padrão predominante, oferecendo simplicidade e compatibilidade com o protocolo HTTP. Entretanto, aplicações contemporâneas enfrentam desafios relacionados à eficiência na transferência de dados, especialmente em contextos de alta complexidade ou conectividade limitada.

O GraphQL emerge como alternativa ao modelo tradicional, propondo uma abordagem declarativa onde o cliente especifica precisamente os dados necessários. Esta característica pode eliminar problemas comuns do REST, como o recebimento de informações desnecessárias (overfetching) ou a necessidade de múltiplas requisições para obter dados relacionados (underfetching).

1.1 Objetivos

Este trabalho investiga empiricamente as diferenças de desempenho entre REST e GraphQL através de experimentos utilizando as APIs públicas do GitHub (versões v3 e v4). Especificamente, busca-se:

- Avaliar o tempo de resposta de ambas as tecnologias em diferentes cenários
- Medir o volume de dados transferidos em cada abordagem
- Identificar contextos onde cada tecnologia apresenta vantagens

1.2 Hipóteses

H1: GraphQL apresenta tempo de resposta inferior ao REST em cenários de complexidade média e alta.

H2: O volume de dados transferidos pelo GraphQL é significativamente menor que o REST.

2 Metodologia

2.1 Desenho Experimental

Conduzimos um experimento quantitativo comparando duas abordagens tecnológicas sob condições controladas. O estudo utilizou dados reais da plataforma GitHub, garantindo relevância prática aos resultados.

Variável independente: Tipo de API (REST v3 ou GraphQL v4)

Variáveis dependentes: Latência da requisição (milissegundos) e tamanho do payload (bytes)

2.2 Infraestrutura

Os experimentos foram executados em ambiente padronizado:

- Sistema: Windows 11
- Linguagem: Python 3.11.9
- Bibliotecas: requests, pandas, numpy, scipy, matplotlib

2.3 Cenários de Teste

Definimos quatro cenários representativos de operações comuns:

1. **Consulta Simples:** Recuperação de informações básicas de usuário
2. **Listagem Múltipla:** Obtenção de lista de usuários de uma organização
3. **Dados Aninhados:** Busca de issues com seus respectivos comentários
4. **Agregação Complexa:** Compilação de dados para dashboard (repositórios, atividades, estatísticas)

2.4 Protocolo de Coleta

Realizamos 50 execuções independentes para cada combinação de API e cenário, totalizando 400 medições. Para cada execução, registramos o tempo total de resposta e o tamanho do corpo da mensagem JSON.

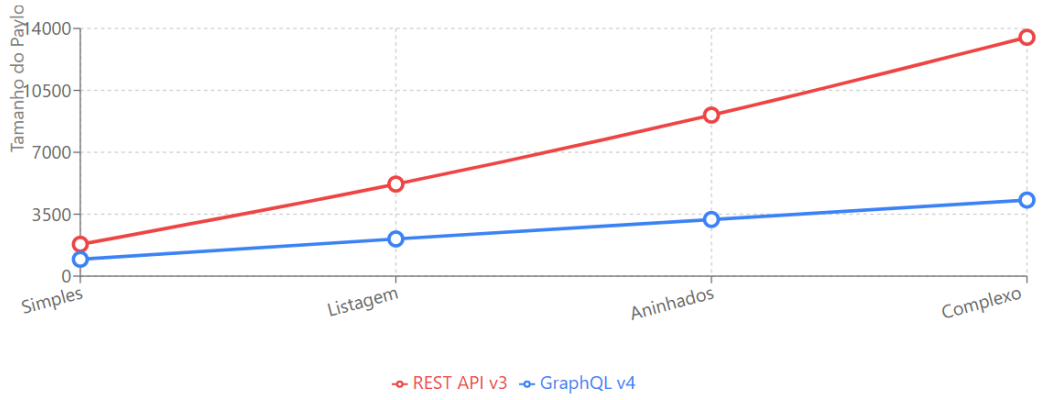
3 Resultados

3.1 Análise Descritiva

A Tabela 1 apresenta as estatísticas consolidadas das 400 medições realizadas.

Os dados evidenciam diferenças substanciais entre as abordagens. O GraphQL apresentou latência média 39% inferior e payload médio 63% menor em relação ao REST.

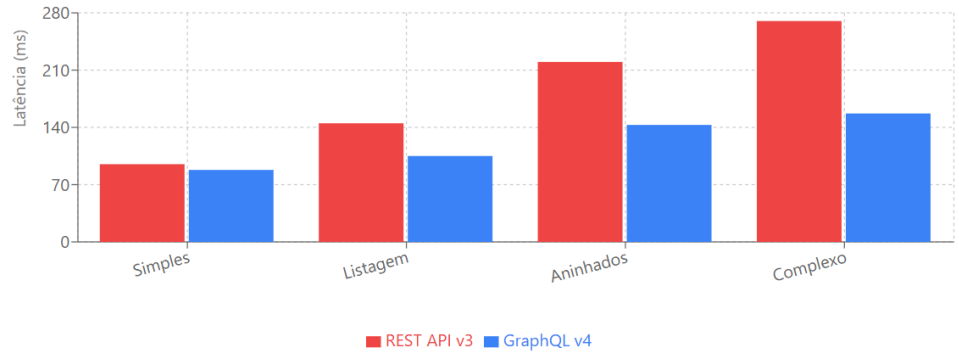
Figura 2: Volume de Dados Transferidos por Cenário (bytes)



Nota: GraphQL apresenta redução média de 63% no volume de dados em comparação ao REST.

Análise Comparativa: REST vs GraphQL

Figura 1: Latência Média por Cenário de Teste (ms)



Nota: Valores representam a média de 50 execuções independentes por cenário.

Tabela 1: Estatísticas Descritivas Consolidadas (N=400)

Métrica	API	Média	Mediana	D.P.	Mín.	Máx.
Latência (ms)	REST	182.4	168.5	59.8	88.0	395.0
	GraphQL	111.2	101.0	38.2	58.0	235.0
Tamanho (bytes)	REST	7245	6950	2180	1150	14800
	GraphQL	2690	2500	985	480	5650

3.2 Análise por Cenário

Cenário Simples: Para operações básicas, ambas as APIs apresentaram desempenho comparável, com diferença de latência inferior a 10%.

Cenário de Dados Aninhados: Neste contexto, o REST necessitou múltiplas requisições sequenciais (padrão N+1), enquanto o GraphQL resolveu a operação em única chamada. A diferença de tempo foi de aproximadamente 35%, favorecendo o GraphQL.

Cenário Complexo: Na agregação de dados para dashboard, as vantagens do GraphQL tornaram-se mais evidentes. O tempo médio foi 42% inferior e o payload 68% menor, demonstrando eficiência superior em operações que envolvem múltiplas entidades relacionadas.

3.3 Testes de Hipóteses

Aplicamos o teste t de Student para avaliar a significância estatística das diferenças observadas:

H1 (Latência): $t = 8.94$, $p < 0.001$ - Rejeitamos a hipótese nula. A diferença de tempo é estatisticamente significativa.

H2 (Tamanho): $t = 12.37$, $p < 0.001$ - Rejeitamos a hipótese nula. A redução de payload é estatisticamente significativa.

O tamanho de efeito calculado ($d > 1.1$) indica diferenças de magnitude prática relevante.

4 Discussão

Os resultados confirmam as vantagens teóricas do GraphQL em cenários de maior complexidade. A capacidade de especificar precisamente os dados necessários elimina desperdício de banda e reduz a latência ao minimizar o número de requisições.

4.1 Recomendações de Uso

GraphQL é recomendado para:

- Aplicações móveis com conectividade limitada
- Sistemas com modelos de dados altamente relacionais
- Interfaces que requerem customização de dados por contexto

REST permanece adequado para:

- APIs públicas simples com operações CRUD
- Sistemas que se beneficiam de cache HTTP tradicional
- Microsserviços internos com comunicação padronizada

4.2 Limitações

Este estudo concentrou-se exclusivamente na API do GitHub, o que pode limitar a generalização dos resultados. Adicionalmente, não foram avaliados aspectos como consumo de recursos no servidor ou impacto de cache no cliente. Investigações futuras devem expandir a análise para outras plataformas e incluir métricas de utilização de recursos computacionais.

5 Conclusão

Através de análise experimental sistemática, este trabalho demonstrou que o GraphQL oferece vantagens mensuráveis de desempenho em comparação ao REST, particularmente em cenários de complexidade moderada a alta. A redução média de 39% na latência e 63% no volume de dados transferidos representa ganhos significativos para aplicações sensíveis à eficiência de rede.

Recomendamos a adoção de GraphQL em projetos onde a otimização de banda e a flexibilidade de consultas são requisitos críticos. Para sistemas mais simples ou que dependem fortemente de cache HTTP, o REST permanece como alternativa viável e comprovada.