PERBANDINGAN EFISIENSI REST API DAN GRAPHQL PADA SISTEM PENJADWALAN PERKULIAHAN BERBASIS CLOUD-NATIVE DENGAN DOCKER, KUBERNETES, DAN LOAD BALANCING

PROPOSAL PENELITIAN AWAL

Tugas Mata Kuliah Metodologi Penelitian (IF25-41029)
Program Studi Teknik Informatika
Institut Teknologi Sumatera

Oleh:
Reyhan Capri Moraga
123140022

PROGRAM STUDI TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI SUMATERA
2025

DAFTAR ISI

BAB I
PENDAHULUAN

1.1 Latar Belakang

Transformasi digital di sektor pendidikan tinggi telah mendorong peningkatan ekspektasi terhadap ketersediaan dan performa layanan akademik. Sistem informasi krusial, seperti sistem penjadalan perkuliahan, dituntut untuk tidak hanya fungsional tetapi juga harus skalabel dan responsif, terutama saat menghadapi lonjakan lalu lintas data (traffic) secara tiba-tiba, seperti pada masa pengisian Rencana Studi Mahasiswa (RSM) [1]. Untuk menjawab tantangan ini, arsitektur aplikasi modern telah beralih dari sistem monolitik menuju arsitektur cloud-native yang memanfaatkan teknologi seperti kontainerisasi (Docker) dan orkestrasi (Kubernetes) untuk mencapai skalabilitas elastis dan ketahanan sistem (resilience) [2]. Dalam arsitektur ini, mekanisme load balancing menjadi krusial untuk mendistribusikan lalu lintas data secara merata ke berbagai copy layanan (kontainer), memastikan tidak ada satu layanan pun yang kelebihan beban dan meningkatkan ketersediaan layanan [3].

Inti dari sistem terdistribusi ini adalah Application Programming Interface (API), yang berfungsi sebagai jembatan komunikasi antar layanan. Selama bertahun-tahun, REST (Representational State Transfer) telah menjadi standar de-facto dalam perancangan API karena kesederhanaan dan adopsinya yang luas [4]. Namun, seiring dengan meningkatnya kompleksitas relasi data pada aplikasi modern—seperti sistem penjadalan yang melibatkan entitas mahasiswa, dosen, mata kuliah, dan ruangan—kelemahan REST mulai terlihat, terutama masalah over-fetching (mengambil data berlebih) dan under-fetching (memerlukan banyak request untuk mendapatkan data lengkap) [5].

Sebagai alternatif, GraphQL muncul sebagai query language untuk API yang menawarkan fleksibilitas tinggi, memungkinkan klien untuk meminta data yang spesifik hanya dengan satu request [6]. Sejumlah penelitian telah berupaya membandingkan efisiensi kedua arsitektur ini. Banyak studi awal, seperti yang dilakukan oleh Hartanto [4] dan Cerny & Donahoo [5], berfokus pada perbandingan performa dalam lingkungan aplikasi monolitik atau lingkungan pengujian lokal. Studi-Sistem-studi ini umumnya menemukan bahwa GraphQL unggul dalam mengurangi ukuran payload dan waktu respons untuk query yang kompleks.

Meskipun demikian, terdapat celah penelitian (research gap) yang signifikan. Sangat sedikit penelitian yang mengevaluasi kedua arsitektur API ini dalam konteks cloud-native yang sesungguhnya [2, 6]. Kinerja di lingkungan laboratorium (monolitik) mengabaikan variabel dinamis produksi seperti overhead latensi keamanan

(misal, validasi token) [7], latensi jaringan antar-layanan, service discovery, dan dampak langsung dari algoritma load balancing Kubernetes terhadap distribusi beban. Perbandingan yang adil (fair comparison) di lingkungan yang lebih realistis ini masih sangat dibutuhkan.

Oleh karena itu, penelitian ini akan melakukan analisis perbandingan efisiensi REST API dan GraphQL pada studi kasus sistem penjadalan perkuliahan, yang diimplementasikan sebagai testbed yang adil di atas platform cloud-native (Docker, Kubernetes) dan diuji menggunakan skenario load balancing untuk mensimulasikan beban kerja dunia nyata.

1.2 Rumusan Masalah

1. Bagaimana merancang sebuah testbed perbandingan yang adil (fair comparison) untuk prototipe backend REST dan GraphQL, yang mencakup simulasi overhead latensi keamanan (mock middleware), di atas platform Docker dan Kubernetes?
2. Seberapa signifikankah perbedaan kinerja (ditinjau dari throughput, latency, dan payload size) antara REST dan GraphQL saat diuji dengan skenario query yang bervariasi: (a) query sederhana (data tunggal), (b) query kompleks (relasi data majemuk), dan (c) simulasi under-fetching REST?
3. Manakah arsitektur API yang menunjukkan skalabilitas dan ketahanan (resilience) lebih baik ketika dihadapkan pada skenario beban tinggi (high load) yang didistribusikan melalui load balancer Kubernetes?

1.3 Tujuan Penelitian

1. Merancang dan mengimplementasikan prototipe backend sistem penjadwal (REST dan GraphQL) dengan jaminan fairness—termasuk implementasi mock middleware latensi—dan menjalankannya sebagai testbed di atas platform Docker dan Kubernetes.
2. Menganalisis secara kuantitatif perbedaan kinerja kedua API pada tiga skenario query spesifik (sederhana, kompleks, dan simulasi under-fetching) untuk metrik throughput, latency, dan payload size.
3. Mengevaluasi dan menentukan arsitektur API yang memiliki skalabilitas dan ketahanan (resilience) superior di bawah skenario pengujian beban tinggi (stress testing) dengan load balancing.

1.4 Batasan Masalah

1. Penelitian berfokus pada perbandingan performa API di sisi backend. Aspek frontend (UI/UX) dari aplikasi penjadwal tidak akan dikembangkan atau dievaluasi.
2. Logika bisnis atau algoritma untuk generating jadwal perkuliahan (misalnya, penyelesaian konflik jadwal) berada di luar cakupan penelitian. Sistem dianggap sudah memiliki data jadwal yang siap diakses melalui API.
3. Pengujian dilakukan dalam lingkungan cluster Kubernetes yang terkontrol. Variabel spesifik dari penyedia layanan cloud publik (seperti AWS, GCP, atau Azure) tidak akan dibahas secara mendalam.
4. Penelitian ini tidak mengimplementasikan sistem keamanan (otentikasi/otorisasi) secara fungsional. Namun, sebuah mock middleware yang mensimulasikan overhead latensi konstan (misal: 1-2ms untuk validasi token) akan ditambahkan pada kedua API untuk menjamin fairness perbandingan yang lebih mendekati kondisi nyata.
5. Penelitian tidak berfokus pada optimasi query di database layer. Kinerja basis data diasumsikan setara dan konstan untuk kedua API, yang akan menggunakan skema dan query data mentah yang serupa.
6. Upaya fairness dilakukan pada logika bisnis, namun perbedaan performa minor yang berasal dari library atau framework-specific (misal: ORM, GraphQL resolver) merupakan batasan yang diketahui.

1.5 Kontribusi Penelitian

1. Menyajikan data empiris dan analisis baru mengenai perbandingan performa REST API vs. GraphQL dalam konteks arsitektur cloud-native melalui sebuah testbed perbandingan yang adil (fair comparison) dan lebih mendekati skenario nyata (memasukkan simulasi overhead latensi dan load balancing).
2. Memberikan rekomendasi berbasis data bagi para arsitek sistem dalam memilih arsitektur API yang paling efisien untuk aplikasi berskala besar, khususnya yang memiliki karakteristik relasi data kompleks dan pola lalu lintas spiky (padat pada waktu-waktu tertentu).
3. Menghasilkan blueprint dan prototipe sistem sebagai testbed yang dapat direplikasi untuk penelitian lebih lanjut mengenai optimasi performa API di lingkungan cloud-native.

DAFTAR PUSTAKA

[1] Y. Pratama and A. B. Santoso, "Analisis kinerja sistem informasi akademik berbasis cloud computing saat terjadi lonjakan trafik," *Jurnal Teknologi Informasi dan Ilmu Komputer (JTIK)*, vol. 8, no. 3, pp. 501–510, 2021.

[2] L. Wang and J. Li, "Performance comparison of graphql and rest apis in microservices architecture," in *2022 IEEE International Conference on Web Services (ICWS)*, Shanghai, China, 2022, pp. 310–318.

[3] R. Sharma and P. K. Gupta, "Performance analysis of load balancing algorithms in kubernetes clusters," in *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, 2021.

[4] D. Hartanto, "Analisis performa graphql dan restful api pada aplikasi e-commerce monolitik," *Jurnal Teknika: Jurnal Sains dan Teknologi*, vol. 17, no. 1, pp. 45–54, 2021.

[5] T. Cerny and M. Donahoo, "Graphql vs. rest: A performance comparison," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 10, no. 1, pp. 1–15, 2021.

[6] A. Suryadi, "Payload size optimization using graphql in mobile applications," *International Journal of Computer Science and Network Security*, vol. 22, no. 4, pp. 112–119, 2022.

[7] A. F. A. Wibowo and I. D. P. H. S., "Analisis overhead latensi autentikasi jwt pada rest api gateway," *Jurnal Nasional Teknik Elektro dan Teknologi Informasi (JNTETI)*, vol. 10, no. 1, 2021.