GraphQL vs. REST: Investigating Performance and Scalability for Serverless Data Persistence

Runjie Jin
rjjin@uw.edu

2025 IEEE International Conference on Cloud Engineering (IC2E '25)

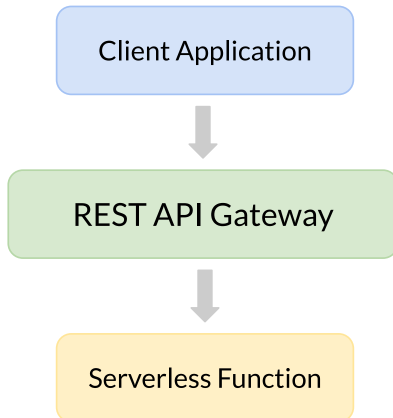
September 25, 2025

School of Engineering and Technology
University of Washington Tacoma

Outline

- Background and Motivation
- Research Questions
- Data-intensive Relational Database Use Case
- Conclusions and Future Work

Serverless Computing



Serverless function-as-a-service (FaaS) platforms offer various features:

- No infrastructure management
- Automatic scaling
- Event-driven architecture
- Pay-per-use billing model

A typical REST (Representational State Transfer)-based architecture can be represented by three layers:

Client application, API Gateway, and the actual serverless functions

REST Interface Challenges

Over-fetching

Retrieve unnecessary data:

- Increased bandwidth
- Higher processing overhead
- Increased cost

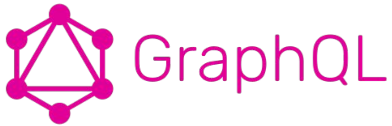
Under-fetching

Multiple API calls:

- Higher latency
- More round-trips
- Complex client logic

Impact on Serverless Applications:

- Performance: increased execution time and latency
- Resources: Higher resource usage
- Cost: additional compute time and data transfer



Why GraphQL?

Key benefits of GraphQL:

- **Precise data retrieval**
- **Single request solution: server-side orchestration**
- **Strong typing**
- **Real-time support and more**

Companies are starting to use GraphQL

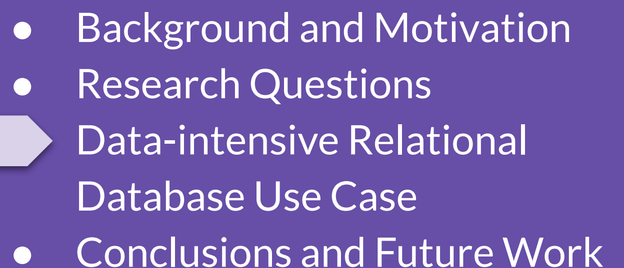
Outline

- Background and Motivation
- Research Questions
- Data-intensive Relational Database Use Case
- Conclusions and Future Work

Research Questions

- RQ-1 (***Relational DB API performance***): How well does GraphQL perform in serverless environments compared to REST for DB.
- RQ-2 (***Relational DB managed vs. unmanaged***): How do GraphQL services and REST scale under increasing concurrency for the data-intensive workload.

Outline



Data-intensive Relational Database Use Case

Database setup

- AWS Aurora PostgreSQL 16.4 cluster, db.r5.4xlarge
- 2018 U.S. Centers for Medicare & Medicaid Services Open Payments Dataset
    - General payments: 10.8M rows, 5.9 GB
    - Research payments: 0.58M rows, 478 MB
    - Ownership payments: 3.3k rows, 1.5 MB
- 9 API endpoints: lookup, count, aggregation

Server Implementations

- AppSync GraphQL: managed AWS service
- Apollo GraphQL: self-hosted on EC2 c7i.8xlarge
- REST API: Amazon API Gateway + Lambda

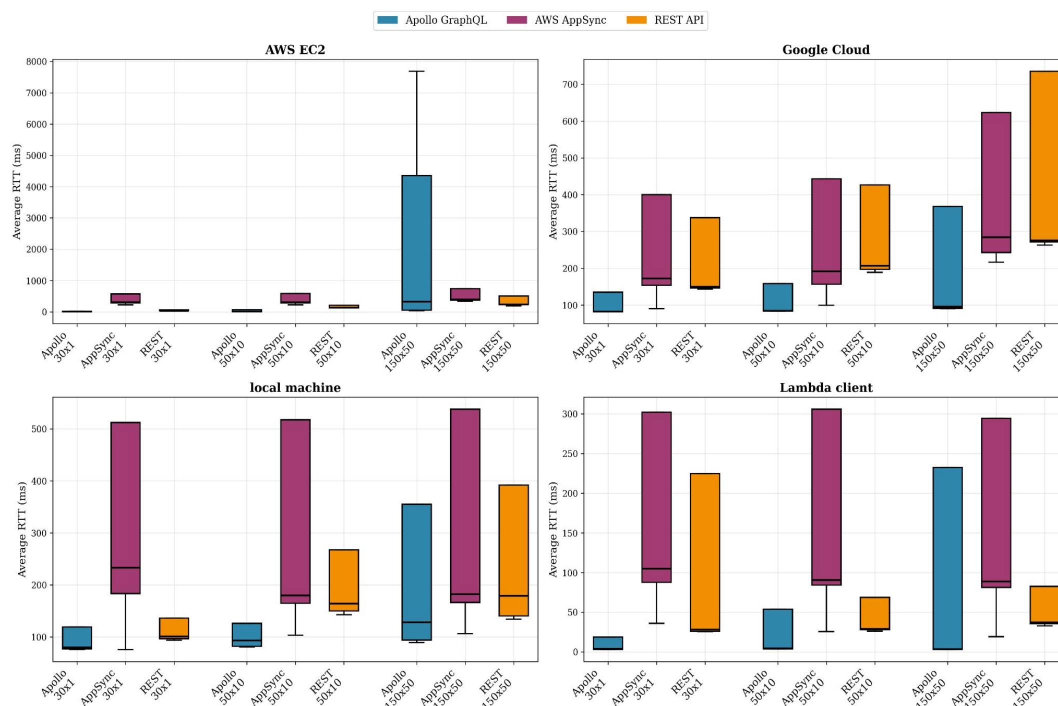
Test Infrastructure

- Clients: local, EC2 VM, GCP VM, Lambda
- Test scenarios(calls x thrd): 30x1, 50x10, 150x50
- Concurrency scaling: 1-100 threads
- Metrics: RTT, throughput

Reasoning for the setup

- Database-intensive operations
- Multiple query patterns
- High concurrency testings

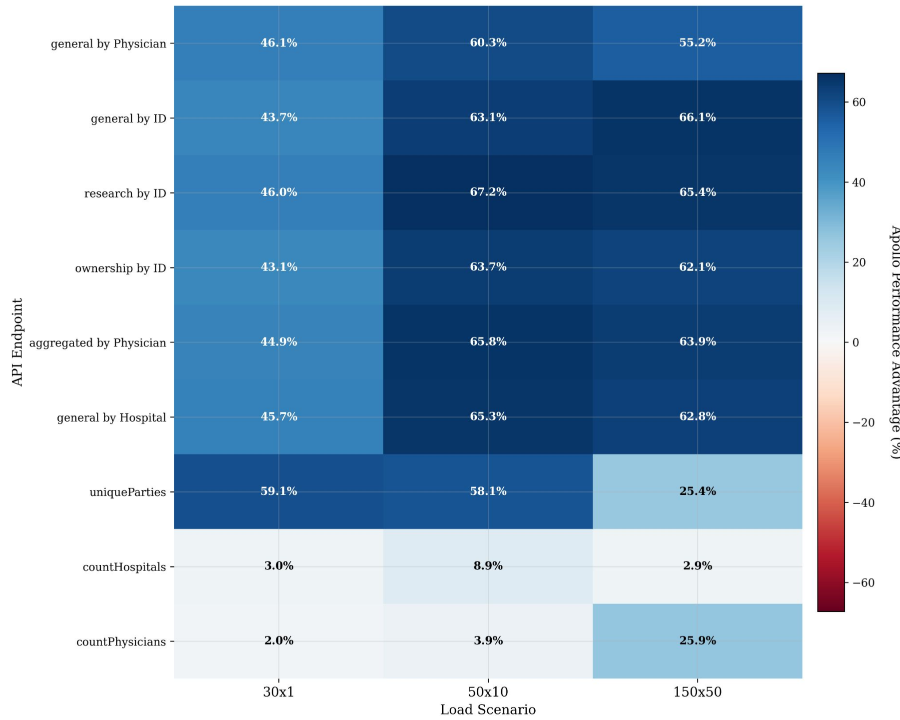
GraphQL vs. REST Performance (RQ-1)



RTT distributions for clients

- Apollo provides the lowest median RTT most of the time, but sensitive to test clients
- AppSync gives consistently high RTT
- REST is moderately sensitive to test clients, but is slower than Apollo

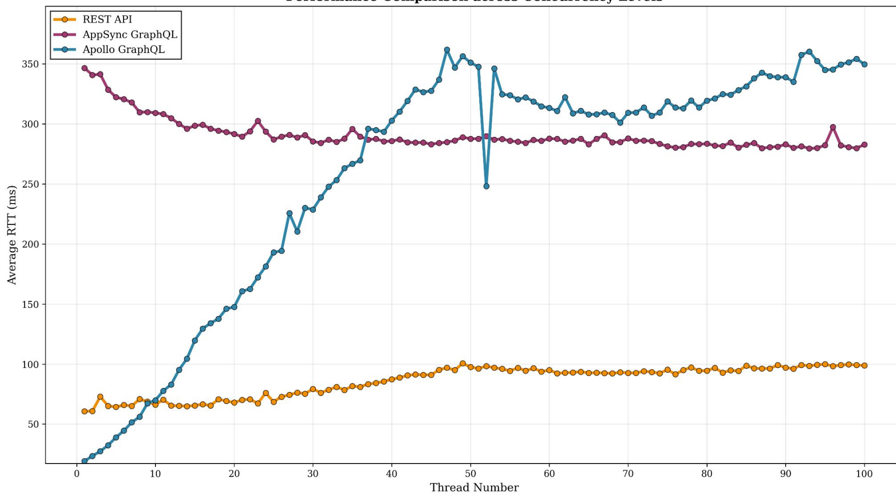
GraphQL vs. REST Performance (RQ-1)



Endpoint wise comparison Apollo vs. REST

- Apollo is consistently faster for these testing scenarios
- Best performance comes from the endpoints that are fast to execute
- For endpoints that take long DB time (last two), the advantage isn't very noticeable

Scaling comparisons (RQ-2)



Scaling test for three APIs under heavy load

- Test runs = 30 times thread number
- **Apollo** is fast as first, but with only one instance server, its performance quickly degrades
- **AppSync** is consistent, with an initial warm up phase, ending at around 280 ms
- **REST** is finally the fastest of all 3 under heavy load

Outline

- Background and Motivation
- Research Questions
- The Data-intensive Relational Database Case
- Conclusions and Future Work

Conclusion Summary

RQ-1: GraphQL API Performance

How well does GraphQL perform for the database use case on round trip-time and cost?

- Apollo showed 25-67% RTT improvement over REST in most operations
- Performance advantage is best for simple operations
- AppSync is consistent, but gives higher RTT across environments

RQ-2: GraphQL Managed vs. Unmanaged

What are the performance differences for hosting GraphQL APIs using unmanaged vs. managed solutions?

- REST showed best scalability with consistent 60-100ms RTT
- Apollo excelled at low concurrency but degrades heavily
- AppSync RTT drops, but is still consistently high

Future work

More workloads and use cases
Try complex, hybrid workloads with both CPU processing and database queries. Investigate aggregated database queries in future use cases.

Multi-platform and Scalability
Investigate other cloud platforms (e.g. Google Cloud, Azure) to study vendor lock-in issues. Use Apollo federation or Kubernetes to horizontally scale Apollo server.

Advanced GraphQL Features
Other than basic CRUD, future work can study advanced features provided by GraphQL like real-time subscriptions and data streaming in serverless contexts.

Thank You!