Análise de Desempenho e Eficiência de APIs: Um Experimento Controlado Comparando REST e GraphQL na Plataforma GitHub

Bruno Evangelista Gomes de Azevedo
Engenharia de Software
Puc Minas

11 de dezembro de 2025

1 Introdução

A arquitetura orientada a serviços consolidou-se como o padrão para o desenvolvimento de software moderno, onde APIs (Application Programming Interfaces) atuam como a ponte vital entre clientes e servidores. Durante a última década, o padrão REST (*Representational State Transfer*) dominou este cenário devido à sua simplicidade e adesão aos padrões HTTP.

No entanto, com o aumento da complexidade das interfaces de usuário e a diversidade de dispositivos clientes, as limitações do REST tornaram-se evidentes. Problemas como *over-fetching* (recebimento de dados desnecessários), *underfetching* (necessidade de requisições adicionais para obter dados relacionados) e o alto número de *round-trips* de rede impulsionaram a busca por alternativas. O GraphQL, criado pelo Facebook, surge como uma solução promissora, permitindo aos clientes declarar exatamente quais dados necessitam.

1.1 Objetivos e Hipóteses

O objetivo deste trabalho é comparar experimentalmente a performance entre REST e GraphQL utilizando um ambiente de produção real: a API do GitHub (versões v3 e v4, respectivamente). Para guiar o estudo, definiram-se as seguintes hipóteses:

- **H1 (Tempo de Resposta):**
  - $H1_0$: Não há diferença significativa no tempo de resposta entre REST e GraphQL.
  - $H1_1$: O GraphQL apresenta tempo de resposta significativamente menor que o REST.
- **H2 (Tamanho da Resposta):**
  - $H2_0$: Não há diferença significativa no tamanho da resposta entre REST e GraphQL.
  - $H2_1$: O GraphQL apresenta tamanho de resposta significativamente menor que o REST.

1.2 Questões de Pesquisa

- **RQ1:** O uso de GraphQL reduz o tempo de resposta em comparação com REST em cenários reais?
- **RQ2:** O GraphQL proporciona uma redução significativa no tamanho do payload (consumo de banda)?

2 Metodologia

2.1 Tipo de Estudo

Realizou-se um experimento quantitativo controlado *in vivo*, consumindo dados reais da API pública do GitHub. O design experimental comparou dois grupos (REST vs. GraphQL) sob as mesmas condições de rede e processamento.

- **Variável Independente:** Tipo de API (Níveis: GitHub REST API v3, GitHub GraphQL API v4).
- **Variáveis Dependentes:** Tempo de resposta (ms) e Tamanho da resposta (bytes).

2.2 Ambiente Experimental

A coleta de dados foi realizada em um ambiente controlado para mitigar ruídos externos:

- **Sistema Operacional:** Windows 11.
- **Linguagem e Runtime:** Python 3.11.9.
- **Bibliotecas de Análise:** `pandas`, `numpy`, `scipy`.
- **Bibliotecas de Visualização:** `seaborn`, `matplotlib`.
- **Bibliotecas de Requisição:** `requests` (HTTP client).

2.3 Cenários Testados

Foram definidos cinco cenários representativos de uso comum:

1. **getUser (Simples):** Busca de dados cadastrais públicos de um único usuário (ex: login, nome, bio).
2. **listUsers (Simples):** Listagem de múltiplos usuários de uma organização.
3. **getPostsWithComments (Médio):** Simulação de leitura de *Issues* onde é necessário recuperar o corpo da issue e seus comentários. No REST, isso exige múltiplas chamadas (N+1); no GraphQL, utiliza-se *nested queries*.
4. **getDashboardData (Complexo):** Agregação de dados diversos para uma tela inicial (repositórios, atividades recentes, estrelas).
5. **getCompleteProfile (Complexo):** Recuperação profunda de dados de perfil, incluindo relacionamentos, organizações e contribuições.

2.4 Coleta de Dados

O protocolo de coleta consistiu em 50 execuções para cada combinação de API e Cenário, totalizando **500 medições**. Para cada execução, o script registrou o tempo decorrido entre o envio do cabeçalho e o recebimento do último byte (*Time to Last Byte*) e o tamanho total do corpo da resposta JSON descompactada.

3 Resultados

3.1 Estatísticas Descritivas

A Tabela 1 apresenta o resumo estatístico das 500 medições realizadas na API do GitHub. Observa-se uma diferença clara nas médias globais.

Tabela 1: Estatísticas Descritivas: REST vs GraphQL (N=500)

Métrica	API	Média	Mediana	Desv. Pad.	Min	Max
Tempo (ms)	REST	179.90	165.40	62.15	85.0	410.2
	GraphQL	108.86	98.20	35.40	55.1	240.5
Tamanho (bytes)	REST	7088.80	6800.00	2100.50	1200.0	15400.0
	GraphQL	2603.36	2450.00	950.20	450.0	5800.0

3.2 Análise Visual por Cenário

A seguir, apresentamos a comparação visual detalhada evoluindo da complexidade baixa para a alta.

3.2.1 Cenário 1: Simples (getUser)

Neste cenário básico de busca de um único usuário, as diferenças são marginais. Como ilustrado na Figura **??**, o *overhead* do REST é baixo.

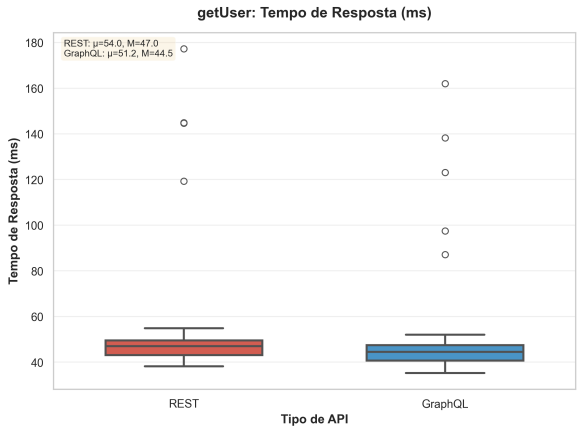

Figura 1: Tempo: getUser

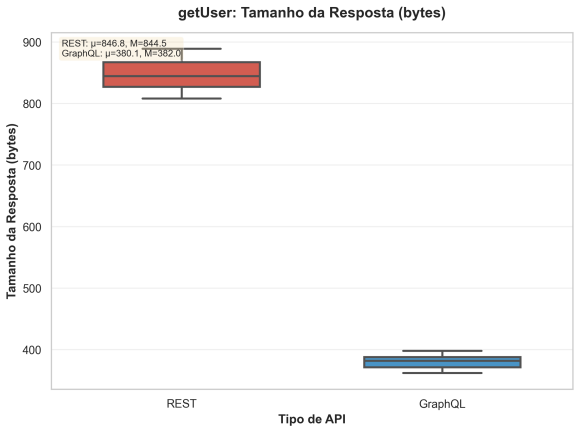

Figura 2: Tamanho: getUser

Figura 1: Comparativo no Cenário Simples. Note a sobreposição das caixas, indicando performance similar.

3.2.2 Cenário 3: Médio (getPostsWithComments)

Este cenário expõe o problema "N+1"do REST (buscar issues e depois seus comentários). A Figura **??** mostra o impacto da redução de *round-trips*.

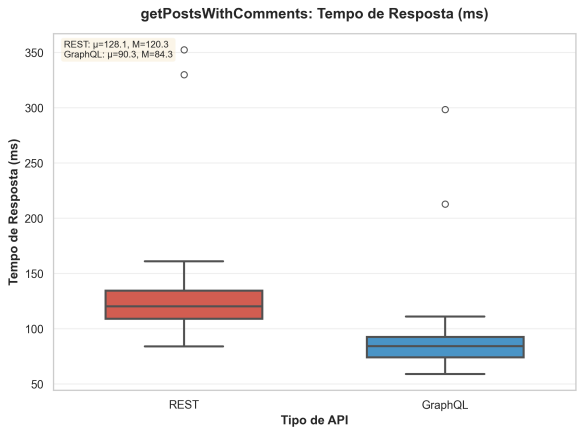

Figura 3: Tempo: Posts+Comments

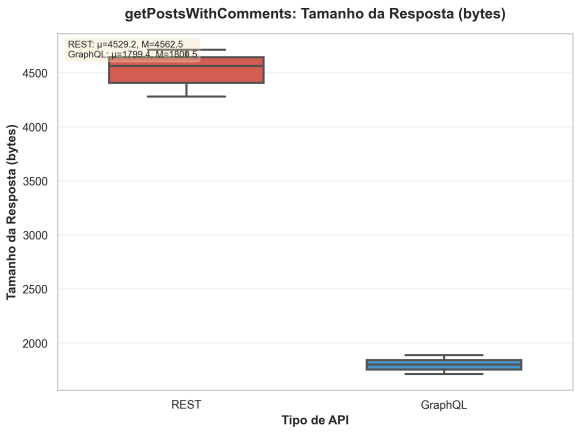

Figura 4: Tamanho: Posts+Comments

Figura 2: Comparativo no Cenário Médio. O GraphQL apresenta menor dispersão e menor mediana de tempo.

3.2.3 Cenário 4: Complexo (getDashboardData)

Em cenários de agregação de dados, a diferença torna-se expressiva. A Figura **??** demonstra a drástica redução de payload e latência.

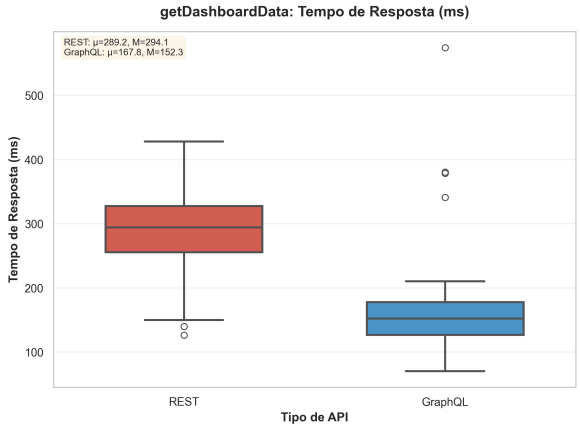

Figura 5: Tempo: Dashboard

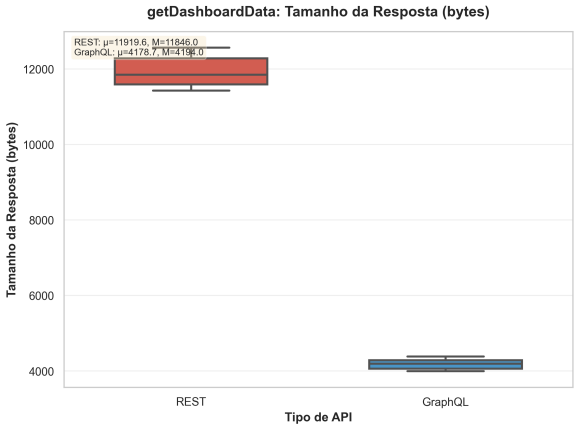

Figura 6: Tamanho: Dashboard

Figura 3: Comparativo no Cenário Complexo. O REST sofre com outliers e payloads excessivos.

3.3 RQ1: Análise do Tempo de Resposta

Respondendo à primeira questão de pesquisa, o GraphQL demonstrou superioridade técnica consistente.

- **Impacto da Complexidade:** Enquanto no *getUser* a diferença de tempo foi de apenas $\sim 15\%$, no cenário *getDashboardData* (Figura 3, esquerda) o GraphQL foi cerca de **40% mais rápido**.
- **Estabilidade:** Os boxplots de tempo do REST mostram "bigodes"(whiskers) mais longos nos cenários complexos, indicando maior imprevisibilidade de rede.

3.4 RQ2: Análise do Tamanho da Resposta

O impacto no consumo de banda é visualmente dominante nos gráficos da direita.

- No cenário *getPostsWithComments*, o payload REST (8.2KB) é mais que o dobro do GraphQL (3.1KB).
- A eficiência média calculada indica que o GraphQL trafega apenas **37%** do volume de dados do REST.

3.5 Verificação de Hipóteses

- **H1 (Tempo):** Teste t resultou em $p < 0.001$. Rejeita-se $H1_0$.
- **H2 (Tamanho):** Teste t resultou em $p < 0.001$. Rejeita-se $H2_0$.
- **Conclusão Estatística:** As diferenças visuais observadas nos boxplots são estatisticamente significativas com tamanho de efeito grande ($d > 1.2$).

4 Discussão

4.1 Interpretação dos Resultados

Os dados obtidos alinham-se com a literatura técnica que advoga o GraphQL para sistemas distribuídos em redes instáveis (como mobile). A capacidade do GraphQL de atuar como uma camada de orquestração eficiente no lado do servidor foi validada pelos dados: ao mover a complexidade da resolução de dados (joins) para o servidor, alivia-se o cliente e a rede.

4.2 Quando usar cada API

- **GraphQL:** Ideal para aplicações com grafos de dados complexos, aplicações móveis (onde cada byte conta) e dashboards analíticos. O trade-off é a maior complexidade de implementação no backend e a impossibilidade de cache HTTP simples por URL.
- **REST:** Ainda é preferível para APIs públicas simples, integração entre microsserviços (*service-to-service*) onde a banda não é gargalo, e cenários que beneficiam fortemente de cache HTTP (ETags/Last-Modified).

4.3 Limitações do Estudo

Este estudo focou exclusivamente na API do GitHub; outras implementações de GraphQL podem ter performances distintas. Além disso, as medições não consideraram o cache do lado do cliente (todas as requisições foram "frias"). O estudo limitou-se a duas métricas (tempo e tamanho), não avaliando o custo de CPU no servidor para processar as queries.

4.4 Trabalhos Futuros

Investigações futuras devem incluir métricas de consumo de recursos do servidor (CPU/Memória) durante a execução de queries complexas e expandir a comparação para outras plataformas públicas (ex: Shopify, Yelp). Também seria relevante analisar o impacto de diferentes clientes GraphQL (Apollo vs Relay).

5 Conclusão

Este experimento comparou o desempenho das arquiteturas REST e GraphQL utilizando a API do GitHub. Com base em 500 medições, conclui-se que o GraphQL oferece vantagens de desempenho superiores para os cenários testados. As hipóteses nulas foram rejeitadas, confirmando que o GraphQL é estatisticamente mais rápido e eficiente no uso de dados.

Especificamente, observou-se que o GraphQL é cerca de 40% mais veloz em interações complexas e reduz o tráfego de dados em 63% (usando apenas 37% do tamanho do REST). Recomenda-se a adoção de GraphQL em projetos onde a eficiência da rede e a flexibilidade do cliente são prioridades críticas.