Journal of Applied Informatics and Computing (JAIC)
Vol.9, No.2, April 2025, pp. 450~457
e-ISSN: 2548-6861

Comparative Performance Analysis of GRPC and Rest API Under Various Traffic Conditions and Data Sizes Using a Quantitative Approach

Moch. Zukhruf Ain[1]*, Rizka Ardiansyah[2]*, Septiano Anggun Pratama[3]*, Muhammad Akbar[4]*, Nouval Trezandy Lapatta[5]*
[*] Informatics Engineering, Engineering Faculty, Universitas Tadulako, Indonesia
zukhrufclash@gmail.com [1], rizka@untad.ac.id [2], septiano@untad.ac.id [3], akbarfatek@untad.ac.id [4], nouval@untad.ac.id [5]

Article Info

Article history:

Received 2025-03-12
Revised 2025-03-19
Accepted 2025-03-25

Keyword:

gRPC,
REST API,
Web 3.0,
quantitative analysis,
HTTP/2.

ABSTRACT

Web 3.0 presents challenges in efficient data exchange, especially in decentralized systems. REST API (HTTP/1.1) remains widely used due to its broad compatibility but has communication inefficiencies, while gRPC (HTTP/2) offers better performance with multiplexing and Protocol Buffers. This study compares REST API and gRPC under various traffic conditions and data sizes using Apache JMeter and Wireshark, measuring throughput, response time, latency, and data transfer efficiency. Results show that REST API has higher throughput in low-traffic scenarios (995 vs. 29.5 req/min) and faster GET response time (3 ms vs. 20 ms), while gRPC excels in large data transfers (276.34 KB/s vs. 134.1 KB/s) and stable latency (0.147 ms). However, ANOVA analysis ($p > 0.05$) indicates no statistically significant difference. REST API is ideal for standard web applications, while gRPC is suited for microservices and real-time systems.

I. PENDAHULUAN

Web 3.0 menghadirkan paradigma desentralisasi yang mengutamakan transparansi dan interoperabilitas dalam sistem digital. Tidak seperti Web 2.0 yang bergantung pada server terpusat, Web 3.0 memungkinkan aplikasi terdistribusi (DApps) beroperasi tanpa perantara dengan memanfaatkan teknologi blockchain dan jaringan peer-to-peer. Dalam konteks ini, efisiensi pertukaran data menjadi faktor krusial karena volume dan kompleksitas komunikasi antar aplikasi semakin meningkat. Salah satu komponen utama dalam komunikasi ini adalah Application Programming Interface (API), yang memungkinkan sistem berbeda untuk bertukar data dengan standar tertentu[1][2][3].

REST API berbasis HTTP/1.1 telah menjadi standar utama dalam pengembangan layanan web karena kesederhanaan, fleksibilitas, dan kompatibilitasnya yang luas. Namun, dengan meningkatnya skala sistem dan kebutuhan akan komunikasi yang lebih cepat serta efisien, REST API menghadapi beberapa tantangan. Salah satu tantangan utama adalah overhead tinggi akibat penggunaan format teks seperti JSON atau XML [4], yang memperbesar ukuran data yang dikirim dan meningkatkan waktu pemrosesan. Selain itu, HTTP/1.1 menggunakan model request-response yang menyebabkan antrean permintaan saat lalu lintas tinggi, yang berdampak pada peningkatan latensi dan penurunan throughput. [5] Sebagai alternatif, gRPC (Google Remote Procedure Call) menawarkan pendekatan yang lebih efisien dengan memanfaatkan HTTP/2 dan format data biner Protocol Buffers (protobuf), yang lebih optimal dalam komunikasi antar layanan terdistribusi. HTTP/2 memiliki fitur multiplexing yang memungkinkan beberapa permintaan dikirim dalam satu koneksi tanpa mengalami blocking seperti pada HTTP/1.1. Dengan demikian, gRPC dapat mengurangi latensi dan meningkatkan throughput dalam kondisi lalu lintas tinggi, menjadikannya pilihan menarik untuk sistem modern berbasis Web 3.0 [6][7][8].

Beberapa penelitian terdahulu telah membandingkan REST API dan gRPC dalam berbagai kondisi skenario. Studi menemukan bahwa gRPC memiliki keunggulan dalam latensi dan efisiensi bandwidth dibandingkan REST API pada aplikasi berbasis microservices[9][10]. Penelitian lain

menunjukkan bahwa meskipun REST API lebih sederhana dan banyak digunakan, gRPC lebih unggul dalam pemrosesan data dalam jumlah besar karena efisiensi serialisasi dan mekanisme transport yang lebih baik. Namun, REST API tetap menjadi pilihan utama dalam berbagai sistem karena ekosistem yang luas serta dukungan yang lebih baik di berbagai platform dan perangkat [11][12][13].

Dalam penelitian ini, dilakukan analisis kinerja REST API dan gRPC API dengan berbagai skenario uji untuk mengevaluasi performanya dalam kondisi lalu lintas yang berbeda serta variasi ukuran data. Pengujian dilakukan menggunakan Apache JMeter [14] dan Wireshark untuk mengukur throughput, response time, latency, efisiensi transfer data serta payload menggunakan requiuest method yang sesuai [15]. Hasil penelitian ini diharapkan dapat memberikan wawasan bagi pengembang dalam memilih teknologi API yang sesuai untuk sistem modern yang membutuhkan efisiensi tinggi dan komunikasi yang optimal dalam lingkungan Web 3.0.

II. METODE

Penelitian ini menggunakan pendekatan kuantitatif eksperimental untuk membandingkan kinerja REST API dan gRPC API. Pengujian akan dilakukan dalam sebuah lingkungan terkontrol menggunakan virtual machine (Ubuntu Server 22.04) pada VirtualBox untuk meminimalkan variabel eksternal [9].

A. Analisis Kebutuhan

Adapun spesifikasi dari perangkat keras dan perangkat lunak yang dibutuhkan dan digunakan pada penelitian ini bisa dilihat dibawah ini.

TABEL I
DAFTAR PERANGKAT LUNAK

Nama	*Keterangan*
Visual Studio Code	Software yang akan digunakan untuk membuat sistem
Virtual Box	Software yang akan digunakan untuk virtualisasi sistem
Apache Jmeter	Tools yang digunakan untuk menjalankan pengujian terhadap Throughput dan Received KB/sec
Wireshark	Tools yang digunakan untuk menjalankan pengujian terhadap Response Time
Postman	Tools untuk pengujian fungsi API

TABEL II
SPESIFIKASI PERANGKAT KERAS

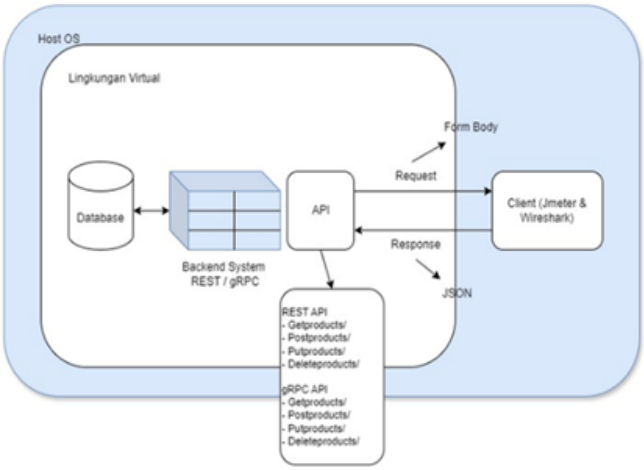
Keterangan	Spesifikasi
Processor	Intel i7 12700H
Graphic Card	NVIDIA GeForce RTX 3050
Storage	SSD 1,5 TB
RAM	16GB DDR 5
Operating System	Windows 11 Home

TABEL III
SPESIFIKASI PERANGKAT KERAS

Keterangan	Spesifikasi
System Name	REST gRPC API
Processor	4 CPU
Storage	40 GB
Base Memory	4 GB
Operating System	Ubuntu (64-bit)
Fungsi	Akan digunakan sebagai server yang akan menjalankan REST API dan gRPC API

B. Desain Perancangan Sistem

Pada tahap ini dilakukan perancangan dan desain terhadap lingkungan sistem yang mencakup database, struktur REST API, struktur gRPC API, serta komponen pendukung lainnya. [16][17] Perancangan ini bertujuan untuk memastikan efisiensi akses data serta kelancaran komunikasi antar layanan. REST API menggunakan format JSON berbasis HTTP, sedangkan gRPC API memanfaatkan Protobuf untuk meningkatkan performa. Selain itu, sistem juga dilengkapi dengan mekanisme pemantauan dan pengumpulan data untuk analisis lebih lanjut. Rancangan sistem yang dibuat dapat dilihat pada gambar 1.



Gambar 1. Desain Rancangan Sistem

Berdasarkan desain rancangan sistem diatas yang dimana terdiri dari client (menggunakan Apache JMeter dan Wireshark) yang mengirim permintaan CRUD ke dua layanan terpisah: REST API berbasis HTTP/1.1 dengan format JSON dan gRPC berbasis HTTP/2 dengan protokol biner Protocol Buffers. Kedua layanan dijalankan pada sebuah server virtual (Ubuntu) yang terhubung ke database untuk operasi data, sementara alat pemantau seperti Wireshark dan JMeter mencatat metrik kinerja (throughput, latensi, response time, transfer data dan payload) dalam lingkungan terkontrol. Desain ini dirancang untuk membandingkan performa kedua protokol secara adil pada kondisi lalu lintas rendah/tinggi dan variasi ukuran data, sekaligus memastikan validitas dari hasil melalui isolasi variabel eksternal. Gambar ini menjadi fondasi visual yang memperjelas metodologi penelitian,

terutama dalam konteks kebutuhan efisiensi Web 3.0 dan aplikasi terdesentralisasi.

C. Analisis Data

Analisis statistik menggunakan metode analisis ANOVA dimana digunakan untuk mengevaluasi dan membandingkan kinerja antara REST API dan gRPC API berdasarkan beberapa metrik utama, yang terdiri dari latensi, response time, throughput, dan efisiensi protokol. Penggunaan analisis data ANOVA memungkinkan pengujian apakah terdapat sebuah perbedaan yang signifikan di antara kedua jenis API dalam berbagai skenario pengujian yang telah ditentukan [18][19][20].

Dalam analisis ini, hipotesis dirumuskan untuk menguji apakah terdapat perbedaan yang signifikan dalam performa antara REST API dan gRPC API ketika diuji dalam berbagai kondisi beban kerja. Pengujian dilakukan dengan mempertimbangkan faktor-faktor seperti jumlah permintaan (request) yang bervariasi dari rendah hingga tinggi, ukuran data yang dikirim dan diterima, serta jenis metode permintaan yang digunakan. Hasil dari pengujian ini diharapkan dapat memberikan pemahaman yang lebih mendalam mengenai efisiensi, kecepatan, dan konsumsi sumber daya dari masing-masing API. Hipotesis dirumuskan sebagai berikut:

- H_0: Tidak ada perbedaan signifikan antara kedua API.
- H_a: Terdapat perbedaan signifikan pada parameter yang diuji.

Pengukuran dilakukan dengan Postman, Apache JMeter, dan Wireshark pada berbagai skenario beban dan ukuran data. Dimana hasil uji ini akan menghasilkan Kesimpulan yang dapat diambil dari hipotesa yang ada.,

D. Pengujian

Pengujian arsitektur REST API dan gRPC API bertujuan untuk membandingkan performa keduanya pada berbagai macam scenario lalu lintas. Pengujian dilakukan pada berbagai scenario menggunakan metode request yang mencangkup GET, POST, PUT, dan DELETE pada kedua secara setara, adapun skenarionya adalah sebagai berikut.

1) Skenario Pengujian GET: Pengujian ini dilakukan untuk menerima data dari web service yang telah dibuat dan dikembalikan ke client berdasarkan data yang diminta untuk output yang diharapkan.
2) Skenario Pengujian POST: Pengujian ini dilakukan untuk mengirim data ke server sesuai dengan kueri yang telah ditentukan kemudian server akan menyimpan sumber data yang telah dibuat dan client akan menerima response apakah data berhasil disimpan atau gagal.
3) Skenario Pengujian PUT: Pengujian ini dilakukan untuk mengirim data ke server sesuai dengan kueri yang telah ditentukan kemudian server akan memperbaharui data sesuai dengan sumber data ke dalam server yang telah dibuat dan client akan menerima response apakah data berhasil diubah atau gagal.
4) Skenario Pengujian Delete: Pengujian ini dilakukan untuk menghapus sumber data yang ada di server dengan mengirim primary key (id) kemudian client menerima response apakah data berhasil dihapus atau gagal.

Skenario diatas akan di gunakan dalam pengujian yang menggunakan aplikasi Apache Jmeter dan Wireshak. [9][21] Adapun detail konfigurasi yang akan digunakan pada pengujian ini di aplikasi Apache Jmeter dimana terdapat dua model konfigurasi yaitu Low Traffic dan High Traffic dan juga konfigurasi dari Wireshark dapat dilihat pada Tabel dibawah ini.

TABEL IV
SKENARIO 1: KONFIGURASI LOW TRAFFIC APLIKASI APACHE JMETER

Number Of Thread	20	Banyaknya user yang melakukan permintaan
Loop Count	1	Pengulangan yang akan dilakukan per user
Ramp-Up Period	1	Total waktu untuk melakukan jumlah dari number threads, jadi setiap user yang melakukan permintaan akan membutuhkan per satu detik
HTTP Cookie Manager	-	Menghapus cookies setiap iterasi yang telah dilakukan

TABEL V
SKENARIO 2 : KONFIGURASI HIGH TRAFFIC APLIKASI APACHE JMETER

Number Of Thread	100	Banyaknya user yang melakukan permintaan
Loop Count	1	Pengulangan yang akan dilakukan per user
Ramp-Up Period	1	Total waktu untuk melakukan jumlah dari number threads, jadi setiap user yang melakukan permintaan akan membutuhkan per satu detik
HTTP Cookie Manager	-	Menghapus cookies setiap iterasi yang telah dilakukan

TABEL VI
KONFIGURASI APLIKASI WIRESHARK

	REST API	**gRPC API**
Filter	HTTP 1	HTTP 2
Connection	Ethernet	Ethernet

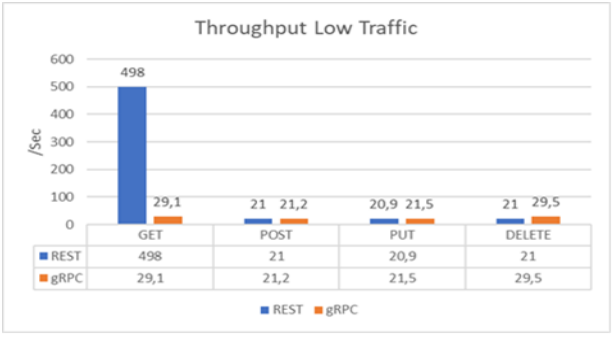
III. HASIL DAN PEMBAHASAN

A. Perhitungan Dengan Apache Jmeter Skenario 1 (Low Traffic)

1) Throughput

Berikut adalah hasil pengujian menggunakan apache jmeter untuk skenario throughput low traffic pada arsitekur REST API dan gRPC API dalam bentuk grafik dapat dilihat di bawah ini pada gambar 2. Pada pengujian throughput low traffic dimana mengukur seberapa banyak request yang dapat diproses dalam satu detik. Pada kondisi low traffic, REST API diketahui memiliki throughput yang jauh lebih tinggi dibandingkan dengan gRPC, terutama pada metode GET, yang mencapai 498 request per detik, sementara gRPC hanya

29,1 request per detik. Namun, pada metode POST, PUT, dan DELETE, perbedaan throughput antara keduanya tidak terlalu besar, menunjukkan bahwa dalam kondisi beban rendah, REST lebih optimal untuk metode GET, sedangkan gRPC tetap cukup stabil pada metode lainnya.



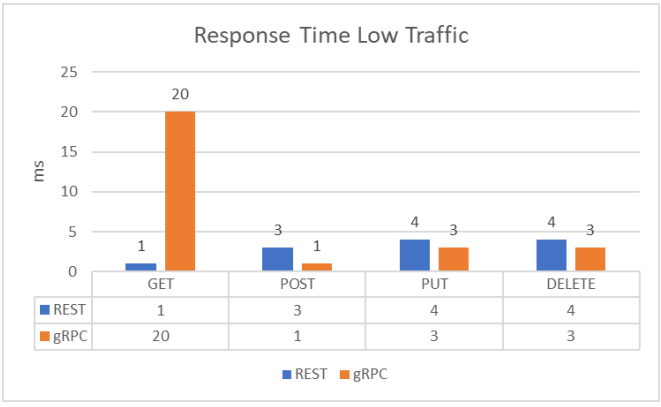
Gambar **2.** Hasil Pengujian Throughput Low Traffic

TABEL VII
HASIL PENGUJIAN THROUGHPUT LOW TRAFFIC

	GET	**POST**	**PUT**	**Delete**
REST	498 req/sec	21 req/sec	20,9 req/sec	21 req/sec
GRPC	29,1 req/sec	21,2 req/sec	21,5 req/sec	29,5 req/sec

2) Response Time

Berikut adalah hasil pengujian menggunakan apache jmeter untuk skenario Response Time low traffic pada REST API dan gRPC API dalam bentuk grafik dapat dilihat di bawah ini pada gambar 3.



Gambar 3. Hasil Pengujian Skenario Response Time Low Traffic

Pada pengujian response time dalam kondisi low traffic, dilakukan pengukuran terhadap waktu yang dibutuhkan server untuk memberikan respons terhadap setiap request yang diterima. Hasil pengujian menunjukkan bahwa pada metode GET, REST API memiliki response time yang lebih cepat, hanya 1 ms, dibandingkan dengan gRPC yang mencapai 20 ms.

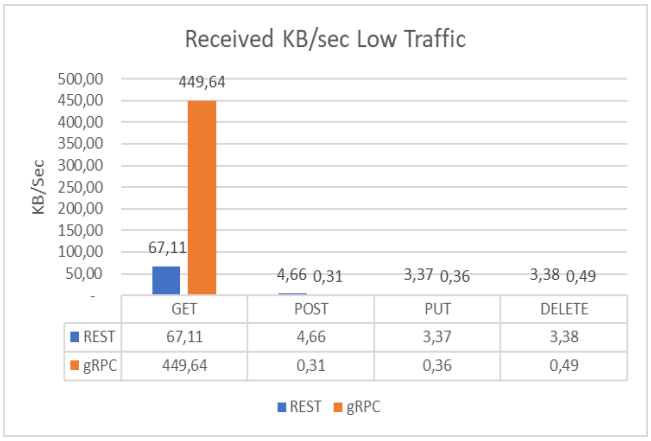
TABEL VIII
HASIL PENGUJIAN RESPONSE TIME LOW TRAFFIC

	GET	POST	PUT	Delete
REST	1 ms	3 ms	4 ms	4 ms
GRPC	20 ms	1 ms	3 ms	3 ms

Hal ini menunjukkan bahwa dalam kondisi permintaan sederhana seperti GET, REST API lebih efisien dalam menangani request dengan overhead yang lebih kecil. Namun, pada metode POST, gRPC memiliki keunggulan dengan response time sebesar 1 ms, sedangkan REST API membutuhkan 3 ms. Perbedaan ini dapat disebabkan oleh optimasi komunikasi biner pada gRPC yang lebih efisien dalam menangani pengiriman data dalam format serialisasi Protobuf. Untuk metode PUT dan DELETE, hasil pengujian menunjukkan bahwa kedua API memiliki response time yang hampir sama dengan selisih yang tidak terlalu signifikan. Hal ini menunjukkan bahwa dalam kondisi low traffic, efektivitas masing-masing API dapat bervariasi tergantung pada jenis operasi yang dilakukan.

3) Received Data

Berikut adalah hasil pengujian menggunakan apache jmeter untuk skenario skenario Received Data low traffic pada arsitekur REST API dan gRPC API dalam bentuk grafik dapat dilihat pada gambar 4.



Gambar 4. Hasil Pengujian Skenario Received Data Low Traffic

TABEL IX
HASIL PENGUJIAN RECEIVED DATA LOW TRAFFIC

	GET	POST	PUT	Delete
REST	67,11 KB/Sec	4,66 KB/Sec	3,37 KB/Sec	3,38 KB/Sec
GRPC	449,64 KB/Sec	0,31 KB/Sec	0,36 KB/Sec	0,49 KB/Sec

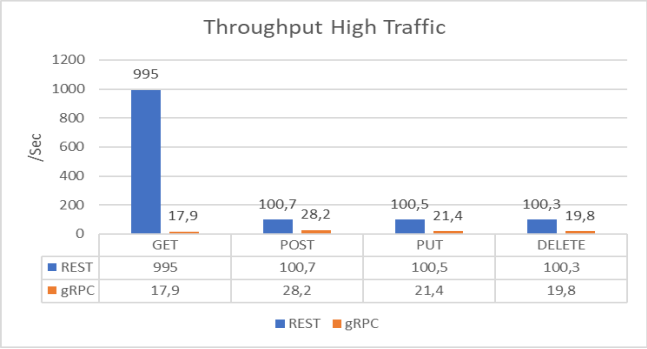
Pada pengujian Received Data, dilakukan pengukuran terhadap jumlah data yang diterima oleh server dalam satu detik untuk masing-masing metode HTTP dan RPC. Pengujian ini bertujuan untuk mengevaluasi efisiensi transfer data antara REST API dan gRPC API dalam kondisi low

traffic. Hasil pengujian menunjukkan bahwa pada metode GET, gRPC memiliki keunggulan signifikan dengan data yang diterima sebesar 449,64 KB/s, jauh lebih tinggi dibandingkan REST API yang hanya mencapai 67,11 KB/s. Perbedaan ini disebabkan oleh penggunaan format biner dalam gRPC yang lebih padat dan efisien dalam mengelola komunikasi data, sehingga dapat mengirimkan lebih banyak informasi dalam waktu yang sama. Namun, pada metode POST, PUT, dan DELETE, REST API menerima data yang lebih besar dibandingkan gRPC. Hal ini menunjukkan bahwa meskipun gRPC unggul dalam efisiensi data untuk permintaan GET, REST API lebih seimbang dalam menangani berbagai metode lainnya. Dengan demikian, pemilihan API yang optimal tergantung pada jenis operasi yang lebih dominan dalam suatu aplikasi yang dikembangkan.

B. Perhitungan Dengan Apache Jmeter Skenario 2 (High Traffic)

1) Throughput

Berikut adalah hasil pengujian menggunakan apache jmeter untuk skenario Throughput high traffic pada arsitekur REST API dan gRPC API dalam bentuk grafik gambar 5.



Gambar 5. Hasil Pengujian Throughput High Traffic

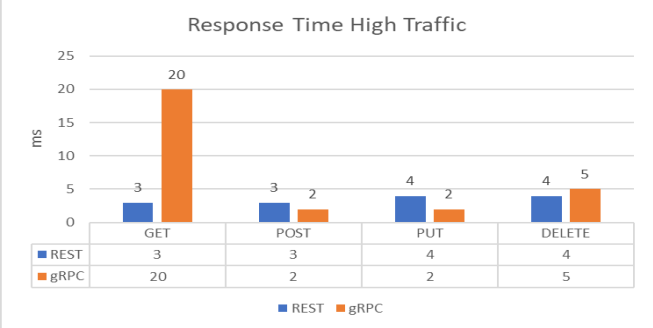
TABEL X
HASIL PENGUJIAN THROUGHPUT HIGH TRAFFIC

	GET	POST	PUT	Delete
REST	995 req/sec	100,7 req/sec	100,5 req/sec	100,3 req/sec
GRPC	17,9 req/sec	28,2 req/sec	21,4 req/sec	19,8 req/sec

Pada pengujian throughput high traffic, perbedaan throughput antara REST dan gRPC semakin terlihat. Pada metode GET, REST API masih unggul dengan 995 request per detik, sedangkan gRPC mengalami penurunan hingga 17,9 request per detik. Untuk metode POST, PUT, dan DELETE, REST tetap memiliki throughput yang lebih tinggi dibandingkan gRPC, meskipun selisihnya tidak sebesar pada metode GET. Ini menunjukkan bahwa REST lebih mampu menangani lonjakan request dalam jumlah besar dibandingkan gRPC

2) Response Time

Berikut adalah hasil pengujian menggunakan apache jmeter untuk skenario Response Time high traffic pada arsitekur REST API dan gRPC API dalam bentuk grafik dapat dilihat di bawah ini pada Gambar 6



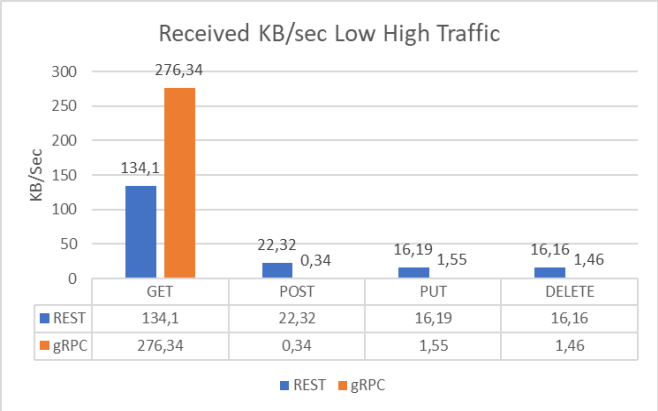
Gambar 6. Hasil Pengujian Skenario Response Time High Traffic

TABEL XI
HASIL PENGUJIAN SKENARIO RESPONSE TIME HIGH TRAFFIC

	GET	POST	PUT	Delete
REST	3 ms	3 ms	4 ms	4 ms
GRPC	20 ms	2 ms	2 ms	5 ms

Pada pengujian response time high traffic, REST API masih mampu mempertahankan response time yang cukup stabil di kisaran 3–4 ms untuk semua metode. Sebaliknya, gRPC masih mengalami response time yang lebih tinggi pada metode GET, yaitu 20 ms, namun lebih cepat pada metode POST dan PUT, masing-masing 2 ms, dibandingkan REST. Hal ini menunjukkan bahwa dalam kondisi high traffic, gRPC diketahui memiliki response time yang lebih baik pada metode POST dan PUT, tetapi masih lebih lambat pada metode GET dibandingkan REST.

3) Received Data



Gambar 7. Hasil Pengujian Skenario Received Data High Traffic

Hasil pengujian menggunakan apache jmeter untuk skenario Received Data high traffic pada arsitekur REST API dan gRPC API dalam bentuk grafik dapat dilihat pada Gambar 7.

TABEL XII
HASIL PENGUJIAN SKENARIO RECEIVED DATA HIGH TRAFFIC

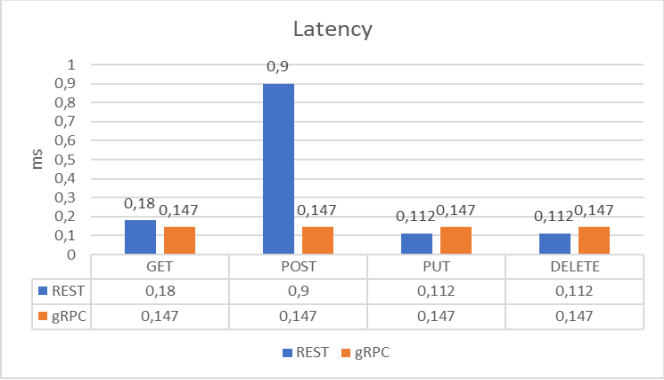
	GET	POST	PUT	Delete
REST	134,1 KB/Sec	22,32 KB/Sec	16,19 KB/Sec	16,16 KB/Sec
GRPC	276,34 KB/Sec	0,34 KB/Sec	1,55 KB/Sec	1,46 KB/Sec

Pada pengujian received data high traffic jumlah data yang diterima REST API meningkat pada metode POST, PUT, dan DELETE, dengan metode POST mencapai 22,32 KB/s. Sebaliknya, gRPC tetap menerima data yang lebih besar pada metode GET, yaitu 276,34 KB/s, namun jumlah data yang diterima pada metode lainnya lebih rendah dibandingkan REST. Ini menunjukkan bahwa REST API mampu menangani distribusi data dengan lebih merata dalam kondisi high traffic, sedangkan gRPC lebih terfokus pada metode GET.

C. *Perhitungan Dengan Wireshark*

1) Latency

Berikut adalah hasil data dari pengujian Latency dengan menggunakan Wireshark pada arsitekur REST API dan gRPC API dalam bentuk grafik dapat dilihat pada Gambar 8.



Gambar 8. Hasil Pengujian Skenario Latency Wireshark

TABEL XIII
HASIL PENGUJIAN SKENARIO LATENCY WIRESHARK

	GET	POST	PUT	Delete
REST	0,18	0,9	0,112	0,112
GRPC	0,147	0,147	0,147	0,147

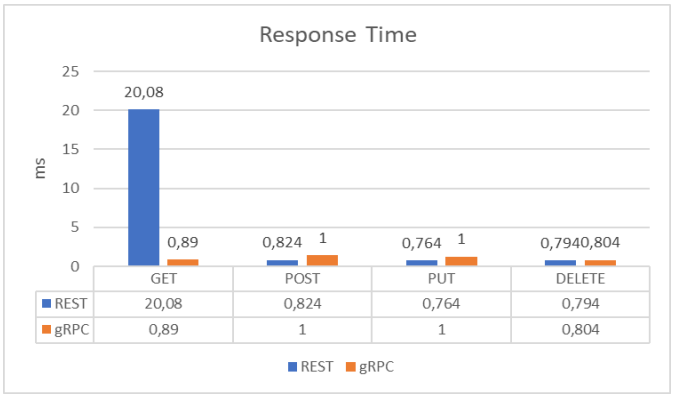
Pengujian latency menggunakan Wireshark dilakukan untuk mengukur waktu tunda dalam komunikasi antara klien dan server pada REST API dan gRPC API. Hasil pengujian menunjukkan bahwa REST API memiliki variasi latency tergantung pada metode yang digunakan. Metode POST mencatat latency tertinggi sebesar 0,9 ms, sedangkan metode GET memiliki latency yang lebih rendah, yaitu 0,18 ms. Perbedaan ini disebabkan oleh kompleksitas pemrosesan setiap metode, di mana POST membutuhkan lebih banyak langkah dalam menangani data yang dikirimkan dibandingkan dengan GET, yang umumnya hanya melakukan permintaan data tanpa modifikasi.

Di sisi lain, gRPC menunjukkan performa yang lebih konsisten dengan latency yang relatif stabil di semua metode, yaitu sekitar 0,147 ms. Konsistensi ini menunjukkan bahwa gRPC lebih efisien dalam mengelola komunikasi antara klien dan server tanpa mengalami fluktuasi yang signifikan. Hal ini disebabkan oleh penggunaan protokol biner yang lebih ringan serta mekanisme komunikasi berbasis HTTP/2 yang lebih optimal dibandingkan REST API yang masih menggunakan HTTP/1.1 dalam implementasinya.

Dengan stabilitas latency yang lebih baik, gRPC menjadi pilihan yang lebih optimal untuk aplikasi yang membutuhkan respons cepat dan konsisten, seperti sistem real-time atau layanan dengan beban lalu lintas tinggi..

2) Response Time

Berikut adalah hasil data dari pengujian Latency dengan menggunakan Wireshark pada arsitekur REST API dan gRPC API dalam bentuk grafik dapat dilihat pada Gambar 9.



Gambar 9. Hasil Pengujian Skenario Response Time Wireshark

TABEL XIV
HASIL PENGUJIAN SKENARIO RESPONSE TIME WIRESHARK

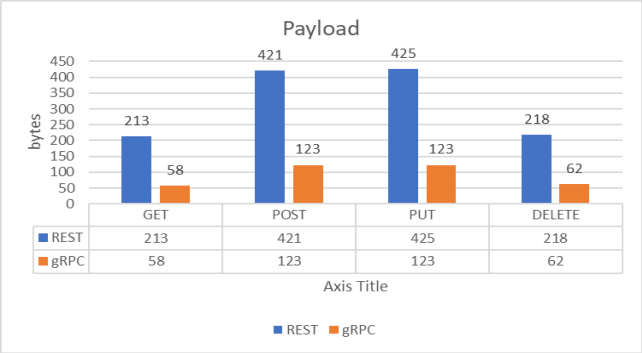
	GET	POST	PUT	Delete
REST	20,08	0,824	0,764	0,794
GRPC	0,89	1	1	0,804

Pada pengujian ini dimana response time akan mengukur berapa lama waktu yang dibutuhkan dari server untuk memproses request yang ada sebelum mengirimkan respons kepada user. Diketahui bahwa pada metode GET dalam REST API memiliki response time yang jauh lebih tinggi, yaitu 20,08 ms, sedangkan gRPC hanya 0,89 ms, Dimana menunjukkan bahwa gRPC jauh lebih cepat dalam menangani metode GET. Namun, pada metode POST, PUT, dan DELETE, diketahui bahwa perbedaan yang didapatkan tidak terlalu mencolok diantara keduanya. REST API memiliki response time 0,824 ms untuk POST, 0,764 ms untuk PUT, dan 0,794 ms untuk DELETE. Sementara itu, gRPC memiliki response time 1 ms untuk POST dan PUT, serta 0,804 ms untuk DELETE. Hal ini menunjukkan bahwa gRPC lebih

unggul dalam metode GET, sedangkan pada beberapa metode lainnya, performa gRPC diketahui relatif sebanding dengan REST API.

3) Payload

Berikut adalah hasil data dari pengujian payload dengan menggunakan Wireshark pada arsitekur REST API dan gRPC API dalam bentuk grafik dapat dilihat pada Gambar 10.



Gambar 10. Hasil Pengujian Skenario Payload Wireshark

TABEL XV
HASIL PENGUJIAN SKENARIO PAYLOAD WIRESHARK

	GET	POST	PUT	Delete
REST	0,18 bytes	0,9 bytes	0,112 bytes	0,112 bytes
GRPC	0,147 bytes	0,147 bytes	0,147 bytes	0,147 bytes

Berdasarkan pengujian menggunakan Wireshark, gRPC secara konsisten menghasilkan ukuran payload yang lebih kecil dibandingkan REST pada semua metode HTTP (GET, POST, PUT, DELETE), dengan perbedaan mencapai 2–4× lebih efisien. Pada metode GET, payload gRPC (58 byte) 3,7× lebih ringkas daripada REST (213 byte), sedangkan pada POST dan PUT, gRPC (123 byte) menghemat 70% bandwidth dibanding REST (421–425 byte). Efisiensi ini disebabkan penggunaan Protobuf pada gRPC yang mengoptimalkan serialisasi data biner, mengurangi overhead struktural seperti pada format teks (JSON/XML) di REST. Selain itu, gRPC menunjukkan konsistensi ukuran payload untuk operasi POST dan PUT (123 byte), sementara REST memiliki variasi yang lebih besar (421–425 byte), mengindikasikan fleksibilitas yang berpotensi menambah kompleksitas data. Pada metode DELETE, gRPC tetap unggul (62 byte vs. 218 byte pada REST), memperkuat keunggulannya dalam skenario low-bandwidth atau aplikasi real-time. Hasil ini merekomendasikan gRPC untuk sistem yang memprioritaskan efisiensi jaringan, sementara REST tetap relevan untuk kasus yang memerlukan kemudahan debugging atau kompatibilitas dengan alat eksternal. Kombinasi kedua protokol dapat dipertimbangkan berdasarkan kebutuhan operasional dominan dalam sistem.

D. Hasil Uji Anova

Setelah data pengujian terkumpul, analisis dilakukan untuk mengevaluasi apakah terdapat perbedaan signifikan antara performa REST API dan gRPC API. Teknik analisis data yang digunakan yaitu teknik analisa data ANOVA dimana akan membandingkan rata-rata metrik kinerja kedua API pada berbagai kondisi pengujian. ANOVA memungkinkan identifikasi perbedaan yang signifikan secara statistik antara kelompok data, baik dalam kondisi beban rendah maupun tinggi, serta pada variasi ukuran data, telah di dapatkan data seperti gambar dibawah ini

```
Category: Throughput | Metric: Throughput Low Traffic
 F-value: 0.9279005957427268, P-value: 0.37260898293184214
No significant difference found in Throughput metric.

Category: Throughput | Metric: Throughput High Traffic
 F-value: 1.8272243416128546, P-value: 0.22518809770702
No significant difference found in Throughput metric.

Category: Response Time | Metric: Response Time Low Traffic
 F-value: 0.6951596292481977, P-value: 0.4363429579183199
No significant difference found in Response Time metric.

Category: Response Time | Metric: Response Time High Traffic
 F-value: 0.7541899441340782, P-value: 0.41852998415789394
No significant difference found in Response Time metric.

Category: Latency | Metric: Latency
 F-value: 0.8691363159607942, P-value: 0.3871717394102951
No significant difference found in Latency metric.

Category: Payload | Metric: Payload
 F-value: 0.8691363159607942, P-value: 0.3871717394102951
No significant difference found in Payload metric.
```

Gambar 11. Hasil Pengujian Analisis Data ANOVA

Berdasarkan hasil uji ANOVA, seluruh metrik kinerja (throughput, response time, dan latency) tidak menunjukkan perbedaan yang signifikan antara REST API dan gRPC API, baik pada kondisi lalu lintas rendah maupun tinggi dimana didapatkan hasil yaitu Hipotesis Nol (H_0): diterima untuk semua kategori metrik, yang berarti tidak ada bukti statistik bahwa REST API dan gRPC API memiliki performa yang berbeda secara signifikan dalam pengujian ini. Hasil ini menunjukkan bahwa dalam skenario pengujian tertentu, baik REST API maupun gRPC API memberikan performa yang sebanding untuk metrik-metrik yang diuji. Faktor-faktor lain, seperti kompleksitas data atau kebutuhan spesifik protokol, mungkin lebih relevan dalam menentukan pilihan teknologi API.

IV. KESIMPULAN

Penelitian ini menunjukkan bahwa tidak terdapat perbedaan yang signifikan secara statistik ($p > 0.05$) antara REST API dan gRPC API dalam metrik yang diuji, yaitu throughput, latency, dan response time. Oleh karena itu, tidak dapat disimpulkan bahwa salah satu API lebih unggul secara objektif dalam pengujian ini. Namun, analisis deskriptif mengungkap kecenderungan performa yang berbeda antara kedua protokol dalam kondisi tertentu.

REST API menunjukkan kecenderungan memiliki throughput lebih tinggi dalam kondisi low traffic serta response time yang lebih stabil pada permintaan GET. Hal ini disebabkan oleh overhead komunikasi yang lebih rendah dalam permintaan sederhana berbasis HTTP/1.1 dan format

teks JSON. Sebaliknya, gRPC lebih efisien dalam menangani transfer data besar, berkat penggunaan Protocol Buffers (Protobuf) dan HTTP/2, yang memungkinkan komunikasi lebih cepat dengan multiplexing dalam skenario tertentu.

Karena tidak ada perbedaan signifikan secara statistik, pemilihan antara REST API dan gRPC API sebaiknya mempertimbangkan aspek praktis. REST API lebih cocok untuk aplikasi web yang membutuhkan response cepat dan kompatibilitas luas dengan berbagai platform. Di sisi lain, gRPC lebih sesuai untuk komunikasi antar-microservices, sistem real-time, serta aplikasi yang memerlukan transfer data dalam jumlah besar dengan efisiensi tinggi.

Namun, perlu dicatat bahwa REST API dalam penelitian ini menggunakan HTTP/1.1, sedangkan gRPC menggunakan HTTP/2. Keunggulan gRPC dalam beberapa aspek dapat dipengaruhi oleh fitur HTTP/2 yang lebih optimal dalam menangani banyak permintaan secara paralel dibandingkan HTTP/1.1 yang masih menggunakan model request-response tradisional. Oleh karena itu, hasil penelitian ini tidak hanya merefleksikan perbedaan antara REST API dan gRPC, tetapi juga dampak dari perbedaan protokol HTTP yang digunakan.

Sebagai rekomendasi, penelitian lanjutan disarankan untuk menguji REST API dengan HTTP/2 agar perbandingan menjadi lebih adil. Dengan menggunakan HTTP/2 pada REST API, akan lebih mudah untuk menilai apakah keunggulan gRPC berasal dari model RPC dan penggunaan Protobuf, atau lebih dipengaruhi oleh keunggulan HTTP/2 dibandingkan HTTP/1.1. Selain itu, analisis effect size atau confidence interval dapat dieksplorasi lebih lanjut untuk memahami kecenderungan performa masing-masing API.

Temuan ini menegaskan bahwa REST API dan gRPC API bukanlah pilihan yang harus bersifat eksklusif dalam implementasi sistem, melainkan dapat digunakan secara komplementer untuk memanfaatkan keunggulan masing-masing. REST API dapat menjadi pilihan utama untuk komunikasi eksternal yang memerlukan fleksibilitas dan dukungan luas, sementara gRPC lebih efektif untuk komunikasi internal dalam sistem berbasis microservices atau aplikasi yang memerlukan latensi rendah. Dengan demikian, integrasi kedua protokol dapat menjadi strategi yang efektif untuk membangun arsitektur yang responsif, efisien, dan siap menghadapi tantangan desentralisasi di masa depan.

DAFTAR PUSTAKA

[1] H. F. Herdiyatmoko, "Back-End System Design Based on Rest Api," *J. Tek. Inf. dan Komput.*, vol. 5, no. 1, p. 123, 2022, doi: 10.37600/tekinkom.v5i1.401.

[2] P. L. Azzahra and A. Salam, "Desentralisasi dalam Perspektif Technology Trend terhadap Blockchain dan Web3," *J. Digit. Technol. Trend*, vol. 2, no. 1, pp. 34–42, 2023, doi: 10.56347/jdtt.v2i1.149.

[3] E. D. Demircioğlu and O. Kalipsiz, "API Message-Driven Regression Testing Framework," *Electron.*, vol. 11, no. 17, 2022, doi: 10.3390/electronics11172671.

[4] B. Marii and I. Zholubak, "ADVANCES IN CYBER-PHYSICAL SYSTEMS," vol. 7, 2022.

[5] M. I. Yanuardi, A. Aminudin, and M. Faiqurahman, "Analisis Perbandingan Efektivitas Arsitektur Restful dan Arsitektur Grpc pada Implementasi Web Service," *J. Edukasi dan Penelit. Inform.*, vol. 10, no. 2, p. 333, 2024, doi: 10.26418/jp.v10i2.80845.

[6] B. Shafabakhsh, R. Lagerström, and S. Hacks, "Evaluating the impact of inter process communication in microservice architectures," *CEUR Workshop Proc.*, vol. 2767, no. QuASoQ, pp. 55–63, 2020.

[7] K. S. M. Hari and R. Sharma, "Comparative Study of Orchestration Using Grpc Api and Rest Api in Server Creation Time: an Openstack Case," *Int. J. Comput. Networks Commun.*, vol. 16, no. 1, pp. 87–104, 2024, doi: 10.5121/ijcnc.2024.16106.

[8] F. Hanif, I. Ahmad, D. Darwis, I. Lazuardi Putra, and M. Fauzan Ramadhani, "Analisa Perbandingan Metode Graphql Api Dan Rest Api Dengan Menggunakan Asp.Net Core Web Api Framework," *J. Telemat. Inf.*, vol. 3, no. 2, pp. 2774–5384, 2022.

[9] A. Niklasson and V. Werèlius, "RESTful API vs. GraphQL a CRUD performance comparison," p. 40, 2023.

[10] M. Johansson and O. Isabella, "Comparative Study of REST and gRPC for Microservices in Established Software Architectures," 2023, [Online]. Available: https://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-195563

[11] E. Nilsson and D. Demir, "Performance comparison of REST vs GraphQL in different web environments : Node.js and Python," p. 29, 2023.

[12] M. Śliwa and B. Pańczyk, "Performance comparison of programming interfaces on the example of REST API, GraphQL and gRPC," *J. Comput. Sci. Inst.*, vol. 21, no. October, pp. 356–361, 2021, doi: 10.35784/jcsi.2744.

[13] P. Diantono, A. Susanto, A. R. Supriyono, and D. N. Prasetyanti, "Perbandingan Kinerja Antara Gatling dan Apache JMeter pada Uji Beban RESTful API," vol. 15, no. 01, pp. 211–215, 2024, doi: 10.35970/infotekmesin.v15i1.2176.

[14] M. Zaenul Hasan Basri and M. Zaenul Hasan, "Analysis and Security Testing for GRPC," no. January, pp. 2020–2023, 2024, [Online]. Available: https://www.researchgate.net/publication/377360768

[15] Rangga Gelar Guntara and V. Azkarin, "Implementasi dan Pengujian REST API Sistem Reservasi Ruang Rapat dengan Metode Black Box Testing," *J. Minfo Polgan*, vol. 12, no. 1, pp. 1229–1238, 2023, doi: 10.33395/jmp.v12i1.12691.

[16] B. Sutara and S. Gunawan, "Comparative Analysis of Rest Api Performance Between Express.Js Framework and Hapi.Js Using Apache Jmeter," *J. Ris. Tek. Inform.*, vol. 1, no. 1, pp. 19–26, 2024.

[17] B. A. Thango, "Application of the Analysis of Variance (ANOVA) in the Interpretation of Power Transformer Faults," *Energies*, vol. 15, no. 19, 2022, doi: 10.3390/en15197224.

[18] V. Mandailina, D. Pramita, Syaharyddin, Ibrahim, Nurmiwati, and Abdillah, "Uji hipotesis menggunakan software jasp sebagai upaya peningkatan kemampuan teknik analisa data pada riset mahasiswa," *J. Character Educ. Soc.*, vol. 5, no. 2, pp. 512–519, 2022, [Online]. Available: http://journal.ummat.ac.id/index.php/JCEShttps://doi.org/10.31764/jces.v5i2.6109https://doi.org/10.31764/jces.v3i1.XXX

[19] G. Jain and Anubha, "Application of SNORT and Wireshark in Network Traffic Analysis," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 1119, no. 1, p. 012007, 2021, doi: 10.1088/1757-899x/1119/1/012007.

[20] Yeni Setiani, Nabila Rachmah, and Indra Purnama, "Visualisasi Data Malnutrisi Anak Di Asia Menggunakan Looker Studio Serta Analisis Data Dengan Metode ANOVA," *J. Ilm. Sist. Inf. dan Ilmu Komput.*, vol. 3, no. 3, pp. 188–212, 2023, doi: 10.55606/juisik.v3i3.701.