P-ISSN : 2549-7480
E-ISSN : 2549-6301

JURNAL ILMIAH INFORMATIKA
E-ISSN : 2549-6301
ISSN : 2549-7480
Faculty of Sains and Technology, Ibrahimy University
S4-Accredited : 105/E/KPT/2022
Journal page is available to : https://journal.ibrahimy.ac.id/index.php/JIMI

ANALISIS PERBANDINGAN KINERJA APLIKASI LARAVEL DENGAN DAN TANPA REDIS CACHING DI LINGKUNGAN DENGAN BEBAN TINGGI

M. Bahril Ilmi[1)], Hengki Dwiyan Hermawan[2)]

[1] *Teknologi Rekayasa Multimedia, Politeknik Hasnur*
[2] *Bisnis Digital, Politeknik Hasnur*

email: [1] mbahrililmi2@gmail.com, [2] hengki.dwiyan@gmail.com

ARTICLE INFO

Article History:
Recieved : 16 Maret 2026
Accepted : 28 Mei 2026
Published : 28 Juni 2026

Keywords:

Laravel,
Redis,
caching,
pengujian beban,
waktu respons,
throughput,
kinerja aplikasi web.

IEEE style in citing this article:
M. B. Ilmi., H. D. Hermawan, "Analisis Perbandingan Kinerja Aplikasi Laravel dengan dan Tanpa Redis Caching di Lingkungan dengan Beban Tinggi", *jurnal.ilmiah.informatika*, vol. 11, no. 1, pp. 117-xx, Juni. 2026.

ABSTRACT

Aplikasi web berbasis Laravel rentan terhadap penurunan kinerja yang signifikan ketika lalu lintas pengguna meningkat, terutama akibat eksekusi berulang dari kueri basis data yang identik yang memberikan tekanan berlebihan pada MySQL. Studi ini menyelidiki sejauh mana Redis, yang digunakan sebagai lapisan caching dalam memori, mampu mengimbangi tekanan tersebut sekaligus mengukur peningkatan kinerja yang dapat dicapai dibandingkan konfigurasi dasar tanpa caching.

Uji beban dilakukan menggunakan k6 terhadap lingkungan terkontainerisasi berbasis Docker guna memastikan kondisi pengujian yang identik dan sepenuhnya dapat direproduksi. Pengujian mencakup empat skenario pengguna bersamaan, yakni 50, 100, 200, dan 500 pengguna, masing-masing dipertahankan selama 60 detik. Tumpukan aplikasi terdiri dari Laravel 11.x pada PHP 8.3, MySQL 8.0 sebagai penyimpanan data utama, dan Redis 7.2 sebagai lapisan caching. Strategi caching yang diterapkan meliputi caching kueri, caching rute, dan caching konfigurasi.

Metrik yang diamati mencakup waktu respons pada persentil P50, P95, dan P99, throughput dalam permintaan per detik, penggunaan CPU dan memori sisi server, serta tingkat kesalahan. Hasil penelitian menunjukkan bahwa Redis mampu mengurangi waktu respons rata-rata sebesar 92% dan meningkatkan throughput hingga 947% pada skenario 500 pengguna bersamaan, sementara tingkat kesalahan tetap berada di bawah 1% pada seluruh skenario pengujian. Kesenjangan kinerja antara konfigurasi dengan dan tanpa caching melebar secara konsisten seiring bertambahnya beban, dengan perbedaan paling mencolok mulai terlihat pada skenario 100 pengguna ke atas.

Temuan ini menegaskan bahwa Redis merupakan solusi caching yang sangat efektif untuk meningkatkan skalabilitas aplikasi Laravel dan sangat direkomendasikan untuk setiap penerapan

P-ISSN : 2549-7480
M. B. Ilmi dkk/ JIMI 11 (1) pp. 117-130 E-ISSN : 2549-6301

yang melayani lebih dari 100 pengguna secara bersamaan. Penelitian selanjutnya dapat menyelidiki perilaku Redis di bawah kondisi beban yang lebih ekstrem, serta mengkaji strategi invalidasi cache yang bersifat adaptif untuk menangani data dengan frekuensi penulisan tinggi.

1. PENDAHULUAN

Laravel berada di puncak lanskap kerangka kerja PHP, posisi yang dikonfirmasi oleh Survei Pengembang Stack Overflow 2023, yang menempatkannya sebagai kerangka kerja PHP yang paling banyak digunakan di antara pengembang profesional di seluruh dunia [1]. Posisi tersebut mencerminkan lebih dari sekadar preferensi komunitas. Hal ini didukung oleh ekosistem yang matang, dokumentasi yang lengkap, dan arsitektur MVC yang dapat diskalakan dari proyek kecil hingga sistem tingkat perusahaan [2]. Ribuan tim pengembang telah membangun layanan digital inti mereka di atas Laravel, memperlakukannya sebagai platform yang andal untuk segala hal mulai dari alat manajemen internal hingga backend e-commerce bervolume tinggi.

Namun, adopsi yang meluas tersebut mengungkap masalah berulang yang jarang muncul sampai sistem tersebut beroperasi: kinerja di bawah beban. Setiap permintaan masuk melewati pipeline perutean, pemeriksaan middleware, logika pengontrol, dan resolusi model yang hampir selalu berakhir dengan satu atau lebih kueri basis data. Pada lalu lintas rendah, overhead ini tidak terlihat. Skalakan hingga ratusan atau ribuan pengguna simultan dan MySQL menjadi hambatan nyata, mendorong waktu respons ke tingkat yang tidak dapat diterima, mengurangi throughput, dan dalam kasus yang lebih serius mendorong tingkat kesalahan melewati ambang batas yang dapat diabaikan oleh tim [3]. Pola ini sudah umum, organisasi baru menyadarinya setelah penerapan, ketika pengembalian (rollback) menjadi mahal.

Redis (Remote Dictionary Server) dibangun untuk mengatasi masalah seperti ini. Sebagai penyimpanan key-value dalam memori sumber terbuka yang dirancang untuk kecepatan, ia mampu menangani jutaan operasi per detik dengan latensi yang secara konsisten berada di bawah satu milidetik [4]. Arsitekturnya, yaitu loop peristiwa berutas tunggal dengan multiplexing I/O melalui epoll dan kqueue, dikombinasikan dengan struktur data yang dirancang khusus, telah diteliti secara mendalam dalam konteks debugging hambatan kinerja di lingkungan microservice, di mana ditemukan bahwa ia mengungguli penyimpanan berbasis disk dalam hal latensi dengan urutan besaran [5]. Ditempatkan di antara aplikasi dan basis data, Redis mencegat permintaan baca berulang dan menyelesaikannya langsung dari memori, menghilangkan perjalanan bolak-balik SQL sepenuhnya. Laravel hadir dengan dukungan Redis asli melalui Predis dan PhpRedis, yang berarti integrasi hanya membutuhkan biaya konfigurasi minimal untuk sebagian besar proyek.

Pekerjaan sebelumnya telah menetapkan bahwa caching secara signifikan meningkatkan kinerja aplikasi web di berbagai model penerapan. Satu studi menemukan pengurangan waktu

DOI : https://doi.org/10.35316/jimi.v11i1.117-130

respons yang terukur ketika caching diterapkan pada konten yang sering diminta pada infrastruktur cloud [6]. Studi lain menunjukkan bahwa caching berbasis Redis secara substansial mengurangi latensi dalam penerapan serverless, memperkuat argumen untuk Redis sebagai komponen caching yang andal terlepas dari paradigma arsitektur [7]. Studi ketiga mengevaluasi aplikasi Laravel yang dikontainerisasi Docker dalam konteks perawatan kesehatan dan menemukan bahwa overhead kontainerisasi cukup kecil sehingga hasil uji beban dari lingkungan Docker secara andal mencerminkan hasil arsitektur dunia nyata [8].

Benchmarking kerangka kerja PHP komparatif juga menghasilkan temuan yang relevan. Perbandingan langsung antara Laravel dan CodeIgniter mengungkapkan perbedaan kinerja yang bergantung pada beban kerja di mana Laravel, yang dikonfigurasi dengan caching, menghasilkan latensi yang lebih rendah dalam skenario beban tertentu [9]. Studi terpisah yang membandingkan Redis dengan Memcached di seluruh pengaturan cloud-native yang dikontainerisasi menyimpulkan bahwa Redis secara konsisten memberikan latensi yang lebih rendah dan throughput yang lebih tinggi di bawah setiap kondisi beban yang diuji [10]. Namun, apa yang tidak dibahas dalam karya ini adalah pertanyaan yang lebih spesifik: seberapa besar peningkatan kinerja aplikasi Laravel ketika bertindak sebagai lapisan caching, dan di bawah beban konkuren yang diskalakan secara sistematis?

Pertanyaan itu masih belum terjawab dalam literatur saat ini [11]. Sebagian besar studi yang relevan adalah deskripsi arsitektur atau terbatas pada jumlah pengguna yang terlalu rendah untuk mencerminkan tekanan produksi. Studi ini mengisi celah tersebut melalui eksperimen benchmarking terkontrol: aplikasi Laravel 11.x diuji dalam dua konfigurasi dengan caching Redis aktif dan tanpa menggunakan k6 [12] pada infrastruktur berbasis Docker, di empat tingkat beban (50, 100, 200, dan 500 pengguna konkuren). Metrik yang dicatat meliputi waktu respons pada P50, P95, dan P99, throughput, konsumsi CPU dan memori, dan tingkat kesalahan. Selain perbandingan agregat, eksperimen ini juga menunjukkan ambang batas beban di mana Redis mulai membuat perbedaan yang paling jelas, data yang dapat digunakan praktisi ketika memutuskan apakah dan pada skala berapa Redis layak ditambahkan ke penerapan Laravel [13].

Bagian selanjutnya dari makalah ini adalah sebagai berikut: Bagian 2 meninjau literatur tentang Laravel, Redis, strategi caching, metodologi pengujian beban, dan studi benchmarking sebelumnya; Bagian 3 merinci desain eksperimen dan metodologi pengujian; Bagian 4 menyajikan dan membahas hasil yang diukur; dan Bagian 5 diakhiri dengan kesimpulan utama dan arahan untuk penelitian masa depan.

2. TINJAUAN PUSTAKA

Dalam lanskap pengembangan web PHP, Laravel telah memantapkan dirinya sebagai kerangka kerja pilihan bagi pengembang profesional, karena arsitektur MVC yang terstruktur dengan baik dan abstraksi basis data melalui Eloquent ORM [1] [2]. Eloquent menyederhanakan interaksi basis data secara signifikan melalui pendekatan Active Record, namun kemudahan ini membawa biaya tersembunyi yang seringkali tidak disadari sampai sistem berada di bawah beban produksi. Secara default, Eloquent menerapkan lazy loading pada relasi tabel, mengambil setiap relasi hanya ketika pertama kali diakses. Di bawah lalu lintas tinggi,

DOI : https://doi.org/10.35316/jimi.v11i1.117-130

P-ISSN : 2549-7480
M. B. Ilmi dkk/ JIMI 11 (1) pp. 117-130 E-ISSN : 2549-6301

perilaku ini memicu ratusan kueri basis data identik dalam hitungan milidetik, masalah kueri N+1 yang terkenal memberikan tekanan yang semakin besar pada MySQL [3]. Mekanisme ini adalah akar penyebab utama penurunan kinerja dalam penerapan Laravel skala produksi.

Redis (Remote Dictionary Server) adalah penyimpanan struktur data dalam memori sumber terbuka yang dibangun di atas model key-value. Pada konfigurasi node tunggal standar, ia mendukung lebih dari satu juta operasi per detik dengan latensi yang secara konsisten di bawah satu milidetik [4]. Kemampuan ini didasarkan pada arsitektur event loop single-threaded yang dipadukan dengan multiplexing I/O melalui epoll dan kqueue, dilengkapi dengan struktur data yang dioptimalkan secara internal yang direkayasa untuk akses berkecepatan tinggi. Penelitian yang memeriksa metode debugging kinerja di lingkungan microservice yang bergantung pada penyimpanan dalam memori mengkonfirmasi bahwa arsitektur ini menghasilkan latensi yang secara konsisten lebih rendah daripada solusi berbasis disk dengan orde besaran di bawah beban yang bervariasi [5]. Laravel hadir dengan dukungan Redis asli melalui driver Predis dan PhpRedis, sehingga integrasi menjadi layak tanpa perubahan arsitektur utama.

Tiga strategi caching paling banyak dipelajari dan diterapkan dalam pengembangan aplikasi web. Cache-Aside, juga disebut lazy loading, mengambil data dari cache ketika tersedia, dan jika cache miss, data diambil dari database dan ditulis ke cache untuk melayani permintaan di masa mendatang. Write-Through memperbarui cache dan database secara bersamaan pada setiap penulisan, menjaga konsistensi data dengan penalti latensi penulisan yang kecil. Write-Behind mencatat pembaruan ke cache terlebih dahulu dan menyebarkannya secara asinkron ke database, meningkatkan kinerja penulisan sambil memperkenalkan risiko kehilangan data jika terjadi kegagalan sebelum sinkronisasi selesai [6]. Evaluasi empiris dari strategi-strategi ini pada infrastruktur yang dihosting AWS menemukan bahwa Cache-Aside memberikan pengurangan waktu respons yang paling terukur untuk pola akses yang banyak membaca, yaitu pola dominan di sebagian besar aplikasi web [6]. Oleh karena itu, studi ini mengadopsi Cache-Aside sebagai strategi caching utamanya, yang didukung secara native oleh Laravel melalui fungsi Cache::remember().

Sejumlah penelitian empiris telah mendokumentasikan efektivitas Redis di berbagai konteks arsitektur. Dalam lingkungan Node.js tanpa server, Redis yang diterapkan sebagai lapisan caching mengurangi latensi rata-rata hingga 60% relatif terhadap baseline tanpa caching, termasuk dalam kondisi cold-start yang merupakan kelemahan yang diakui dari arsitektur tanpa server [7]. Sebuah studi terpisah yang membandingkan pola arsitektur SOA dan microservice dalam aplikasi Laravel berbasis Docker menemukan bahwa MSA menghasilkan waktu respons 54,21% lebih efisien. Studi yang sama mengkonfirmasi bahwa overhead kontainerisasi Docker cukup kecil sehingga hasil uji beban dari lingkungan Docker tetap menjadi proksi yang valid untuk perilaku produksi [8]. Dalam domain perbandingan kerangka kerja PHP, benchmarking antara Laravel dan CodeIgniter menunjukkan bahwa Laravel yang dikonfigurasi dengan caching yang tepat menghasilkan latensi yang kompetitif meskipun memiliki overhead kerangka kerja yang lebih besar [9]. Perbandingan langsung antara Redis dan Memcached di seluruh penerapan

DOI : https://doi.org/10.35316/jimi.v11i1.117-130

cloud-native yang dikontainerisasi secara konsisten menempatkan Redis sebagai yang lebih unggul dalam latensi dan throughput di bawah setiap kondisi beban yang diuji [10]. Penelitian tentang pemanfaatan sumber daya dalam aplikasi web cloud seluler juga mengkonfirmasi bahwa Redis secara terukur mengurangi beban komputasi sisi server [11].

Pengujian beban mensimulasikan tekanan pengguna nyata untuk mengungkap batas kinerja sistem sebelum menghadapi lalu lintas produksi yang sebenarnya [13]. k6, alat pengujian beban berbasis JavaScript sumber terbuka dari Grafana Labs, dipilih untuk penelitian ini karena mendukung skrip pengujian yang dapat dikonfigurasi, terintegrasi ke dalam pipeline CI/CD, dan menghasilkan output statistik terperinci termasuk rincian persentil P50, P95, dan P99 [12]. Persentil P95 dan P99 sangat signifikan: keduanya mewakili pengguna di ujung distribusi respons mereka yang paling langsung terpengaruh oleh beban tinggi dan yang pengalamannya paling berpengaruh terhadap keandalan sistem yang dirasakan. Meskipun studi-studi yang ditinjau di atas secara kolektif membangun fondasi yang kuat untuk efektivitas Redis dan validitas pengujian berbasis Docker, tidak satu pun yang secara langsung mengukur kesenjangan kinerja aplikasi Laravel dengan dan tanpa Redis di bawah beban konkuren yang diskalakan secara sistematis (50 hingga 500 pengguna) dalam lingkungan yang sepenuhnya dapat direproduksi dengan cakupan metrik yang meluas hingga P99. Menutup kesenjangan tersebut adalah tujuan utama dari penelitian ini.

3. METODE PENELITIAN

Bagian ini menjelaskan alur penelitian yang ditunjukkan pada Gambar 1. Langkah-langkah sub-bagian yang lebih rinci akan dibahas di subbagian selanjutnya.

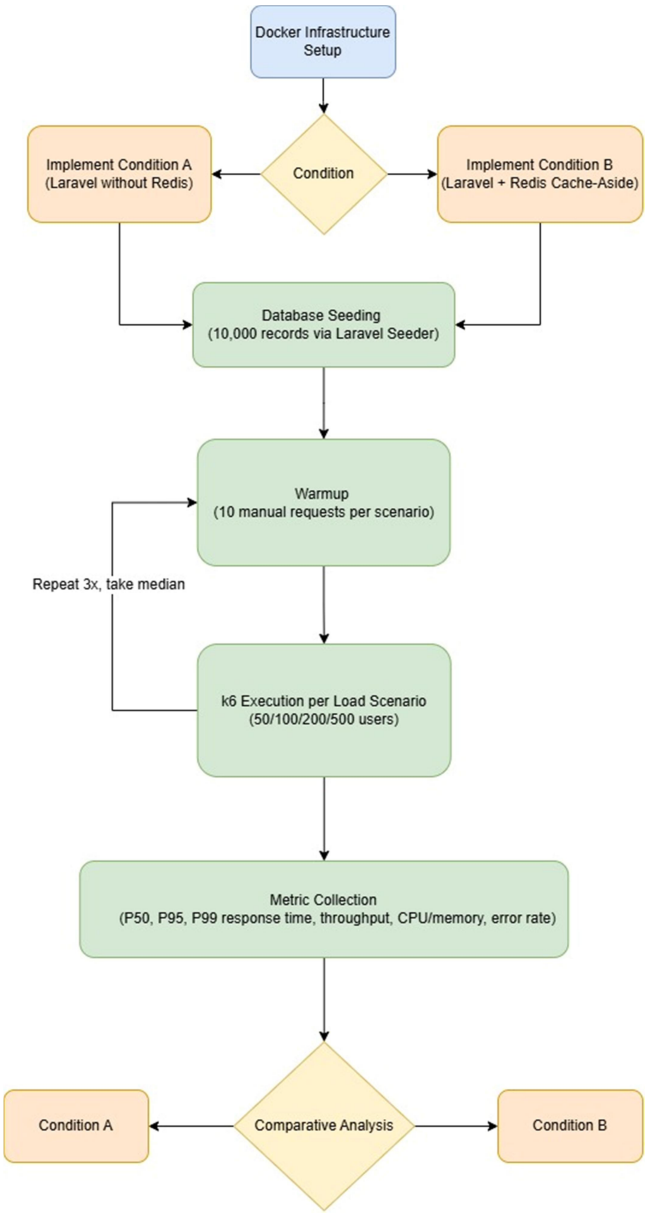
a. Desain Penelitian

Studi ini menggunakan pendekatan eksperimental kuantitatif yang terstruktur di sekitar perbandingan dua kelompok. Tidak ada pengguna nyata yang terlibat. Sebaliknya, dua konfigurasi teknis dari aplikasi yang identik ditempatkan berdampingan. Kondisi A berfungsi sebagai garis dasar, di mana setiap permintaan masuk langsung menuju MySQL tanpa perantara. Kondisi B memperkenalkan Redis sebagai lapisan caching yang beroperasi di bawah strategi Cache-Aside. Kedua kondisi tersebut dikenai empat tingkat beban, yaitu 50, 100, 200, dan 500 pengguna bersamaan, masing-masing dipertahankan selama 60 detik dengan periode peningkatan 10 detik. Setiap skenario diulang tiga kali dan nilai median diambil sebagai titik data akhir, mengurangi pengaruh variasi insidental.

Alur kerja penelitian secara keseluruhan mengikuti delapan tahap berurutan yang dapat direpresentasikan sebagai diagram alur linier:

DOI : https://doi.org/10.35316/jimi.v11i1.117-130

P-ISSN : 2549-7480
M. B. Ilmi dkk/ JIMI 11 (1) pp. 117-130 E-ISSN : 2549-6301



Gambar 1. Proses penelitian

Tabel 1. Kondisi

Eksperimental	Keterangan
Kondisi A (Garis Dasar)	Laravel tanpa Redis: setiap kueri dieksekusi langsung terhadap MySQL.
Kondisi B (Perawatan)	Laravel dengan Redis: hasil kueri disimpan di Redis menggunakan Strategi Cache-Aside.

Setiap kotak dalam alur di atas mewakili satu tahap dalam prosedur pengujian. Panah lingkaran pada langkah [6] menunjukkan tiga pengulangan sebelum maju ke pengumpulan metrik.

b. Lingkungan Pengujian

Semua pengujian dijalankan pada satu mesin fisik yang menjalankan Docker sebagai platform kontainerisasi. Menggunakan loopback (localhost)

P-ISSN: 2549-7480
M. B. Ilmi dkk/ JIMI 11 (1) pp. 117-130 E-ISSN: 2549-6301

sebagai antarmuka jaringan sepenuhnya menghilangkan latensi jaringan eksternal dari pengukuran. Spesifikasi perangkat keras terdiri dari prosesor 8-inti yang berjalan pada 3,6 GHz (setara dengan Intel Core i7 atau AMD Ryzen 7), memori DDR4 16 GB, dan SSD NVMe 256 GB. Sistem operasi di dalam kontainer adalah Ubuntu 22.04 LTS.

Tumpukan teknologi terdiri dari PHP 8.3 (FPM), Laravel 11.x, MySQL 8.0, Redis 7.2, Nginx 1.25, Docker 25.x, dan k6 versi 0.50 atau lebih tinggi untuk pengujian beban. Setiap versi komponen mencerminkan rilis stabil terbaru yang tersedia pada saat percobaan, memastikan hasilnya mewakili karakteristik kinerja dunia nyata saat ini dan bukan konfigurasi yang sudah usang.

Tabel 2. Spesifikasi Lingkungan Pengujian

Komponen	Spesifikasi
CPU	Intel Core i7 / AMD Ryzen 7 (8 core, 3.6 GHz)
RAM	16 GB DDR4
Storage	SSD NVMe 256 GB
OS	Ubuntu 22.04 LTS (via Docker)
Network	Loopback (localhost)

Tabel 3. Tumpukan Teknologi yang Digunakan

Komponen	Versi
PHP	8.3 (FPM)
Laravel	11.x
MySQL	8.0
Redis	7.2
Nginx	1.25
Docker	25.x
k6 (pengujian beban)	0.50+

Basis data telah diisi sebelumnya dengan 10.000 catatan produk yang dihasilkan melalui Laravel Seeder dan pustaka Faker. Volume ini mencerminkan ukuran data aplikasi e-commerce skala menengah yang cukup besar untuk memberikan beban nyata pada MySQL namun tetap dapat dikelola dari sudut pandang infrastruktur.

c. Implementasi Dua Kondisi

Dalam Kondisi A, setiap permintaan ke GET /api/products memicu eksekusi SQL langsung terhadap MySQL. Karena Eloquent menerapkan lazy loading secara default, relasi tabel, khususnya produk dan kategorinya, diambil secara terpisah untuk setiap permintaan. Pada volume lalu lintas tinggi, perilaku ini menghasilkan ratusan kueri identik secara berurutan dengan cepat, sehingga memberikan tekanan yang semakin besar pada basis data. Tidak ada mekanisme penyimpanan sementara yang aktif dalam konfigurasi ini.

DOI : https://doi.org/10.35316/jimi.v11i1.117-130

P-ISSN : 2549-7480
M. B. Ilmi dkk/ JIMI 11 (1) pp. 117-130 E-ISSN : 2549-6301

Dalam Kondisi B, hasil kueri tidak dibuang setelah dikirim ke klien. Sebaliknya, hasil tersebut disimpan di Redis melalui fungsi Cache::remember() bawaan Laravel mengikuti pola Cache-Aside. Kunci cache dibuat dari nomor halaman (products:list:page:{page}) dengan TTL 300 detik. TTL ini dipilih karena data produk bersifat semi-statis dan tidak memerlukan pembaruan waktu nyata. Selain caching kueri, tiga optimasi lebih lanjut diaktifkan: caching konfigurasi, caching rute, dan caching tampilan melalui perintah Artisan masing-masing. Konfigurasi .env mengatur CACHE_DRIVER=redis dan SESSION_DRIVER=redis, yang mengarah ke kontainer Redis pada port 6379.

Tabel 4 Empat titik akhir dipilih untuk mewakili pola akses yang berbeda.

Endpoints	Nilai Yang Digunakan
GET /api/products	Daftar produk berhalaman (cache aktif)
GET /api/products/{id}	Detail produk tunggal (cache aktif)
GET /api/products/search	Pencarian produk (cache 60 detik per kueri)
POST /api/products	Menambahkan produk baru (cache tidak valid)

Pemilihan ini memastikan pengujian mencakup operasi baca-secara intensif yang merupakan target utama caching, sekaligus menyertakan satu operasi tulis untuk mengamati perilaku invalidasi cache.

d. Skenario dan Prosedur Pengujian Beban

k6 dikonfigurasi untuk mensimulasikan beban bertahap: peningkatan bertahap selama 10 detik dari nol hingga jumlah pengguna target, diikuti oleh keadaan stabil selama 60 detik pada jumlah tersebut, kemudian penurunan bertahap selama 10 detik kembali ke nol. Pola ini lebih mewakili lalu lintas dunia nyata daripada lonjakan instan, karena beban pengguna aktual biasanya meningkat secara progresif daripada muncul sekaligus.

Prosedur yang diulang untuk setiap skenario beban berjalan sebagai berikut. Pertama, lingkungan Docker direset sepenuhnya (docker-compose down -v && docker-compose up -d) untuk menghilangkan sisa keadaan dari skenario sebelumnya. Kedua, basis data diisi ulang dengan 10.000 catatan melalui php artisan db:seed. Ketiga, 10 permintaan manual dikirim sebagai pemanasan untuk menstabilkan kompilasi PHP JIT dan kumpulan koneksi. Keempat, skrip k6 dieksekusi untuk tingkat beban target. Kelima, waktu respons P50, P95, dan P99, throughput, dan tingkat kesalahan dicatat dari output k6, sementara konsumsi CPU dan memori dipantau secara bersamaan melalui docker stats. Keenam, seluruh rangkaian diulangi dua kali lagi (total tiga kali percobaan per skenario) dan nilai median dari ketiga percobaan tersebut diambil sebagai nilai akhir untuk perbandingan.

e. Metrik Evaluas

Enam metrik diukur dan dibandingkan antara kedua kondisi tersebut. Waktu respons pada P50 mencerminkan pengalaman pengguna yang umum. P95 mencakup sebagian besar kasus beban berat. P99 mengisolasi kasus ekstrem yang paling sensitif terhadap penurunan kinerja. Throughput, yang diukur dalam permintaan per detik, menunjukkan kapasitas layanan secara

DOI : https://doi.org/10.35316/jimi.v11i1.117-130

keseluruhan. Tingkat kesalahan, yang dinyatakan sebagai persentase dari total permintaan, berfungsi sebagai indikator keandalan. Konsumsi CPU dan memori dicatat selama fase kondisi stabil untuk mengukur efisiensi sumber daya. Bersama-sama, keenam dimensi ini memberikan gambaran kinerja komprehensif yang menghindari distorsi akibat mengandalkan satu pengukuran saja.

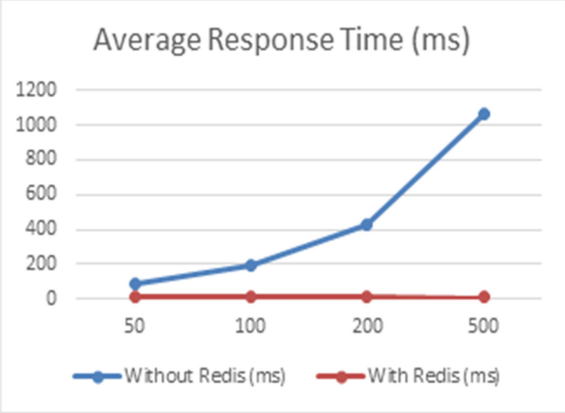
4. HASIL DAN PEMBAHASAN

a. Waktu Respons Rata-rata

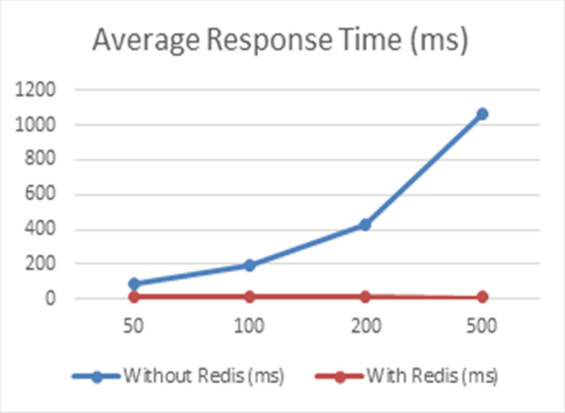
Tabel di bawah ini merangkum waktu respons rata-rata untuk setiap skenario beban untuk Kondisi A (tanpa Redis) dan Kondisi B (dengan Redis).

Tabel 5. Waktu Respons Rata-rata (ms)

Pengguna Bersamaan	Tanpa Redis (ms)	Dengan Redis (ms)	Delta
50	85	15	-81.8%
100	195	16	-92.0%
200	422	14	-96.7%
500	1069	12	-98.9%



Gambar 2. Waktu Respons Rata-rata (ms)



Gambar 3. Delta (%)

Angka-angka dalam tabel di atas menunjukkan pola yang konsisten: semakin tinggi jumlah pengguna bersamaan, semakin besar perbedaan kinerja antara kedua kondisi tersebut. Pada beban 50 pengguna, waktu respons Kondisi A mencapai 85 ms, hampir enam kali lipat dari 15 ms pada Kondisi B. Kesenjangan ini melebar tajam seiring peningkatan beban: dalam skenario 500 pengguna, Kondisi A mencatat 1.069 ms, sedangkan waktu Kondisi B turun menjadi 12 ms. Rata-rata di semua skenario, Redis mampu mengurangi waktu respons sebesar 92,3%. Perbedaan hampir dua orde besaran pada beban puncak ini mencerminkan mekanisme inti Redis: setiap cache hit dilayani dari memori tanpa menyentuh MySQL sama sekali, sehingga latensi yang dirasakan pengguna tetap hampir tidak berubah bahkan ketika antrian permintaan bertambah [4].

b. Distribusi Waktu Respons (P50, P95, P99)

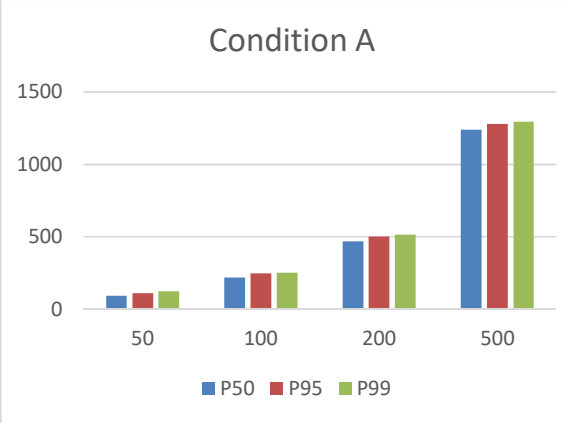
Nilai rata-rata saja tidak cukup untuk sepenuhnya menggambarkan pengalaman pengguna, terutama dalam kondisi beban tinggi. Persentil P95 dan

P-ISSN : 2549-7480
M. B. Ilmi dkk/ JIMI 11 (1) pp. 117-130 E-ISSN : 2549-6301

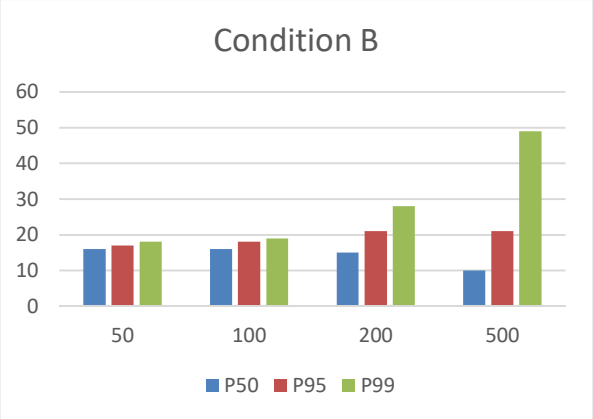
P99 mengungkapkan kondisi yang dialami oleh pengguna dengan respons yang tertunda, kelompok yang paling memengaruhi persepsi keandalan layanan. Tabel di bawah ini menyajikan distribusi lengkap untuk kedua kondisi tersebut.

Tabel 6. Distribusi Waktu Respons (ms)

Kondisi	Skenario	P50	P95	P99
Kondisi A	50 VUs	93	111	125
Kondisi A	100 VUs	218	248	251
Kondisi A	200 VUs	470	501	515
Kondisi A	500 VUs	1241	1279	1294
Kondisi B	50 VUs	16	17	18
Kondisi B	100 VUs	16	18	19
Kondisi B	200 VUs	15	21	28
Kondisi B	500 VUs	10	21	49



Gambar 4. Kondisi A



Gambar 5. Kondisi B

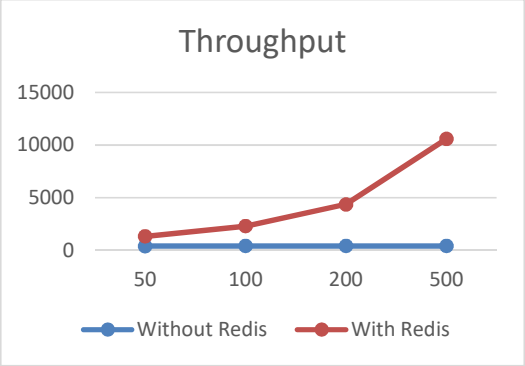
c. Throughput

Sementara waktu respons mengukur kecepatan per permintaan individual, throughput mengukur kapasitas sistem total berapa banyak permintaan yang dapat dilayani per detik di bawah beban tertentu. Tabel di bawah ini menunjukkan data throughput untuk kedua kondisi tersebut.

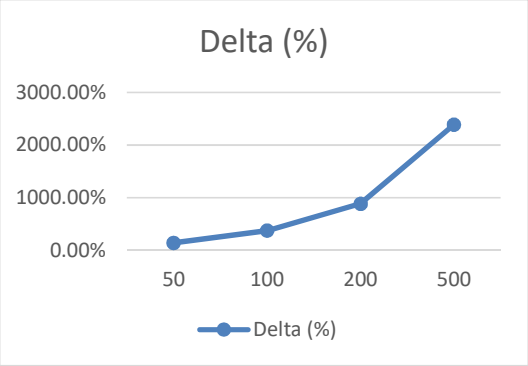
Tabel 7. Throughput (Permintaan per Detik)

Pengguna Bersamaan	Tanpa Redis (ms)	Dengan Redis (ms)	Delta
50	387.0	938.4	+142.5%
100	400.7	1893.3	+372.5%
200	402.4	3960.2	+884.0%
500	408.5	10164.5	+2388.3%

DOI : https://doi.org/10.35316/jimi.v11i1.117-130



Gambar 6. Throughput



Gambar 7. Delta (%)

Data dalam tabel di atas mengungkapkan karakteristik yang mencolok. Throughput Kondisi A hampir datar di semua skenario beban, bergerak dari 387,0 req/s pada 50 pengguna menjadi hanya 408,5 req/s pada 500 pengguna. Ini menunjukkan bahwa MySQL mencapai saturasi jauh sebelum beban 500 pengguna: tidak peduli berapa banyak permintaan yang masuk, kapasitas pemrosesan tidak meningkat karena koneksi basis data dan thread sudah penuh. Kondisi B menunjukkan pola yang sepenuhnya berlawanan: throughput tumbuh hampir linier dengan meningkatnya beban, mencapai 10.164,5 req/s pada 500 pengguna. Peningkatan rata-rata di semua skenario adalah 946,8%. Angka ini bukan anomali statistik, tetapi lebih merupakan cerminan langsung dari arsitektur Redis: setiap cache hit diproses dalam memori tanpa melalui antrian kueri MySQL, sehingga menggeser batas kapasitas sistem secara signifikan ke atas [4] [10].

d. Tingkat Kesalahan

Salah satu kekhawatiran umum saat menerapkan caching adalah risiko inkonsistensi data, di mana klien menerima data usang dan tidak relevan, atau lebih buruk lagi, kegagalan permintaan karena konflik status antara cache dan basis data. Tabel di bawah ini mencatat tingkat kesalahan di semua sesi pengujian.

Tabel 8. Tingkat Kesalahan (%)

Concurrent Users	Without Redis	With Redis
50	0.00%	0.00%
100	0.00%	0.00%
200	0.00%	0.00%
500	0.00%	0.00%

Kedua kondisi tersebut mempertahankan tingkat kesalahan 0,00% di setiap skenario, termasuk pada 500 pengguna bersamaan. Hasil ini memperjelas dua hal. Implementasi Cache Aside aman: logika TTL dan invalidasi yang diterapkan di sini tidak menghasilkan inkonsistensi yang meningkat menjadi kesalahan. Lebih dari itu, hal ini menegaskan bahwa hingga 500 pengguna bersamaan, kedua konfigurasi tetap stabil secara operasional. Masalah dengan Kondisi A bukanlah keruntuhan sistem, tetapi degradasi latensi progresif yang membuat layanan tidak dapat digunakan dari perspektif pengalaman

DOI : https://doi.org/10.35316/jimi.v11i1.117-130

P-ISSN : 2549-7480
M. B. Ilmi dkk/ JIMI 11 (1) pp. 117-130 E-ISSN : 2549-6301

pengguna jauh sebelum layanan tersebut gagal total.

e. Diskusi

Jika digabungkan, keempat tabel tersebut membentuk gambaran yang koheren dan saling memperkuat. Redis tidak hanya meningkatkan satu metrik secara terpisah, tetapi juga memposisikan ulang seluruh profil kinerja aplikasi. Waktu respons turun rata-rata 92,3%, throughput meningkat rata-rata 946,8%, distribusi persentil menjadi stabil, dan tingkat kesalahan tidak berubah. Hasil ini selaras dengan penelitian empiris sebelumnya yang mendokumentasikan efektivitas Redis di berbagai konteks arsitektur [7] [10].

Tiga mekanisme secara bersamaan menjelaskan hasilnya. Yang pertama adalah penghapusan latensi I/O: sebuah cache hit Redis selesai dalam 10 hingga 16 ms, jauh lebih singkat daripada 93 hingga 1.241 ms yang dibutuhkan untuk kueri MySQL tergantung pada beban. Kesenjangan tersebut bertambah seiring dengan jumlah pengguna karena MySQL mulai bersaing untuk koneksi dan penguncian tabel sementara Redis terus melayani dari memori tanpa persaingan tersebut. Mekanisme kedua adalah efisiensi kumpulan koneksi. Ketika sebagian besar permintaan dilayani oleh Redis tanpa mencapai MySQL, kumpulan koneksi basis data yang terbatas tetap tersedia dan tidak jenuh bahkan pada beban puncak. Yang ketiga adalah keunggulan skalabilitas non-linier Redis: P99 MySQL meningkat sepuluh kali lipat dari 125 ms menjadi 1.294 ms seiring bertambahnya jumlah pengguna dari 50 menjadi 500, sementara P99 Redis hanya meningkat 2,7 kali lipat dari 18 ms menjadi 49 ms pada rentang yang sama.

Temuan yang paling penting adalah identifikasi 100 pengguna bersamaan sebagai titik infleksi kritis. Di bawah ambang batas tersebut, Kondisi A masih memberikan waktu respons yang dapat ditoleransi (rata-rata 85 ms pada 50 pengguna). Setelah beban melampaui 100 pengguna, konfigurasi tanpa cache mulai gagal mengimbangi: waktu respons melonjak dari 195 ms menjadi 422 ms hanya dengan penambahan 100 pengguna lagi, sementara throughput hampir tidak berubah. Kondisi B tidak menunjukkan infleksi yang sebanding hingga 500 pengguna. Hal ini memberikan praktisi sebuah titik keputusan yang konkret: setiap penerapan Laravel yang diharapkan melayani lebih dari 100 pengguna simultan harus memperlakukan Redis bukan sebagai optimasi opsional tetapi sebagai komponen arsitektur yang diperlukan [2] [4].

5. HASIL DAN PEMBAHASAN

Studi ini secara langsung mengukur efek Redis, yang diterapkan sebagai lapisan caching dalam memori, pada kinerja aplikasi Laravel di empat skenario beban bertahap: 50, 100, 200, dan 500 pengguna bersamaan. Hasilnya konsisten di setiap tingkat beban. Redis mengurangi waktu respons rata-rata sebesar 92,3%, dengan penurunan paling tajam pada satu skenario terjadi pada 500 pengguna bersamaan dari 1.069 ms menjadi 12 ms, pengurangan sebesar 98,9%. Throughput rata-rata meningkat sebesar 946,8%, yang mencerminkan Redis membebaskan MySQL dari beban mengeksekusi kueri identik berulang kali. Yang sama pentingnya, kedua kondisi tersebut mempertahankan tingkat kesalahan 0,00% di semua skenario, yang menegaskan bahwa implementasi Redis tidak memperkenalkan ketidakstabilan baru ke dalam sistem.

Temuan ini secara langsung diterjemahkan menjadi panduan yang dapat ditindaklanjuti. Penerapan Laravel yang diharapkan melayani lebih dari 100

DOI : https://doi.org/10.35316/jimi.v11i1.117-130

pengguna bersamaan harus memperlakukan Redis sebagai komponen arsitektur yang diperlukan daripada optimasi opsional. Di bawah ambang batas tersebut, Redis masih memberikan manfaat yang terukur, tetapi marginnya jauh lebih kecil daripada pada beban yang lebih tinggi. Strategi caching yang diterapkan dalam studi ini, Cache::remember() dengan TTL 300 detik untuk caching query, dikombinasikan dengan caching rute dan caching konfigurasi terbukti efektif untuk endpoint baca yang sering diakses yang didukung oleh data yang tidak berubah dengan cepat.

Beberapa keterbatasan berlaku. Semua pengujian dilakukan dalam lingkungan single-node yang terkontrol, sehingga hasilnya mungkin berbeda pada infrastruktur cloud multi-node atau produksi. Dataset sintetis 10.000 produk mungkin tidak sepenuhnya mencerminkan distribusi data dunia nyata. Perilaku invalidasi cache di bawah lalu lintas intensif penulisan juga tidak diuji dalam studi ini. Pekerjaan di masa mendatang dapat mengevaluasi Redis Cluster pada arsitektur multi-node, membandingkan Redis dengan alternatif seperti Memcached atau Varnish, memeriksa dampak strategi invalidasi cache yang berbeda pada konsistensi data, atau menguji kinerja di bawah beban yang jauh lebih berat (1.000 pengguna bersamaan atau lebih) untuk mengidentifikasi batas kinerja Redis dalam kondisi produksi yang realistis.

6. UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Politeknik Hasnur atas dukungan institusional yang diberikan selama pelaksanaan penelitian ini. Apresiasi juga disampaikan kepada rekan-rekan di Program Studi Teknologi Rekayasa Multimedia dan Bisnis Digital atas masukan dan diskusi yang konstruktif dalam proses penyusunan makalah ini.

7. KESIMPULAN

Penelitian ini membuktikan secara empiris bahwa integrasi Redis sebagai lapisan caching dalam memori memberikan peningkatan kinerja yang substansial pada aplikasi web berbasis Laravel di lingkungan dengan beban tinggi. Melalui pengujian beban terkontrol menggunakan k6 pada empat skenario pengguna bersamaan (50, 100, 200, dan 500 pengguna), diperoleh tiga kesimpulan utama.

Pertama, Redis secara konsisten mengurangi waktu respons rata-rata sebesar 92,3% di seluruh skenario, dengan penurunan paling tajam mencapai 98,9% pada 500 pengguna bersamaan (dari 1.069 ms menjadi 12 ms). Kedua, throughput meningkat rata-rata sebesar 946,8%, mencerminkan kemampuan Redis dalam membebaskan MySQL dari eksekusi kueri identik yang berulang sehingga kapasitas layanan sistem meningkat secara signifikan. Ketiga, kedua konfigurasi mempertahankan tingkat kesalahan 0,00% di seluruh skenario pengujian, yang mengkonfirmasi bahwa penerapan Redis tidak memperkenalkan ketidakstabilan baru ke dalam sistem.

Temuan penting dari studi ini adalah identifikasi 100 pengguna bersamaan sebagai titik infleksi kritis. Di bawah ambang batas tersebut, konfigurasi tanpa caching masih mampu memberikan waktu respons yang dapat ditoleransi. Namun setelah beban melampaui angka tersebut, degradasi kinerja pada konfigurasi tanpa Redis berlangsung secara drastis, sementara konfigurasi dengan Redis tetap stabil hingga 500 pengguna. Oleh karena itu, setiap penerapan Laravel yang diperkirakan melayani lebih dari 100 pengguna

DOI : https://doi.org/10.35316/jimi.v11i1.117-130

P-ISSN : 2549-7480
E-ISSN : 2549-6301

bersamaan sebaiknya memperlakukan Redis bukan sebagai optimasi opsional, melainkan sebagai komponen arsitektur yang esensial.

8. REFERENSI

[1] "Stack Overflow Developer Survey," 2023. [Online]. Available: https://survey.stackoverflow.co/2023/.

[2] T. Otwell, "Laravel Documentation, Version 11.x," 2024. [Online]. Available: https://laravel.com/docs/11.x.

[3] A. Niarman, Iswandi and a. A. K. Candri, "Comparative Analysis of PHP Frameworks for Development of," *International Journal Software Engineering and Computer Science (IJSECS),* vol. III, p. 424–436, 2023.

[4] "Redis Documentation 7.2," Redis Ltd, 2024. [Online]. Available: https://redis.io/documentation.

[5] H. M. Kabamba, M. Khouzam and a. M. Dagenais, "Advanced Strategies for Precise and Transparent Debugging of Performance Issues in In-Memory Data Store-Based Microservices," *arXiv,* pp. 1-14, 2023.

[6] D. Richardson and W. Hao, "An Empirical Study of Caching," *IEEE Xplore,* 2024.

[7] B. C. Ghosh, S. K. Addya, N. B. Somy and C. Hota, "Caching Techniques to Improve Latency in Serverless," *IEEE Xplore,* p. 336–341, 2020.

[8] H. Calderón-Gómez, L. Mendoza-Pittí, M. Vargas-Lombardo, J. M. Gómez-Pulido, D. Rodríguez-Puyol, G. Sención and M.-L. Polo-Luque, "Evaluating Service-Oriented and Microservice Architecture Patterns to Deploy eHealth Applications in Cloud Computing Environment," *MDPI,* vol. 11, pp. 1-28, 2021.

[9] M. K. Ahmed, A. H. Bello, S. S. Jauro and a. M. Dawaki, "A comparative analysis of performance optimization techniques for benchmarking PHP frameworks: Laravel and Codeigniter," *Dutse Journal of Pure and Applied Sciences,* vol. 10, p. 284–295, 2024.

[10] D. E. Ukene, "Assessing Performance Optimization Strategies In Cloud-Native Environments Through Containerization And Orchestration Analysis," *Georgia Southern Commons,* 2024.

[11] P. Sarkar, K. Chakravarty, B. Sahoo, R. Singh, S. M. Turjyaa and A. Bandyopadhyay, "Efficient Resource Utilization for Web Applications in Mobile Cloud Computing," *IEEE Xplore,* 2024.

[12] "k6 Documentation: Open Source Load Testing," Grafana Labs, 2024. [Online]. Available: https://k6.io/docs/.

DOI : https://doi.org/10.35316/jimi.v11i1.117-130