Relatório Final: Análise Comparativa de Desempenho entre REST e GraphQL na API do GitHub

1. Introdução

O desenvolvimento de aplicações modernas exige interfaces de programação de aplicações (APIs) eficientes e flexíveis. Atualmente, duas abordagens predominam no mercado: **REST (Representational State Transfer)** e **GraphQL**. Enquanto o REST é o padrão da indústria há anos, o GraphQL surgiu como uma alternativa que promete resolver problemas comuns do REST, como o *overfetching* (retorno de dados desnecessários) e o *underfetching* (necessidade de múltiplas requisições para obter dados relacionados).

Este experimento tem como objetivo comparar o desempenho dessas duas tecnologias utilizando a API pública do GitHub. O estudo foca em duas métricas principais: tempo de resposta (latência) e tamanho da resposta (payload).

Para guiar este estudo, foram levantadas as seguintes questões de pesquisa e hipóteses estatísticas:

- **RQ1: Respostas às consultas GraphQL são mais rápidas que respostas às consultas REST?**
    - **H0 (Hipótese Nula):** O tempo de resposta do GraphQL é maior ou igual ao do REST.
    - **H1 (Hipótese Alternativa):** O tempo de resposta do GraphQL é menor que o do REST.
- **RQ2: Respostas às consultas GraphQL têm tamanho menor que respostas às consultas REST?**
    - **H0 (Hipótese Nula):** O tamanho da resposta do GraphQL é maior ou igual ao do REST.
    - **H1 (Hipótese Alternativa):** O tamanho da resposta do GraphQL é menor que o do REST.

2. Metodologia

Para validar as hipóteses, foi executado um experimento controlado seguindo um **desenho fatorial 2x2**.

2.1 Variáveis e Desenho Experimental

- **Variáveis Independentes (Fatores):**
    1. **Tipo de API:** REST vs. GraphQL.
    2. **Complexidade da Consulta:**
        - *Simples:* Consulta de dados básicos de um usuário (login, nome, bio).
        - *Complexa:* Consulta aninhada envolvendo usuário, repositórios e issues.
- **Variáveis Dependentes (Métricas):**

1. **Tempo de Resposta (time_ms):** Medido em milissegundos, contabilizando o tempo entre o envio da requisição e o recebimento completo da resposta.
2. **Tamanho da Resposta (size_bytes):** Medido em bytes, representando o tamanho do payload JSON retornado.

2.2 Procedimento e Ambiente

O experimento foi automatizado utilizando um script em **Python**, fazendo uso da biblioteca requests para chamadas HTTP. O ambiente de execução foi configurado localmente, realizando chamadas reais contra a API de produção do GitHub.

- **Coleta de Dados:** Foram realizadas 100 repetições (*trials*) para cada combinação de tratamento (REST/Simples, REST/Complexa, GraphQL/Simples, GraphQL/Complexa), totalizando 400 observações.
- **Controle:** Para mitigar o efeito de *caching* e latência de rede esporádica, as execuções foram intercaladas e utilizou-se o mesmo token de autenticação e conexão de rede para todas as requisições.

2.3 Definição das Consultas

Para garantir a comparabilidade, as consultas foram desenhadas para buscar a **mesma informação semântica**:

1. **Cenário Simples:** Dados do perfil do usuário "mojombo".
2. **Cenário Complexo:** Dados do perfil + lista de repositórios + últimas issues de cada repositório (no caso do REST, isso simula o *overfetching* natural do endpoint ou a necessidade de chamadas compostas; no GraphQL, solicitou-se apenas os campos específicos).

3. Resultados

Os dados brutos coletados foram processados para extrair estatísticas descritivas (Média e Desvio Padrão). Abaixo apresentamos os resultados calculados a partir das 400 amostras coletadas.

3.1 RQ1: Análise do Tempo de Resposta (Speed)

A tabela abaixo resume os tempos de resposta obtidos (em milissegundos):

Complexidade	**API Type**	**Média (ms)**	**Desvio Padrão (ms)**
Simples	REST	363.30	50.37
	GraphQL	390.96	60.15

Complexa	REST	486.06	57.65
	GraphQL	448.26	56.66

Análise: Observa-se que, para consultas simples, o REST foi ligeiramente mais rápido (média de 363ms contra 390ms do GraphQL), provavelmente devido ao menor overhead de processamento no servidor. No entanto, no cenário complexo, o GraphQL superou o REST (448ms contra 486ms), validando parcialmente a hipótese de que o GraphQL é mais eficiente para recuperar dados aninhados.

3.2 RQ2: Análise do Tamanho da Resposta (Payload Size)

A tabela abaixo resume os tamanhos das respostas obtidas (em bytes):

Complexidade	API Type	Média (Bytes)	Redução (%)
Simples	REST	1354.0	-
	GraphQL	88.0	93.5%
Complexa	REST	56053.0	-
	GraphQL	2582.0	95.4%

Análise: Os resultados confirmam drasticamente a eficiência do GraphQL em relação ao tamanho dos dados trafegados. Em consultas complexas, o GraphQL obteve uma redução de aproximadamente 95% no tamanho da resposta, eliminando completamente o problema de *overfetching* observado no REST, que retornou mais de 56KB de dados contra apenas 2.5KB do GraphQL para a mesma informação útil.

4. Discussão Final

A análise dos resultados permite validar as hipóteses levantadas e compreender os *trade-offs* entre as duas tecnologias.

Sobre o Tempo (RQ1):

Os dados indicam que o GraphQL nem sempre é mais rápido. Na consulta simples, o REST venceu. Isso sugere que o custo computacional de parsing e validação da query GraphQL pode não compensar em buscas triviais. Contudo, em cenários complexos, a capacidade do GraphQL de buscar dados relacionados de forma eficiente resultou em menor latência média.

Sobre o Tamanho (RQ2):

A hipótese de que o GraphQL reduz o tamanho da resposta foi fortemente confirmada. A diferença é de ordens de magnitude. O REST retorna o objeto completo do banco de dados, enquanto o GraphQL retorna apenas as strings e inteiros solicitados. Para dispositivos móveis ou redes com limite de dados, o GraphQL mostra-se indispensável.

Conclusão:

O experimento comprova que, para a API do GitHub, o GraphQL é vastamente superior em eficiência de transferência de dados (tamanho). Em termos de velocidade, a vantagem do GraphQL aparece conforme a complexidade da consulta aumenta. Recomenda-se o uso de GraphQL para aplicações complexas que exigem grafos de dados profundos e otimização de banda.