GraphQL vs. REST: A Performance and Cost Investigation for Serverless Applications

Runjie Jin, Robert Cordingly, Dongfang Zhao, Wes Lloyd
rjjin@uw.edu

School of Engineering and Technology
University of Washington Tacoma

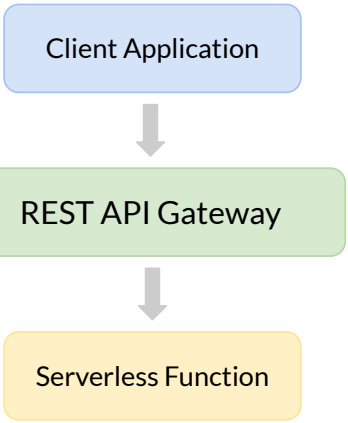
December 2, 2024

25th ACM/IFIP International Middleware Conference
MIDDLEWARE 2024

Outline

- Background and Motivation
- Research Questions
- Methodology
- Results
- Conclusions

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 function

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:

- **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: aggregating individual REST function calls**
- **Strong typing**
- **Real-time support and more**

Companies are starting to use GraphQL

Outline

- Background and Motivation
- Research Questions
- Serverless Proxy System
- Methodology and Results
- Conclusions

Research Questions

- RQ-1 (***GraphQL API performance***): How well does GraphQL perform in serverless environments?
  - Round-trip time, cost, performance vs. REST APIs

- RQ-2 (***GraphQL managed vs. unmanaged***): What are the tradeoffs between managed vs. self hosting?
  - Managed AWS AppSync
  - Unmanaged Apollo server

Outline

- Background and Motivation
- Research Goals
- Methodology
- Results
- Conclusions

Image Processing Pipeline

Pipeline Characteristics

- 7 independent functions: rotate, flip, crop, brighten, contrast, grayscale, resize
- 4.8 MB test image (AppSync 5MB limit on payload size)
- Configurable filter ordering

Suitability for Evaluation

- Computationally intensive
- Heavy I/O
- Multi-stage workflow
- Extensible: supports addition of new functions

Implementation Comparison

GraphQL Implementation

- Single request
- GraphQL resolvers invoke lambda functions
- Built-in pipeline orchestration

REST Implementation

- Client-side workflow orchestration
- Multiple client-to-cloud round-trips
- API Gateway Integration

Test Infrastructure

Serverless Backend (us-east-2)

AWS Lambda Configuration
- ARM64 Graviton2 Lambda functions
- No hyperthread to reduce variance
- Python functions

API Gateway Setup
- REST API configuration
- Direct Lambda integration
- Standard endpoints

GraphQL Servers

AWS AppSync
- Fully managed service
- Auto-scaling
- Direct AWS service integration

Apollo Server
- C7i.8xlarge EC2 instance
- 32 vCPUs, 64 GB RAM
- 3.2 GHz Intel Xeon Platinum

Test Clients

Local Desktop (WA)
- 32 GB RAM, Intel i5-13600K
- 350 Mbps Bandwidth
- Tests high-latency scenarios

AWS EC2
- C7i.8xlarge, us-west-2
- 64 GB RAM, 12.5 Gbps
- Tests same-cloud performance

Google Cloud
- C3.standard-8, us-west-2
- 32 GB RAM, 32 Gbps
- Tests cross-cloud performance

AWS Lamdba
- us-east-2
- Tests scalability performance

Outline

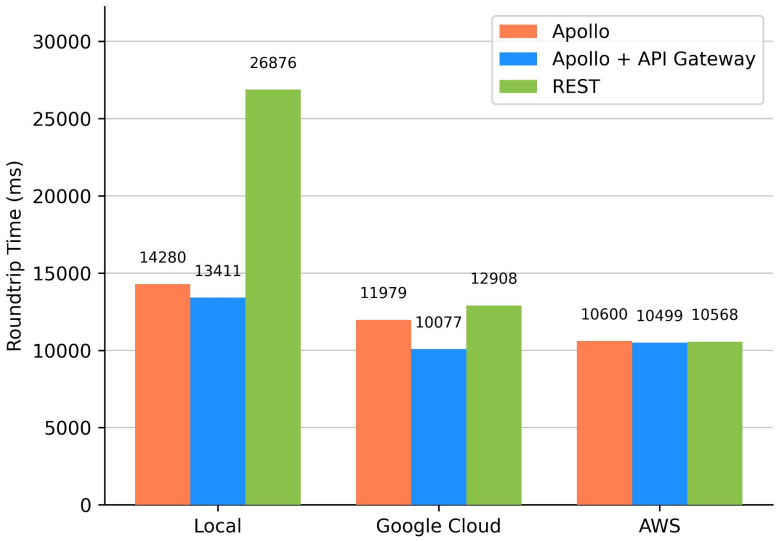
- Background and Motivation
- Research Goals
- Methodology
- Results
- Conclusions

Research Question 1

How well does GraphQL perform in the serverless environment with respect to **roundtrip-time**?

What are the performance implications in contrast to providing the same functionality using **REST APIs** to backend serverless functions?

GraphQL vs. REST: Roundtrip-time Performance



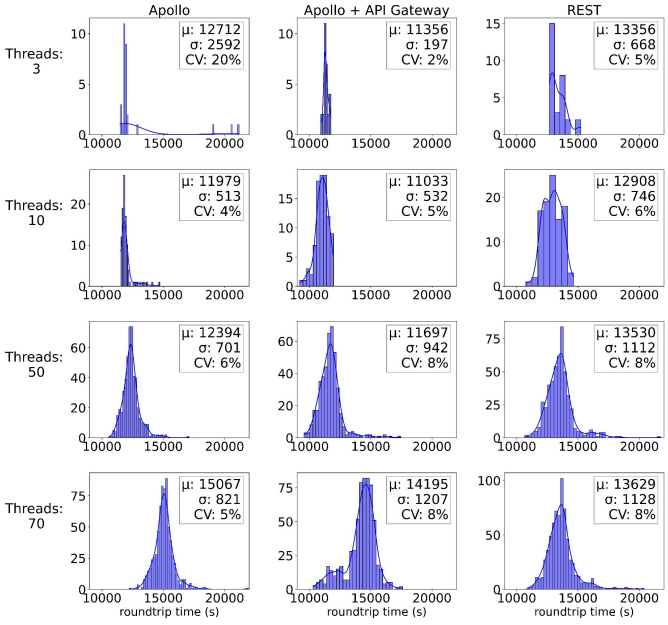
Clients:

- Local Desktop (WA)
- Google Cloud VM
- AWS VM

Key findings:

- Apollo + API Gateway best performance
- GraphQL shows significant advantage in high latency scenarios
- Performances are similar in cloud environment

Performance Distribution Analysis



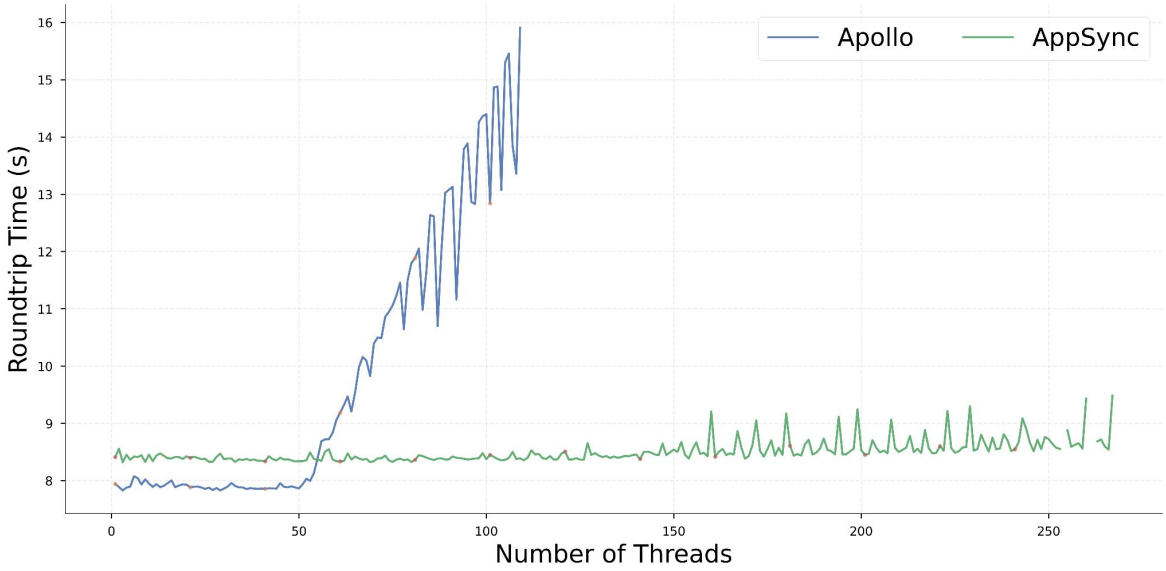
Distribution Patterns

- Low concurrency - more outliers: Apollo server had 20% variance due to cold starts
- High concurrency: 5%-8% variance - log normal distributions
- Near Bimodal at 70 threads for Apollo + API Gateway

Research Question 2

What are the **performance and cost differences** for hosting GraphQL APIs using an unmanaged self-hosted GraphQL server vs. a managed GraphQL service?

Managed vs. Unmanaged GraphQL Scalability

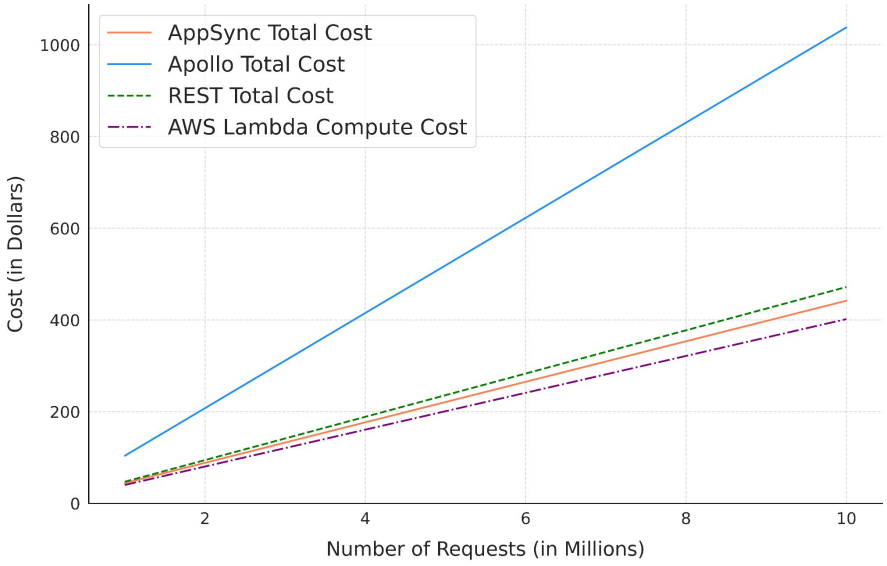


Apollo Server
- Better performance up to around 54 threads
- Scaling limitations

Appsync
- Better under high concurrency
- Automatic scaling

Implications
- High-scale considerations
- Deployment choices

Cost-performance Analysis



Cost Comparison

- AppSync: $4 per million
- Apollo: $63.62 per million
- REST: $7 per million

Considerations:

- Managed service is estimated to be cheaper
- Make deployment choices by actual needs

Outline

- Background and Motivation
- Research Goals
- Methodology
- Results
- Conclusions

Conclusion Summary

RQ-1: GraphQL API Performance

How well does GraphQL perform in the serverless environment with respect to roundtrip-time and cost?

- GraphQL consistently outperformed REST, especially in high-latency environments
- Apollo + API Gateway showed best performance among tested configurations
- Performance advantages diminish in cloud-native setups with low latency

RQ-2: GraphQL Managed vs. Unmanaged

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

- AppSync showed better scalability beyond 54 concurrent requests
- AppSync more cost-effective: $4 vs Apollo's $63.62 per million requests
- Apollo offered better RTT when not over-provisioned but showed scaling limitations

Deployment Recommendations

High-Latency Scenarios

Recommended: GraphQL

Edge computing environments

Mobile applications

Global distributed systems

Benefits from reduced round-trips

High-Scale Requirements

Recommended: AppSync

Large-scale applications

Variable workloads

Cost-sensitive deployments

Optimal cost-performance ratio

Specific Control Needs

Recommended: Apollo Server

Custom optimization needs

Lower request volumes

Specific infrastructure requirements

Greater deployment flexibility

Thank You!

Q & A