Case Studies of RestAPI and GraphQL Architecture

Janni Daniel Balraj
School of Enterprise Computing and Digital Transformation, TU Dublin, Ireland
X00193260@myTUDublin.ie

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
DUBLIN
TECHNOLOGICAL UNIVERSITY DUBLIN

Introduction

The adoption of microservices architecture has experienced significant growth, driven by its appeal in terms of modularity, scalability, and deployment ease. However, this proliferation of microservices has intensified the demand for efficient data exchange among them. While REST API has long been the standard protocol for microservices communication, its limitations in flexibility and performance have spurred the ascent of GraphQL as a more efficient alternative, as highlighted by studies comparing their performance. As the number of microservices continues to rise, traditional methods like REST APIs prove challenging, leading to issues such as over-fetching or under-fetching of data. GraphQL addresses these challenges by allowing each microservice to define its schema, thereby simplifying data integration and reducing coordination efforts between teams. In addition to its flexibility, GraphQL contributes to enhanced performance by empowering clients to specify their precise data requirements, resulting in a reduction of unnecessary requests and improved response times. Despite these advantages, the integration of GraphQL with microservices presents its own set of challenges, including ensuring data consistency across services and requiring upfront planning for schema and resolver functions. This research paper aims to delve into the integration of GraphQL with microservices, conducting a comparative performance analysis with REST API in an on-premises environment.

Research Questions

The conventional approach, using RESTful APIs for communication, often encounters challenges related to over-fetching and under-fetching of data, hindering system performance and efficiency. The emergence of GraphQL offers a compelling solution to these challenges, allowing precise data retrieval and reducing the volume of redundant requests in a microservices environment.

The core problem addressed in this research is:

Q1 - To what extent does the integration of GraphQL improve the efficiency and performance of data exchange between microservices when compared to RESTful APIs, and

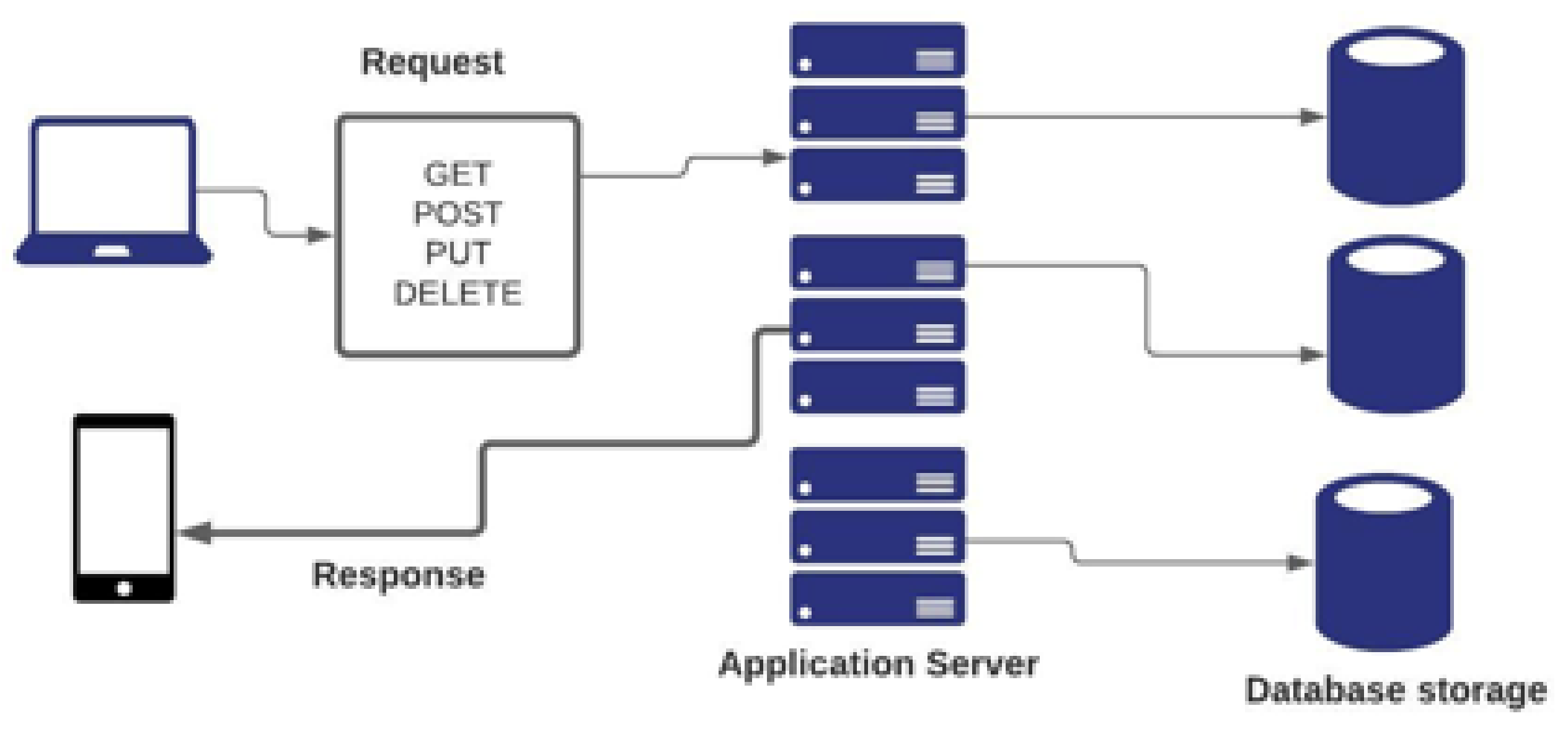
Q2 - what are the implications and challenges of implementing GraphQL in a microservices architecture?

Test Strategy and Methodology

- Functional Testing, Performance Testing and Usability Testing
- Data Preparation: 10 million stock records were generated and inserted into the PostgreSQL database.
- Load tests were conducted using Apache JMeter to simulate concurrent user requests for stock data retrieval.

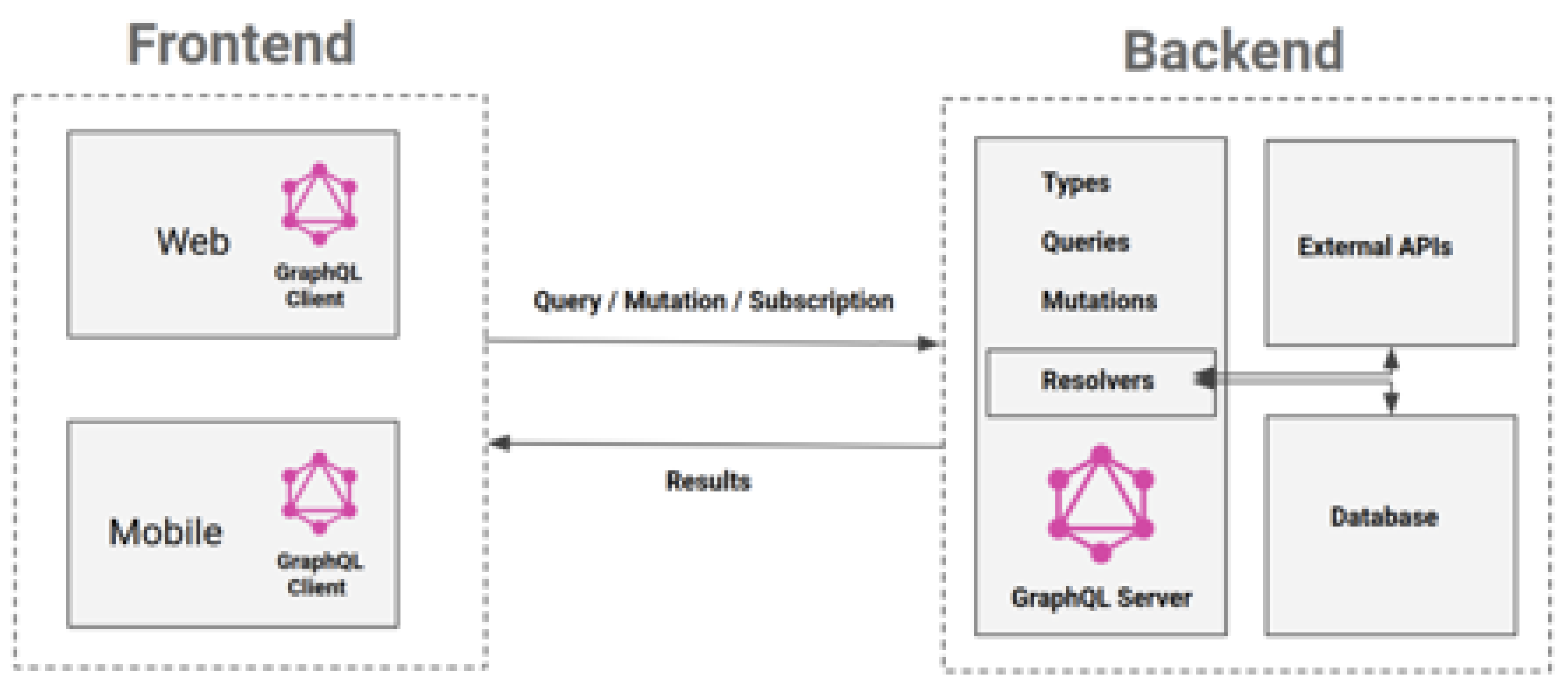
Architecture

1. REST API Architecture:



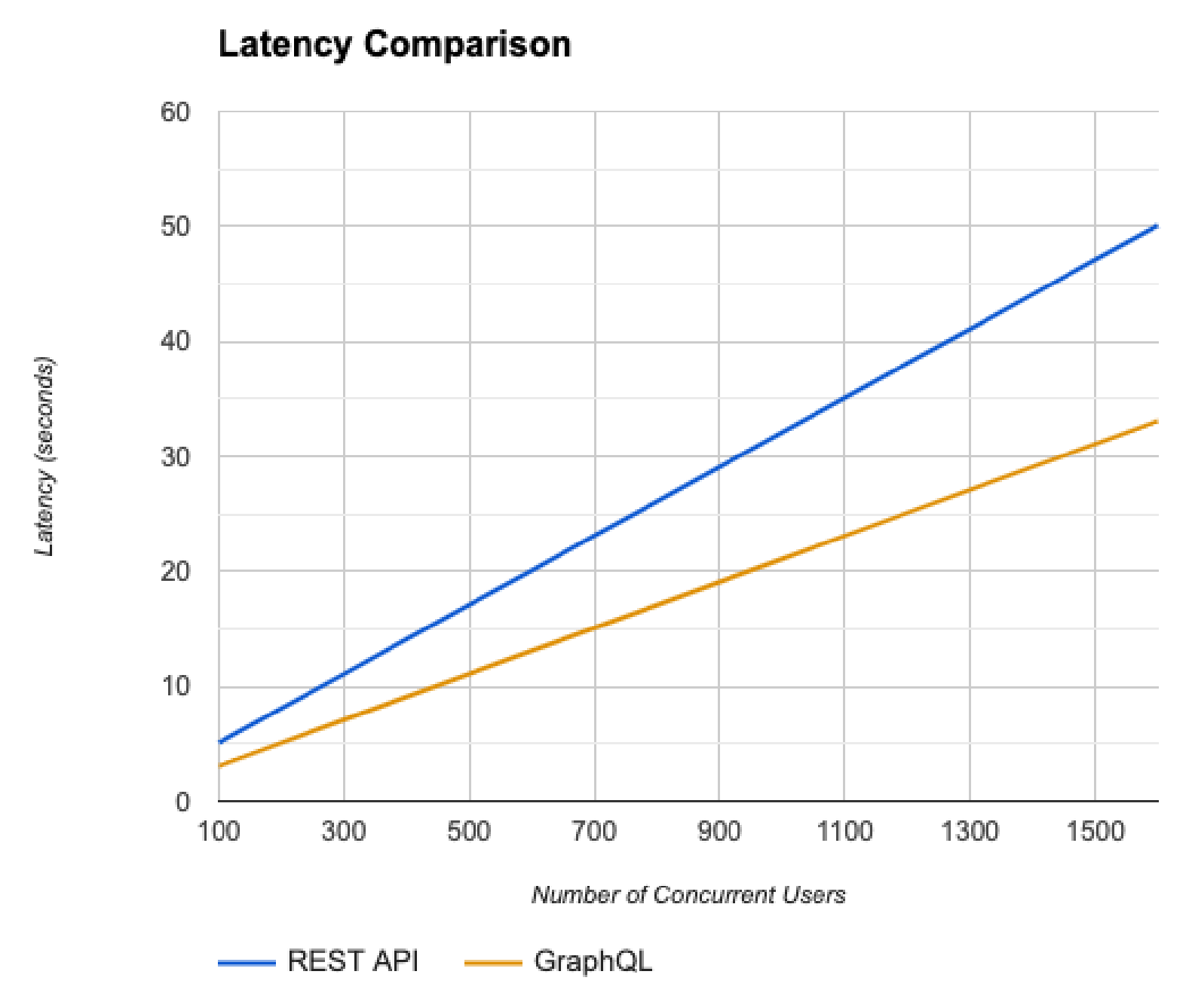
Stateless Communication: REST APIs are stateless, meaning that each request from a client to a server contains all the information needed to understand and fulfill the request. The server does not store any information about the client's state between requests.
Resource-Based: Resources, such as data objects or services, are identified by URLs (Uniform Resource Locators). Clients interact with these resources using standard HTTP methods like GET, POST, PUT, and DELETE.
Representation: Resources are represented in a format such as JSON or XML, and clients can manipulate these representations.

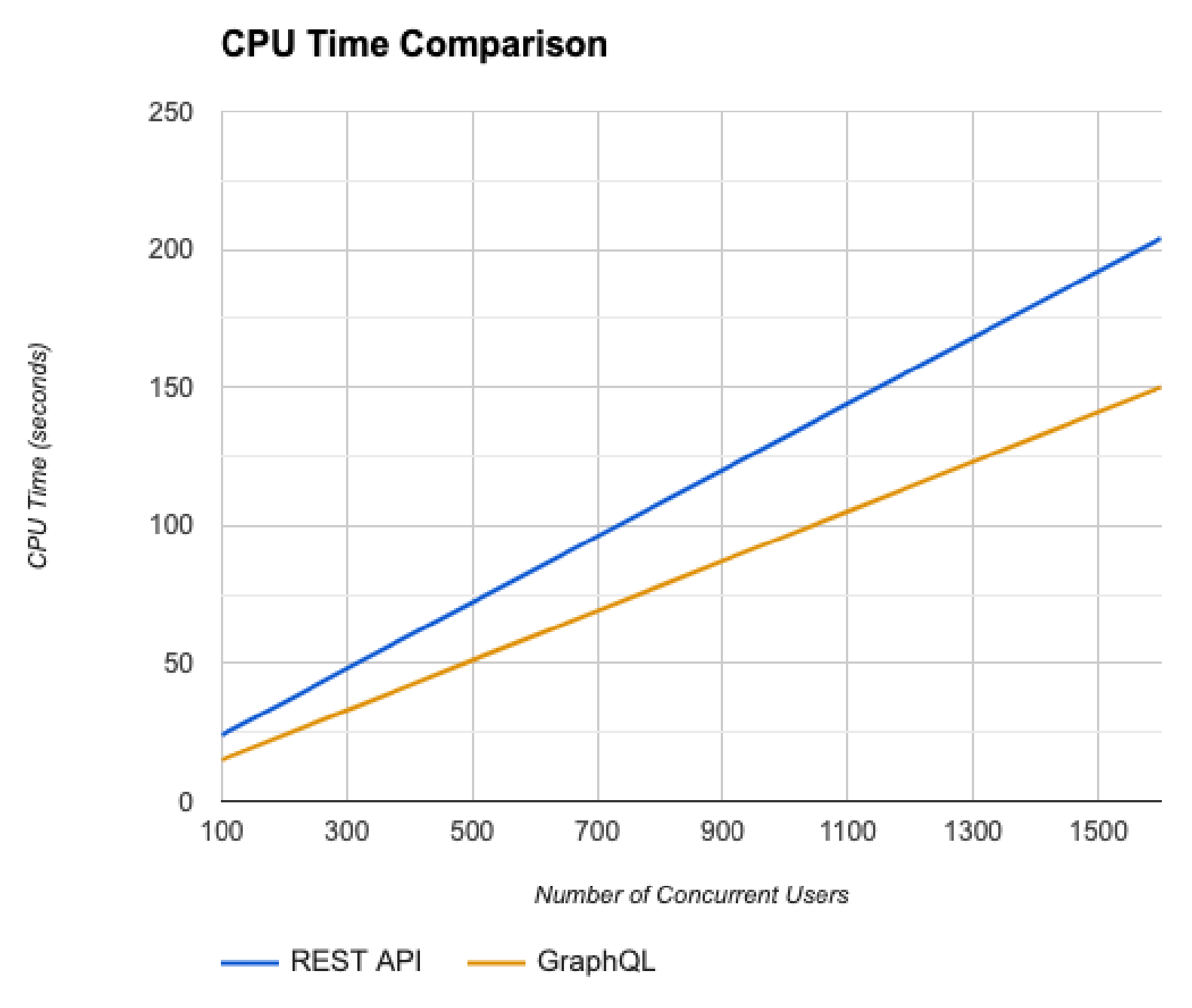
2. GraphQL Architecture:

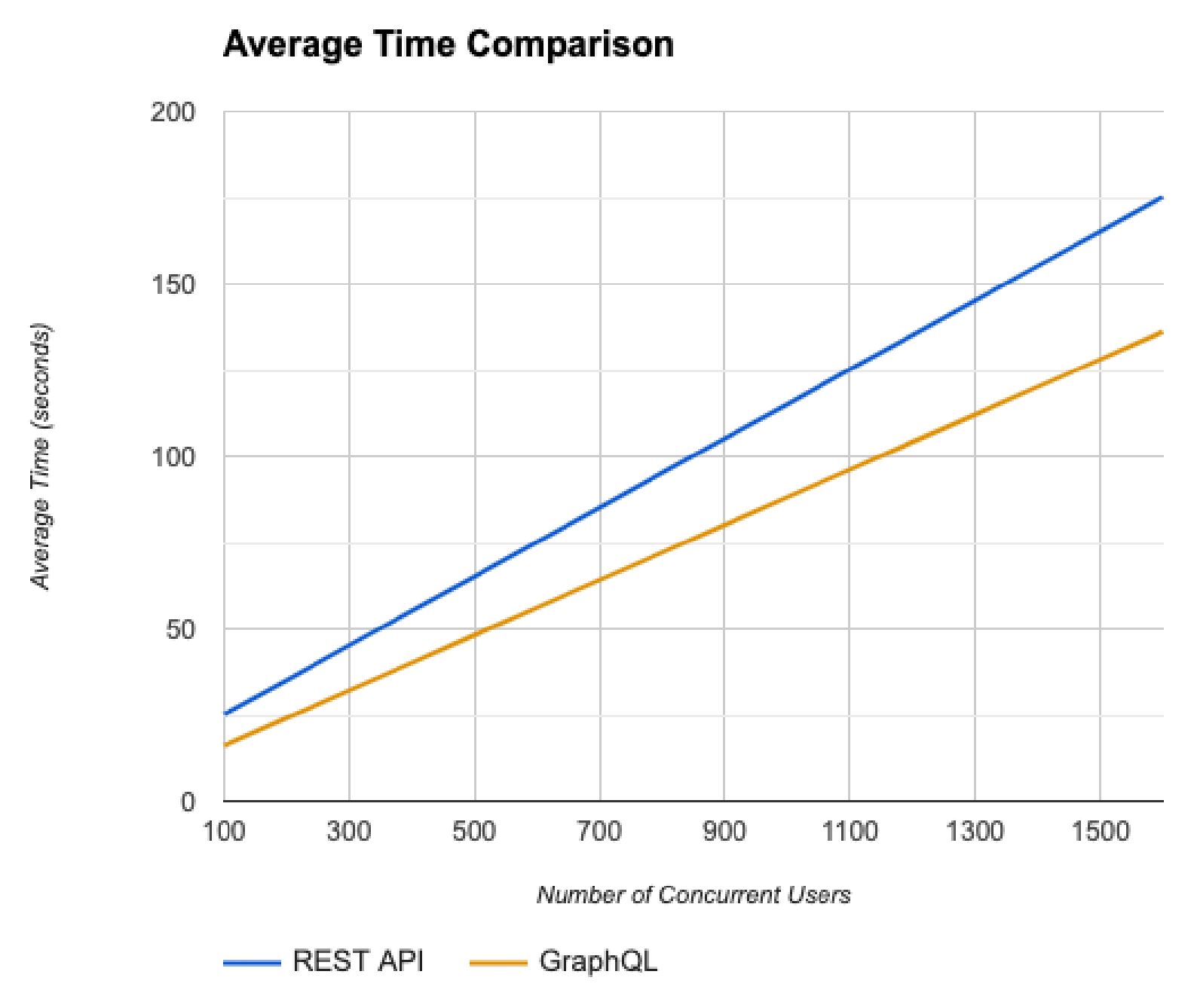


Query Language: GraphQL is a query language for APIs. Clients can request only the data they need, and the server responds with the requested data in a single JSON object.
Hierarchical Structure: The structure of a GraphQL query mirrors the structure of the data it retrieves. This hierarchical nature allows clients to request nested data in a single query.
Single Endpoint: Unlike REST, which often has multiple endpoints for different resources, GraphQL typically exposes a single endpoint for all interactions

Performance Graph: RESTAPI vs GraphQL







Conclusions and Future Work

In real-time data scenarios, GraphQL surpasses REST API by efficiently delivering specific data fields, thereby reducing network traffic and enhancing responsiveness. The adaptability of GraphQL enables seamless integration of new data sources without disrupting existing endpoints in complex applications. Looking ahead, the current implementation in a local environment is slated for deployment in cloud platforms like AWS, GCP, or Azure. The future plan includes enhancing scalability and establishing a resilient architecture to optimize performance and reliability.

QR Code for Recording