GraphQL vs REST

Desenho e Preparação do Experimento

Gabriel Henrique Miranda Rodrigues

1 Desenho do Experimento

Este experimento tem como objetivo comparar o desempenho de APIs GraphQL e REST de forma quantitativa. Para isso, vamos responder duas perguntas de pesquisa:

- **RQ1:** Respostas às consultas GraphQL são mais rápidas que respostas às consultas REST?
- **RQ2:** Respostas às consultas GraphQL têm tamanho menor que respostas às consultas REST?

A API do GitHub foi escolhida como base para o experimento porque oferece tanto uma API REST quanto uma API GraphQL para os mesmos dados, permitindo uma comparação justa.

1.1 A. Hipóteses Nula e Alternativa

As hipóteses definem o que queremos testar estatisticamente. O nível de significância adotado é de 5% (alfa = 0,05).

Para RQ1 (Tempo de Resposta):

- **H0 (Nula):** Não existe diferença significativa no tempo de resposta entre GraphQL e REST.
- **H1 (Alternativa):** Existe diferença significativa no tempo de resposta entre GraphQL e REST.

Para RQ2 (Tamanho da Resposta):

- **H0 (Nula):** Não existe diferença significativa no tamanho das respostas entre GraphQL e REST.
- **H1 (Alternativa):** Existe diferença significativa no tamanho das respostas entre GraphQL e REST.

Se o p-valor obtido nos testes for menor que 0,05, rejeitamos a hipótese nula e aceitamos que existe diferença significativa.

1.2 B. Variáveis Dependentes

As variáveis dependentes são aquelas que medimos durante o experimento:

- **Tempo de resposta:** tempo entre o envio da requisição e o recebimento da resposta completa, medido em milissegundos (ms).
- **Tamanho da resposta:** quantidade de dados retornados pela API, medido em bytes.

1.3 C. Variáveis Independentes

As variáveis independentes são aquelas que controlamos no experimento:

- **Tipo de API:** pode ser REST ou GraphQL.
- **Complexidade da consulta:** pode ser simples, média ou complexa, dependendo da quantidade de dados solicitados.

1.4 D. Tratamentos

Os tratamentos são as combinações das variáveis independentes. Como temos 2 tipos de API e 3 níveis de complexidade, temos 6 tratamentos no total:

Tratamento	Tipo de API	Complexidade
T1	REST	Simples
T2	REST	Média
T3	REST	Complexa
T4	GraphQL	Simples
T5	GraphQL	Média
T6	GraphQL	Complexa

Tabela 1: Matriz de tratamentos do experimento

1.5 E. Objetos Experimentais

Os objetos experimentais são os repositórios do GitHub sobre os quais as consultas serão feitas. Foram escolhidos 10 repositórios populares e de diferentes áreas:

Repositório	Área
facebook/react	Framework JavaScript
microsoft/vscode	Editor de código
tensorflow/tensorflow	Machine Learning
torvalds/linux	Sistema operacional
django/django	Framework Python
python/cpython	Linguagem de programação
nodejs/node	Runtime JavaScript
kubernetes/kubernetes	Orquestração de containers
angular/angular	Framework TypeScript
vuejs/vue	Framework JavaScript

Tabela 2: Repositórios utilizados no experimento

Esses repositórios foram escolhidos por serem ativos, terem bastante dados (issues, contribuidores, branches) e representarem diferentes tipos de projetos.

1.6 F. Tipo de Projeto Experimental

O tipo de projeto é um **fatorial completo**, onde todas as combinações de tratamentos são testadas em todos os objetos experimentais. Isso permite analisar o efeito de cada variável independente e também a interação entre elas.

Cada tratamento é aplicado a cada repositório várias vezes (100 repetições), e a ordem de execução é aleatorizada para evitar que fatores externos influenciem os resultados.

1.7 G. Quantidade de Medições

O número total de medições foi calculado da seguinte forma:

- 6 tratamentos (2 APIs $\times$ 3 complexidades)
- 10 repositórios
- 100 repetições para cada combinação

Total: $6 \times 10 \times 100 = 6.000$ medições

Esse número foi escolhido para garantir que os resultados sejam estatisticamente confiáveis.

1.8 H. Ameaças à Validade

Alguns fatores podem afetar a validade dos resultados:

- **Variação de rede:** a latência da internet pode variar durante o experimento. Para minimizar isso, fizemos 100 repetições de cada medição e aleatorizamos a ordem.
- **Cache:** as APIs podem guardar respostas em cache, retornando mais rápido na segunda vez. Para evitar isso, usamos headers que desativam o cache.

- **Rate limiting:** o GitHub limita o número de requisições por hora. Para não ultrapassar o limite, esperamos 100ms entre cada requisição.
- **Generalização:** os resultados são válidos para a API do GitHub. Outras APIs podem ter comportamentos diferentes.
- **Horário de execução:** o experimento foi executado em um período contínuo para evitar variações entre dias diferentes.

2 Preparação do Experimento

Nesta etapa, montamos todo o ambiente necessário para executar o experimento.

2.1 Ambiente e Ferramentas

- **Linguagem:** Python 3
- **Biblioteca para requisições HTTP:** requests
- **APIs utilizadas:** GitHub REST API v3 e GitHub GraphQL API v4
- **Análise de dados:** pandas, scipy, numpy
- **Visualização:** matplotlib, seaborn

2.2 Autenticação

Para acessar as APIs do GitHub, foi necessário criar um token de acesso pessoal com permissões de leitura de repositórios. Esse token é enviado no header de cada requisição.

2.3 Definição das Consultas

As consultas foram definidas em três níveis de complexidade:

Consulta Simples:

- REST: 1 requisição GET para /repos/{owner}/{repo}
- GraphQL: 1 query pedindo nome, descrição, stars e forks
- Dados retornados: informações básicas do repositório

Consulta Média:

- REST: 2 requisições GET (repositório + últimos 10 issues)
- GraphQL: 1 query pedindo repositório com issues aninhados
- Dados retornados: repositório + lista de issues

Consulta Complexa:

- REST: 4 requisições GET (repositório + issues + contributors + branches)
- GraphQL: 1 query pedindo todos os dados de uma vez
- Dados retornados: repositório + issues + contribuidores + branches

2.4 Procedimentos de Execução

1. Configurar o token de acesso do GitHub no script
2. Executar 5 requisições de aquecimento (warm-up) que são descartadas
3. Criar lista com todas as combinações de tratamentos e repositórios
4. Aleatorizar a ordem dessa lista
5. Para cada item da lista, executar a consulta e registrar tempo e tamanho
6. Esperar 100ms entre cada requisição
7. Salvar todos os resultados em um arquivo CSV

2.5 Métricas Coletadas

Para cada requisição, o script registra:

- Data e hora da execução
- Tipo de API (REST ou GraphQL)
- Nível de complexidade
- Nome do repositório
- Número da execução
- Tempo de resposta em milissegundos
- Tamanho da resposta em bytes
- Código de status HTTP