PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Instituto de Ciências Exatas e de Informática
Curso de Engenharia de Software

GraphQL vs REST - Um experimento controlado

Etapa 1: Desenho do Experimento

Luiz Filipe Nery Costa
Wilken Henrique Moreira

Belo Horizonte
2025

1 Desenho do Experimento

Este laboratório tem como objetivo realizar um experimento controlado para avaliar quantitativamente os benefícios da adoção de uma API GraphQL em comparação à arquitetura REST tradicional. O estudo foca na análise de duas métricas principais: tempo de resposta (latência) e eficiência na transferência de dados (tamanho do payload).

A seguir, são detalhados os componentes do desenho experimental conforme as diretrizes da engenharia de software experimental.

1.1 Hipóteses Nula e Alternativa

Para responder às Questões de Pesquisa (RQ1 e RQ2), definimos formalmente as seguintes hipóteses:

RQ1: Tempo de Resposta

- $H_{0_{tempo}}$: Não há diferença significativa no tempo de resposta entre consultas GraphQL e REST ($\mu_{GraphQL} = \mu_{REST}$).
- $H_{1_{tempo}}$: As respostas às consultas GraphQL são estatisticamente mais rápidas que as respostas REST ($\mu_{GraphQL} < \mu_{REST}$), devido à redução de múltiplos *round-trips*.

RQ2: Tamanho da Resposta

- $H_{0_{tamanho}}$: Não há diferença significativa no tamanho da resposta entre consultas GraphQL e REST ($\mu_{GraphQL} = \mu_{REST}$).
- $H_{1_{tamanho}}$: As respostas às consultas GraphQL possuem tamanho menor que as respostas REST ($\mu_{GraphQL} < \mu_{REST}$), devido à eliminação do *overfetching*.

1.2 Variáveis

1.2.1 Variáveis Dependentes

São as métricas observadas para medir o efeito dos tratamentos:

1. **Tempo de Resposta (Time):** Medido em milissegundos (ms). Refere-se ao tempo total decorrido entre o envio da primeira requisição e o recebimento completo dos dados.
2. **Tamanho da Resposta (Size):** Medido em Kilobytes (KB). Refere-se ao tamanho total do corpo da(s) resposta(s) JSON trafegada(s) pela rede.

1.2.2 Variáveis Independentes

A variável independente manipulada é a **Tecnologia da API**. Esta variável é do tipo **nominal** (categórica), pois define a arquitetura utilizada para acessar os dados, sem implicar uma ordem numérica natural.

1.3 Tratamentos

A variável independente possui dois níveis, que constituem os tratamentos aplicados:

- **Tratamento A (REST):** Obtenção de dados através de múltiplos endpoints HTTP (v3 da API), seguindo a estrutura de recursos padrão.
- **Tratamento B (GraphQL):** Obtenção de dados através de um único endpoint (v4 da API), utilizando queries seletivas.

1.4 Objetos Experimentais

O objeto de estudo selecionado é a plataforma de desenvolvimento de software **GitHub**. A escolha justifica-se pela disponibilidade de duas versões distintas de sua API pública (v3 REST e v4 GraphQL) acessando a mesma base de dados real, hospedadas na mesma infraestrutura global, garantindo a comparabilidade justa entre as tecnologias.

1.5 Cenários de Consulta

Para avaliar o desempenho das tecnologias sob diferentes condições de carga e complexidade, o experimento foi dividido em três cenários distintos. Cada cenário isola uma característica específica das arquiteturas REST e GraphQL.

1. **Cenário 1: Consulta Escalar (Linha de Base)**
    - **Objetivo:** Estabelecer uma linha de base (*baseline*) de conexão e latência mínima, solicitando dados que não possuem relacionamentos complexos.
    - **Dados Solicitados:** Informações públicas do perfil da organização "facebook"(Nome, Descrição/Bio, Website e Localização).
    - **Execução REST:** Uma única requisição `GET /orgs/facebook`.
    - **Execução GraphQL:** Uma query plana solicitando apenas os quatro campos escalares.
    - **Expectativa:** Diferença de desempenho mínima, servindo como controle para os demais testes.
2. **Cenário 2: Listagem de Coleção (Teste de Overfetching)**

- **Objetivo:** Avaliar a eficiência na transferência de dados em listas (*Collections*). Este cenário isola o problema do *Overfetching* (trazer dados desnecessários).
- **Dados Solicitados:** Lista dos **50 repositórios** mais populares da organização, recuperando estritamente o *Nome* e a *Quantidade de Estrelas*.
- **Execução REST:** Uma requisição `GET /orgs/facebook/repos?per_page=50`. *Obs:* A API REST retorna a representação completa de cada repositório (dezenas de campos como URLs, datas de criação, permissões, forks, etc.) que serão descartados pelo cliente.
- **Execução GraphQL:** Uma query solicitando os primeiros 50 nós, selecionando apenas os campos `name` e `stargazerCount`.
- **Expectativa:** O REST apresentará um tamanho de resposta (payload) significativamente maior, impactando o consumo de banda.

3. **Cenário 3: Dashboard Complexa (Teste de Latência e Underfetching)**
   - **Objetivo:** Simular uma interface rica ("Dashboard") que exige dados aninhados de múltiplas entidades relacionadas. Este cenário testa o impacto de múltiplos *round-trips* de rede (problema N+1).
   - **Dados Solicitados:** Os 5 primeiros repositórios, incluindo para cada um:
     (a) Títulos das 3 últimas *Issues* (tarefas) abertas.
     (b) Lista das 3 principais linguagens de programação utilizadas.
   - **Execução REST (Problema 2N+1):** A API REST não fornece os dados aninhados na rota de listagem. São necessárias:
     - 1 requisição para listar os repositórios.
     - 5 requisições para buscar as *Issues* de cada repositório.
     - 5 requisições para buscar as *Linguagens* de cada repositório.
     - **Total:** 11 Requisições HTTP.
   - **Execução GraphQL:** Uma única requisição aninhada recupera a árvore completa de dados (Organização → Repositórios → Issues & Linguagens).
   - **Expectativa:** O GraphQL será drasticamente mais rápido devido à eliminação da latência de rede acumulada das múltiplas chamadas REST.

1.6 Tipo de Projeto Experimental

O experimento segue um desenho de **um fator com dois níveis** (*One factor with two treatments*). A abordagem será **pareada** (*paired design*), onde a mesma solicitação lógica

de dados será executada em ambas as tecnologias, permitindo a comparação direta dos deltas de desempenho para o mesmo conjunto de informações.

1.7 Quantidade de Medições

Para garantir a significância estatística e mitigar o ruído da rede pública:

- Cada cenário será executado **30 vezes** para cada tratamento.
- As primeiras 5 execuções de cada bateria serão descartadas (efeito de aquecimento de cache/DNS).
- Serão calculadas a Média e o Desvio Padrão das medições válidas.

1.8 Ameaças à Validade

Considerando o uso de uma API pública de terceiros, as seguintes ameaças foram mapeadas:

- **Validade Interna (Latência de Rede Não Controlada):** *Ameaça:* Como o servidor do GitHub não está em ambiente local, variações na rota da internet podem afetar o tempo de resposta (RQ1). *Mitigação:* Execução das baterias de teste de forma intercalada e utilização da média de 30 amostras para diluir *outliers*.
- **Validade de Construção (Rate Limiting):** *Ameaça:* O GitHub impõe limites de requisições que podem bloquear o teste. *Mitigação:* Utilização de *Personal Access Token* (PAT) para autenticação, elevando o limite para 5.000 requisições/hora, volume suficiente para o experimento proposto.
- **Validade de Construção (Payload):** *Ameaça:* A API REST retorna campos desnecessários por padrão (*Overfetching*), o que "prejudica"seu desempenho na métrica de tamanho. *Mitigação:* Esta característica não é um erro do experimento, mas sim o fenômeno central que se deseja comprovar (a eficiência do GraphQL em evitar tráfego inútil).
- **Validade Externa (Generalização e Uniformidade):** *Ameaça:* Os resultados obtidos são dependentes da implementação específica do GitHub. É possível que, em outros servidores onde a camada REST seja altamente otimizada (com caching agressivo) e a camada GraphQL seja ineficiente, os resultados se invertam. *Mitigação:* O estudo não reivindica que o GraphQL é superior em *todos* os cenários universais, mas sim que, em ecossistemas complexos e maduros como o do GitHub, ele oferece vantagens mensuráveis de desempenho para consultas aninhadas.