Tersedia secara online di situs web: https://jurnal.iaii.or.id/index.php/RESTI

JURNAL RESTI

(Rekayasa Sistem dan Teknologi Informasi)

Vol. 9 No. 3 (2025) 594 - 600

e-ISSN: 2580-0760

Perbandingan Kinerja Arsitektur Monolitik dan Mikroservis dalam Menangani Transaksi Bervolume Tinggi

Mastura Diana Marieska1*, Arya Yunanta2 , Harisatul Aulia3 , Alvi Syahrini Utami4 , Muhammad Qurhanul Rizqie5

1,2,3,4,5Jurusan Teknik Informatika, Fakultas Ilmu Komputer, Universitas Sriwijaya, Palembang, Indonesia

1mastura.diana@ilkom.unsri.ac.id, 2 aryayun90@gmail.com, 3haris.aulia404@gmail.com, 4 alvisyahrini@ilkom.unsri.ac.id, 5qurhanul.rizqie@ilkom.unsri.ac.id

Abstrak

Arsitektur monolitik dan mikroservis adalah dua pendekatan berbeda untuk mendesain dan mengembangkan aplikasi. Namun, arsitektur ini menunjukkan karakteristik yang kontras. Dalam arsitektur monolitik, semua komponen aplikasi membentuk entitas terpadu dengan bagian-bagian yang saling terkait erat, sedangkan mikroservis memecah aplikasi menjadi layanan-layanan independen dan ringan yang dapat dikembangkan, diimplementasikan, dan diperbarui secara terpisah. Mikroservis sering dianggap lebih unggul daripada arsitektur monolitik dalam hal kinerja. Studi ini bertujuan untuk membandingkan kinerja arsitektur monolitik dan mikroservis dalam menangani volume transaksi yang tinggi. Penting untuk mengamati bagaimana kedua arsitektur tersebut berperilaku ketika menangani transaksi dari sejumlah besar pengguna bersamaan. Sebuah prototipe sistem tiket online diimplementasikan untuk kedua arsitektur untuk memungkinkan analisis komparatif. Metrik kinerja yang dipilih adalah waktu respons dan tingkat kesalahan. Hasil eksperimen menunjukkan bahwa dalam kondisi beban tinggi, mikroservis mengungguli arsitektur monolitik, menunjukkan waktu respons 36% lebih cepat dan 71% lebih sedikit kesalahan. Namun, dalam kondisi beban berlebih—ketika penggunaan CPU melebihi 90%—kinerja mikroservis menurun secara signifikan. Hal ini tidak berarti bahwa microservices tidak dapat menangani sejumlah besar pengguna bersamaan, tetapi menyoroti perlunya manajemen sumber daya yang lebih baik.

Kata kunci: arsitektur berbasis peristiwa; layanan mikro; monolitik; sistem tiket daring

Cara Mengutip: MD Marieska, Arya Yunanta, Harisatul Aulia, Alvi Syahrini Utami, dan Muhammad Qurhanul Rizqie, "Perbandingan Kinerja Arsitektur Monolitik dan Microservices dalam Menangani Transaksi Volume Tinggi", *J. RESTI (Rekayasa Sist. Teknol. Inf.)* , vol. 9, tidak. 3, hlm. 594 - 600, Juni 2025.

Tautan Perma/ DOI: https://doi.org/10.29207/resti.v9i3.6183

Diterima: 20 November 2024
Diterima: 8 Juni 2025
Tersedia daring: 19 Juni 2025

1. Pendahuluan

Layanan monolitik adalah arsitektur perangkat lunak tradisional di mana seluruh aplikasi merupakan entitas tunggal dan terpadu dengan komponen yang saling terkait erat. Pendekatan ini menghadapi masalah signifikan terkait skalabilitas dan fleksibilitas. Peningkatan skala sistem untuk menangani permintaan yang lebih tinggi memerlukan peningkatan skala seluruh aplikasi, yang seringkali tidak efisien [1]. Selain itu, kegagalan pada satu komponen dapat berdampak pada kinerja seluruh aplikasi [2].

Sebaliknya, arsitektur microservices memecah aplikasi menjadi layanan-layanan independen dan ringan yang dapat dikembangkan, diterapkan, dan diperbarui secara terpisah. Pendekatan ini meningkatkan fleksibilitas dan skalabilitas dibandingkan dengan sistem monolitik [3]. Setiap microservice beroperasi secara independen dan berkomunikasi melalui protokol ringan seperti API [4]. Skalabilitas adalah komponen kunci manfaat yang dicari organisasi ketika beralih ke microservices [5].

Pergeseran dari arsitektur monolitik ke microservices menandai evolusi penting dalam pengembangan perangkat lunak, didorong oleh tuntutan akan skalabilitas, fleksibilitas, dan pemeliharaan yang lebih besar. Ini bukanlah perubahan yang sederhana, terutama jika berasal dari sistem berorientasi objek tradisional, karena membutuhkan penataan ulang banyak bagian sistem dan penambahan banyak interaksi antar layanan [6]. Selain itu, proses migrasi dapat memakan banyak sumber daya dan waktu, seringkali membutuhkan penulisan ulang aplikasi yang ada secara komprehensif [7], [8]. Tingkat detail terbaik untuk microservices sangat penting untuk migrasi ini karena memengaruhi kualitas aplikasi dan penggunaan sumber daya [9].

Penelitian komparatif menyoroti bahwa meskipun arsitektur monolitik memiliki basis kode tunggal dan terintegrasi,

Layanan mikro menawarkan layanan yang terpisah dan terhubung secara longgar [10]. Layanan mikro memberikan peningkatan skalabilitas, kelincahan, dan penyebaran layanan independen, yang mengarah pada peningkatan kinerja [11].
Transisi ke microservices dapat meningkatkan stabilitas, kinerja, dan skalabilitas, serta mendorong penggunaan sumber daya yang lebih efisien [12], [13]. Salah satu keunggulan utama microservices adalah kemampuannya untuk melakukan penskalaan secara independen, yang dapat secara signifikan meningkatkan kinerja di bawah permintaan tinggi. Akibatnya, sementara aplikasi monolitik mungkin menunjukkan latensi rata-rata yang lebih rendah untuk fitur-fitur tertentu, microservices seringkali memberikan kinerja yang lebih unggul secara keseluruhan [14].

Untuk mengelola sistem yang mengalami tuntutan kinerja tinggi karena banyaknya pengguna bersamaan secara efektif, sangat penting untuk melakukan studi kasus yang mensimulasikan lingkungan dengan lalu lintas tinggi tersebut. Studi ini sangat penting untuk menilai seberapa baik sistem menangani interaksi pengguna dan beban transaksi yang ekstensif, memberikan wawasan tentang skalabilitas dan keandalannya di bawah tekanan yang signifikan.

Contoh ilustratifnya adalah sistem penjualan tiket online yang dirancang untuk menangani acara berskala besar, seperti konser di tempat yang luas. Sistem seperti itu harus secara efisien mengelola volume transaksi yang tinggi dan akses pengguna secara simultan, yang menyoroti kebutuhan akan kemampuan kinerja yang tangguh. Penjualan tiket konser secara online menawarkan kemudahan yang signifikan dibandingkan dengan penjualan offline, yang mungkin melibatkan antrean panjang dan kerumunan besar.
Namun, penjualan tiket online dapat menghasilkan lalu lintas yang sangat tinggi, dengan ribuan tiket berpotensi terjual dalam hitungan menit.

Beberapa studi lain telah membandingkan arsitektur monolitik dan mikroservis. Satu studi [15] berfokus pada skalabilitas, menyimpulkan bahwa monolitik lebih cocok untuk sistem yang tidak memerlukan penanganan sejumlah besar pengguna bersamaan. Studi lain [16], menekankan aspek kinerja penggunaan basis data, menyimpulkan bahwa
Latensi akses basis data microservices lebih tinggi.
daripada monolitik. Studi lain [17] merinci tantangan yang dihadapi ketika bermigrasi dari monolitik ke layanan mikro.

Berbeda dengan penelitian lain, studi ini membandingkan arsitektur monolitik dan mikroservis untuk mengevaluasi mana yang memberikan kinerja dan keandalan yang lebih baik untuk sistem yang menangani banyak transaksi dan banyak pengguna yang berinteraksi secara bersamaan. Aspek kinerja akan dianalisis berdasarkan waktu respons rata-rata dan persentil ke-90. Aspek keandalan akan dianalisis berdasarkan tingkat kesalahan. Dengan menganalisis sistem tiket daring yang dirancang untuk acara berskala besar, studi ini bertujuan untuk menentukan bagaimana setiap arsitektur menangani tuntutan pemrosesan transaksi bervolume tinggi dan akses pengguna, yang pada akhirnya memberikan wawasan tentang efektivitasnya dalam mengelola skenario yang membutuhkan kinerja tinggi tersebut.

2. Metode

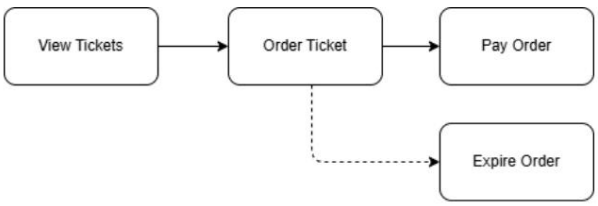
Bagian ini akan menjelaskan metode penelitian yang digunakan dalam studi ini, dengan fokus pada sistem tiket online sebagai objek analisis utama. Kami akan mengeksplorasi dua pendekatan arsitektur yang berbeda: implementasi monolit dan implementasi layanan mikro.
Setiap pendekatan akan diteliti untuk memahami struktur, prinsip desain, dan karakteristik operasionalnya.

2.1 Sistem Tiket Online

Tiket elektronik, versi digital dari tiket kertas tradisional untuk acara, semakin populer karena kenyamanan dan efisiensinya. Dibeli secara online dan disimpan di perangkat seluler, tiket elektronik menghilangkan kebutuhan akan tiket fisik dan menyederhanakan akses ke acara [18]. Tiket elektronik menyederhanakan pembelian, mengurangi penipuan, dan meningkatkan pengalaman pengguna dengan akses dan pembaruan langsung. Tiket elektronik juga terintegrasi dengan aplikasi seluler untuk notifikasi waktu nyata dan transfer yang mudah [18], [19].

Sistem penjualan tiket online dipilih untuk studi kasus ini karena volume transaksi yang signifikan yang harus ditanganinya. Misalnya, selama konser besar di stadion besar dengan penjualan tiket online, sistem tersebut harus mampu menangani volume transaksi yang besar. Memproses sejumlah besar transaksi secara efisien dalam waktu yang sangat singkat. Skenario ini menyoroti kebutuhan akan sistem yang sangat efisien dan tangguh yang mampu mengelola pembelian tiket yang cepat dan banyak secara lancar, sehingga memastikan pengalaman yang lancar dan andal bagi semua pengguna.

Dalam studi ini, sistem penjualan tiket online yang akan dikembangkan akan menggabungkan beberapa fitur utama, termasuk kemampuan untuk melihat tiket yang tersedia, membeli tiket, dan memproses pembayaran. Gambar 1 menunjukkan desain alur sistem antara fitur-fitur utama. Selain itu, sistem ini akan menyediakan fungsionalitas untuk memproses pembatalan tiket dalam jangka waktu tertentu.



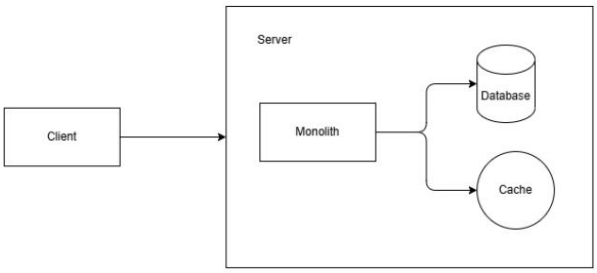
Gambar 1. Alur Sistem

Studi ini menggunakan tumpukan teknologi yang terdiri dari Go, PostgreSQL, Redis, dan Apache Kafka. Go digunakan karena konkurensi dan performanya yang efisien, PostgreSQL untuk manajemen data relasional yang tangguh, Redis untuk akses data dalam memori dan caching yang cepat, dan Kafka untuk streaming dan pemrosesan data secara real-time.

2.2 Implementasi Monolit

Layanan monolitik mewakili arsitektur perangkat lunak konvensional di mana seluruh aplikasi dikembangkan sebagai satu entitas tunggal yang terpadu. Desain ini mengintegrasikan semua komponen ke dalam satu sistem yang kohesif, yang menimbulkan tantangan signifikan bagi skalabilitas dan fleksibilitas. Dalam

Dalam implementasi monolitik, sistem ini menggunakan tiga komponen pada server: layanan web, yang merupakan monolit itu sendiri; cache untuk penyimpanan data sementara; dan basis data untuk penyimpanan data permanen. Gambar 2 mengilustrasikan interaksi antara ketiga komponen tersebut. Semua fitur sistem e-ticketing terintegrasi ke dalam satu layanan web, yang merupakan aplikasi monolitik.

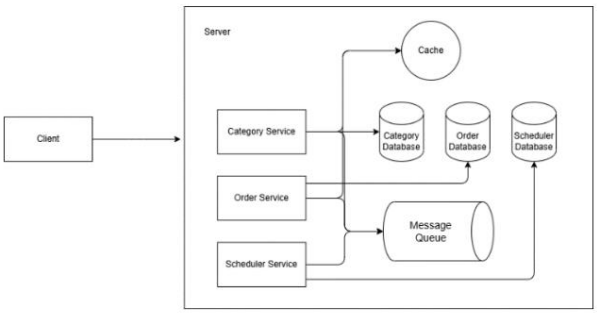


Gambar 2. Desain Sistem Tiket Online pada Monolit

2.3. Implementasi Microservice

Microservice adalah gaya arsitektur di mana suatu sistem dibagi menjadi layanan-layanan kecil yang dapat diimplementasikan secara independen, masing-masing bertanggung jawab atas fungsi tertentu. Pendekatan ini berbeda dengan sistem monolitik karena meningkatkan skalabilitas dan fleksibilitas, tetapi menghadirkan tantangan dalam hal komunikasi dan manajemen.

Dalam implementasi microservice, layanan web tunggal dibagi menjadi beberapa komponen berdasarkan domain bisnis menggunakan Domain-Driven Design (DDD). DDD menyelaraskan arsitektur perangkat lunak dengan domain bisnis dengan memahami secara mendalam industri spesifik dan menciptakan model domain yang mengatasi kompleksitas dan kebutuhannya [20], [21]. Pendekatan ini menghasilkan layanan yang berbeda seperti Layanan Kategori, Layanan Pemesanan, dan Layanan Penjadwal. Gambar 3 mengilustrasikan interaksi antara komponen dan layanan dalam microservice.



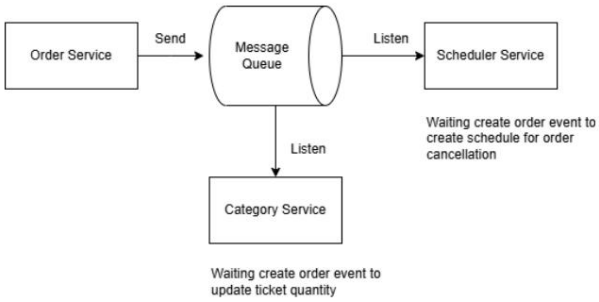
Gambar 3. Desain Sistem Tiket Online Berbasis Mikroservis

Layanan Kategori bertanggung jawab untuk mengelola logika bisnis yang terkait dengan kategori tiket, termasuk informasi kategori terperinci dan jumlah tiket yang tersisa yang tersedia dalam setiap kategori. Layanan Pemesanan menangani logika bisnis yang terkait dengan reservasi tiket, seperti memproses pemesanan tiket dan

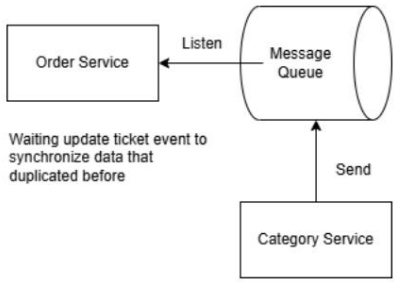
Membayar biaya reservasi tiket. Terakhir, Layanan Penjadwal bertanggung jawab untuk mengelola jadwal guna menentukan kapan reservasi akan dibatalkan sesuai dengan tenggat waktu yang telah ditentukan.

Dalam penelitian ini, microservice yang diimplementasikan menggunakan arsitektur berbasis peristiwa (event-driven architecture). Arsitektur berbasis peristiwa (EDA) adalah pola desain di mana komponen sistem berkomunikasi melalui peristiwa daripada interaksi langsung. Pendekatan ini memungkinkan sistem yang terdecoupled dan fleksibel yang meningkatkan skalabilitas dan responsivitas. Komunikasi asinkron [22]-[24] didukung oleh EDA, yang memungkinkan komponen bekerja secara terpisah dan mudah diubah tanpa memengaruhi sistem [25].

Arsitektur berbasis peristiwa (event-driven architecture) tidak berinteraksi langsung dengan layanan lain, melainkan menyimpan salinan lokal data mereka. Hal ini sering menyebabkan duplikasi data dan memerlukan sinkronisasi dengan mendengarkan perubahan, seperti penambahan atau modifikasi, pada layanan lain. Berikut ini adalah implementasi arsitektur berbasis peristiwa dalam penelitian ini.



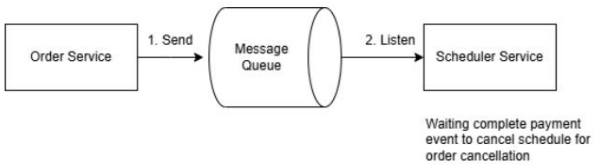
Gambar 4. Desain Alur Sistem Peristiwa Pembuatan Pesanan



Gambar 5. Desain Alur Sistem Peristiwa Setelah Pembuatan Pesanan

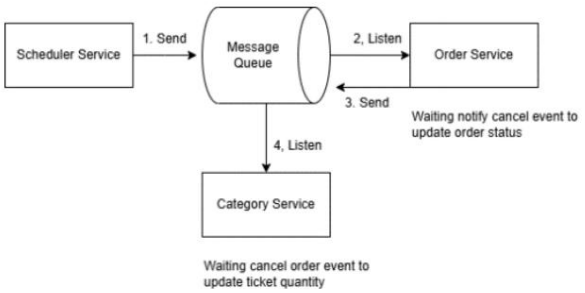
Gambar 4 mengilustrasikan proses pembuatan pesanan, di mana layanan pesanan mengirimkan pesan peristiwa ke antrian pesan. Selanjutnya, layanan lain, seperti penjadwal dan layanan kategori, menangkap data peristiwa yang terkait dengan pembuatan pesanan.

Selanjutnya, Gambar 5 menggambarkan peristiwa lanjutan yang berfokus pada sinkronisasi data setelah pembuatan pesanan. Proses ini melibatkan sinkronisasi data yang relevan, khususnya jumlah tiket. Penting untuk dicatat bahwa hanya layanan pemilik yang memiliki wewenang untuk memperbarui data domainnya, sehingga memastikan integritas data dan kontrol akses yang tepat.

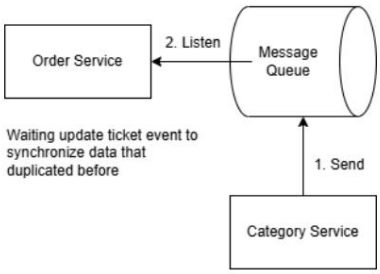


Gambar 6. Desain Alur Sistem Peristiwa Pesanan Lengkap

Selanjutnya, Gambar 6 mengilustrasikan proses penyelesaian pembayaran pesanan. Pada tahap ini, layanan mengirimkan pesan peristiwa ke antrian pesan, yang kemudian ditangkap oleh penjadwal layanan. Hal ini memungkinkan penjadwal untuk membatalkan pembatalan pesanan yang tertunda.



Gambar 7. Desain Alur Sistem Peristiwa Pembatalan Pesanan



Gambar 8. Desain Alur Sistem Peristiwa Setelah Pembatalan Pesanan

Terakhir, Gambar 7 mengilustrasikan proses pembatalan pesanan. Pada fase ini, penjadwal layanan mengirimkan pesan peristiwa ke antrian pesan, yang ditangkap oleh layanan pesanan untuk memulai pembaruan status.
Selanjutnya, layanan pemesanan mengirimkan pesan kejadian lain ke antrian pesan, yang kemudian ditangkap oleh layanan kategori untuk memperbarui jumlah tiket sesuai dengan itu. Selain itu, Gambar 8 mengilustrasikan proses yang mengikuti pembatalan pesanan, khususnya berfokus pada proses sinkronisasi data. Langkah ini memastikan bahwa semua sistem yang relevan diperbarui untuk mencerminkan perubahan yang dihasilkan dari pesanan yang dibatalkan.

2.4 Metrik Kinerja

Waktu respons dan tingkat kesalahan perangkat lunak adalah metrik kunci dalam rekayasa perangkat lunak yang secara signifikan memengaruhi kinerja dan keandalan aplikasi. Sementara waktu respons mengukur kecepatan sistem dalam memberikan umpan balik, tingkat kesalahan mencerminkan frekuensi cacat dalam perangkat lunak. Bersamaan dengan waktu respons, metrik P(90), atau waktu respons persentil ke-90, mengukur ambang batas di mana 90% permintaan diselesaikan.

Persentil ke-90 memberikan representasi yang lebih andal tentang kinerja sistem dalam kondisi beban tinggi, mengurangi pengaruh outlier ekstrem.
[26]. Metrik ini memberikan wawasan tentang pengalaman pengguna, menunjukkan berapa banyak pengguna yang mengalami waktu respons yang lebih lambat selama beban puncak. Tingkat kesalahan perangkat lunak, khususnya, menunjukkan seberapa sering cacat terjadi pada sistem.

2.5 Keterbatasan Penelitian dan Implikasi Manajerial

Terdapat beberapa keterbatasan dalam penelitian ini, khususnya terkait dengan perangkat keras dan aplikasi pengujian yang dikembangkan. Sistem tiket daring yang dibuat merupakan prototipe yang berfokus pada implementasi layanan dan alur setiap acara. Sistem ini belum memiliki
GUI dan belum diuji dengan pengguna sungguhan. Pengguna virtual dibuat menggunakan Grafana K6.

Perangkat keras yang digunakan untuk pengujian kinerja adalah komputer kelas konsumen, dengan spesifikasi yang dirinci dalam subbab Konfigurasi Pengujian. Hal ini mungkin berbeda dari kasus di dunia nyata di mana server yang mendukung aplikasi berkinerja tinggi pasti memiliki spesifikasi tinggi dengan kapasitas RAM yang signifikan.
Namun, pengujian dengan komputer standar kelas konsumen dianggap cukup dalam konteks membandingkan kinerja kedua arsitektur tersebut.

2.6 Konfigurasi Pengujian

Pengujian beban merupakan komponen penting dalam evaluasi kinerja, yang bertujuan untuk menentukan bagaimana suatu sistem berkinerja di bawah lalu lintas pengguna yang diantisipasi. Tujuan utamanya adalah untuk mengidentifikasi potensi hambatan, memastikan stabilitas sistem, dan mengkonfirmasi bahwa aplikasi mempertahankan tingkat kinerja di bawah beban yang diharapkan.

Grafana k6 adalah alat pengujian beban sumber terbuka yang dirancang untuk menilai kinerja aplikasi secara efektif. Alat ini memungkinkan pengembang untuk mensimulasikan interaksi pengguna dan mengevaluasi bagaimana aplikasi mengatasi berbagai kondisi beban. Alat ini sangat berguna untuk memastikan kinerja tinggi yang berkelanjutan sepanjang siklus hidup aplikasi [27].

Metodologi pengujian melibatkan pengiriman permintaan melalui Grafana K6 selama 30 detik, mensimulasikan seluruh proses penjualan tiket, termasuk melihat tiket, memesan, dan melakukan pembayaran. Selain itu, pengujian ini mencakup skenario kegagalan di mana 1/3 dari transaksi disimulasikan gagal karena tidak adanya pembayaran.

Dalam pengujian performa, "pengguna bersamaan" adalah metrik kunci untuk menilai seberapa baik suatu sistem mengelola interaksi simultan. Ini adalah pengguna virtual yang dibuat untuk berinteraksi dengan sistem secara bersamaan selama pengujian beban. Alat seperti k6 memfasilitasi simulasi dan pengelolaan pengguna virtual ini, memungkinkan pengujian stres komprehensif di bawah berbagai kondisi beban.

Ketika banyak pengguna virtual beroperasi secara bersamaan, mereka menghasilkan volume permintaan yang signifikan, sehingga menantang kemampuan sistem untuk menangani dan memprosesnya secara efektif. Pendekatan ini secara akurat mencerminkan skenario dunia nyata, seperti periode penggunaan puncak ketika banyak pengguna mengakses aplikasi secara bersamaan.

Requests Per Second (RPS) adalah metrik kinerja penting yang mengukur berapa banyak permintaan yang dapat ditangani oleh server atau layanan dalam satu detik. Ini sangat penting untuk menilai efisiensi dan skalabilitas aplikasi web, API, dan layanan jaringan. Pemahaman yang jelas tentang RPS

Memungkinkan pengembang untuk mengoptimalkan pengalaman pengguna dan mengalokasikan sumber daya secara lebih efektif. Dalam penelitian ini, kami menguji sistem dengan tingkat pengguna bersamaan sebanyak 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 550, 600, 650, dan 700 untuk mengevaluasi kinerjanya di bawah beban yang bervariasi.

Pendekatan ini memungkinkan kita untuk menilai seberapa baik sistem menangani peningkatan beban dan mengidentifikasi masalah kinerja apa pun. Tes kinerja dilakukan pada laptop yang dikonfigurasi untuk evaluasi menyeluruh di bawah berbagai beban, yang menampilkan prosesor Intel Core i5-8350U, RAM 24 GB, dan Fedora 40 sebagai sistem operasinya.

3. Hasil dan Diskusi

Bagian ini akan menyajikan hasil pengujian, diikuti dengan diskusi tentang temuan dan implikasinya. Kami akan menganalisis hasil pengujian kinerja yang dilakukan pada sistem e-ticketing, membandingkan hasilnya antara implementasi monolit dan microservice. Terakhir, kami akan menarik kesimpulan berdasarkan wawasan yang diperoleh dari analisis tersebut.

3.1 Hasil Tes

Kami melakukan pengujian dalam lingkungan operasional yang telah ditentukan dan dikonfigurasi khusus untuk pengujian beban, memastikan bahwa tidak ada operasi lain yang berjalan secara bersamaan. Pengujian ini dilakukan selama lima iterasi untuk meningkatkan konsistensi dan keandalan.
Hasilnya. Setelah menyelesaikan semua iterasi, kami merata-ratakan hasilnya untuk membangun pengukuran komprehensif yang secara akurat mencerminkan karakteristik kinerja konfigurasi yang sedang diselidiki.
Metodologi ini memperkuat validitas temuan kami dengan meminimalkan pengaruh potensi hasil yang menyimpang.

Data yang dikumpulkan mencakup berbagai metrik kinerja, seperti Permintaan Per Detik (RPS), yang diukur dalam permintaan per detik (req/s), waktu respons rata-rata, dan waktu respons persentil ke-90 (P90).
dinyatakan dalam milidetik (ms), dan tingkat kesalahan dinyatakan sebagai persentase.

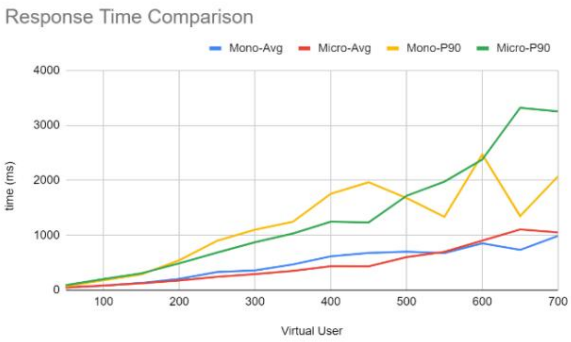
Berikut adalah format hasil pengujian yang akan digunakan nanti, seperti yang diilustrasikan pada Tabel 1. Bagian atas menyajikan hasil untuk arsitektur monolitik, sedangkan bagian bawah menguraikan hasil untuk arsitektur layanan mikro.

Tabel 1. Format Hasil Uji Beban

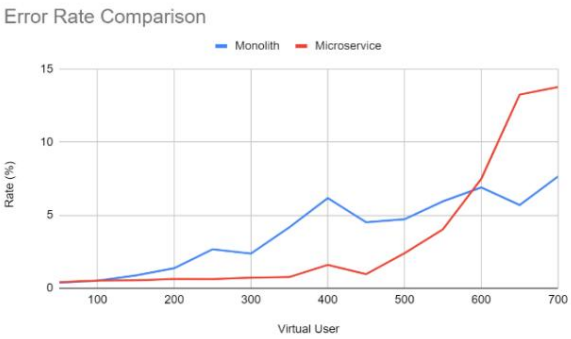
Pengguna	RPS	Rata-	P(90)	Tingkat Kesalahan
50	x1	rata	x3	x4
	y1	x2 y2	y3	

Gambar 9 menunjukkan perbandingan hasil berdasarkan metrik waktu respons, khususnya rata-rata dan persentil ke-90 (P90).

Berdasarkan metrik rata-rata dan persentil ke-90 (P90), grafik menunjukkan bahwa arsitektur microservices memiliki waktu respons yang lebih cepat daripada arsitektur monolitik. Pada awalnya. Seiring bertambahnya jumlah pengguna virtual, metrik waktu respons meningkat; namun, terdapat beberapa fluktuasi yang akan dibahas dalam sesi diskusi selanjutnya.



Gambar 9. Perbandingan Grafik Waktu Respons



Gambar 10. Perbandingan Grafik Tingkat Kesalahan

Tabel 2. Hasil Uji Beban

Pengguna	RPS	Rata-	P(90)	Tingkat Kesalahan
50	471		69	0.3
	453		89	0,4
100	699		177	0,5
	633		200	0,5
150	764	rata	284	0,8
	718	45	304	0,5
200	736	45	542	1.3
	735	80	486	0,6
250	649	82	896	1.3
	711	130	682	0,6
300	700	121	1099	2.3
	735	203	870	0,7
350	661	173	1242	4.1
	742	327	1030	0,7
400	581	240	1753	6.1
	737	357	1244	1.5
450	603	290 466 348 614 468 675	1468	4.5
	797	431	1232	0,9
500	613	699	1676	4.7
	680	599	1716	2.3
550	718	673	1332	5,94
	657	696	1974	4
600	638	859	2470	6.8
	582	899	2376	7.4
650	818	731	1346	5.6
	516	1102	3318	13.2
700	651	986	2070	7.6
	588	1049	3252	13.7

Berdasarkan metrik tingkat kesalahan, grafik perbandingan hasil yang diperoleh ditunjukkan pada Gambar 10. Grafik tersebut menggambarkan tingkat kesalahan yang diperoleh dari hasil pengujian,

menunjukkan bahwa arsitektur microservices umumnya menghasilkan tingkat kesalahan yang lebih rendah dibandingkan dengan arsitektur monolitik. Namun, penting untuk dicatat bahwa fluktuasi terjadi pada titik-titik tertentu dalam data tersebut.

Selanjutnya, hasil pengujian dan perbandingan disajikan dalam format tabel pada Tabel 2. Berdasarkan Tabel
2. Waktu respons rata-rata terendah yang tercatat untuk arsitektur microservices dan monolitik adalah 45 ms.
Waktu respons p90 terendah adalah 69 ms untuk arsitektur monolitik dan 89 ms untuk arsitektur mikroservis. Mengenai tingkat kesalahan, arsitektur monolitik menunjukkan tingkat kesalahan minimum 0,3%, sedangkan mikroservis menunjukkan tingkat kesalahan minimum 0,4%. Selanjutnya, waktu respons rata-rata maksimum untuk arsitektur monolitik adalah 986 ms, dibandingkan dengan 1,1 s untuk mikroservis. Waktu respons p90 maksimum untuk arsitektur monolitik adalah 2,5 s, sedangkan untuk mikroservis adalah 3,3 s. Terakhir, tingkat kesalahan maksimum untuk arsitektur monolitik adalah 7,6%, sedangkan mikroservis menunjukkan tingkat kesalahan maksimum 13%.

3.2 Diskusi

Berdasarkan hasil eksperimen yang dilakukan, perubahan signifikan diamati pada setiap skenario yang melibatkan jumlah pengguna bersamaan yang berbeda-beda, terutama ketika jumlah pengguna melebihi 450.
Data tersebut mengungkapkan pola fluktuasi yang tidak teratur, disertai dengan lonjakan yang mencolok.

Fluktuasi ini disebabkan oleh keterbatasan sumber daya komputasi, terutama CPU. Selama pengujian, pemanfaatan CPU mencapai rata-rata 90%, yang menunjukkan bahwa sistem beroperasi pada kapasitas maksimumnya. Situasi ini menyebabkan waktu respons yang lebih lambat selama proses pengujian dan peningkatan kesalahan.

Dampak fluktuasi ini bahkan lebih terasa pada arsitektur microservices. Model ini biasanya mengonsumsi lebih banyak sumber daya daripada arsitektur monolitik, karena setiap microservice dapat mengoperasikan beberapa instance, termasuk layanan, basis data, dan broker pesan. Akibatnya, dengan semakin banyak komponen yang berfungsi secara bersamaan, permintaan CPU meningkat, yang dapat berdampak negatif pada kinerja seiring bertambahnya jumlah pengguna.

Dari segi waktu respons, ketika jumlah pengguna bersamaan adalah 450 atau kurang, arsitektur microservices menunjukkan kinerja yang unggul.
Secara spesifik, sistem ini 36% lebih cepat daripada sistem monolit dalam hal waktu respons rata-rata dan 56% lebih cepat dalam metrik p90. Namun, ketika terdapat lebih dari 450 pengguna secara bersamaan, waktu respons untuk layanan mikro akan meningkat.
Dalam metrik p90, dibutuhkan waktu 25% lebih lama bagi microservices untuk merespons dibandingkan dengan monolit, dan rata-rata membutuhkan waktu 10% lebih lama.

Terkait tingkat kesalahan, ketika jumlah pengguna bersamaan mencapai 450 atau lebih, microservices mencapai tingkat kesalahan 71% lebih rendah dibandingkan dengan monolit.
Namun demikian, dalam skenario dengan lebih dari 450 pengguna, kesalahan tersebut Tingkat penggunaan microservices meningkat, menjadi 35% lebih tinggi daripada monolit.

Temuan penelitian ini konsisten dengan salah satu temuan dalam penelitian lain [15], khususnya mengenai keterbatasan skalabilitas yang berkontribusi pada penurunan kinerja aplikasi dalam arsitektur microservices. Arsitektur microservices membutuhkan lebih banyak sumber daya daripada arsitektur monolitik, karena dapat menjalankan beberapa instance secara paralel. Namun, kemampuan ini dapat menyebabkan penurunan drastis kinerja aplikasi dalam lingkungan beban tinggi. Hasil pengujian menunjukkan bahwa ketika jumlah pengguna bersamaan melebihi 450, terjadi peningkatan yang nyata pada tingkat kesalahan dan peningkatan waktu respons.

4. Kesimpulan

Hasil penelitian menunjukkan bahwa arsitektur microservices unggul dalam hal performa dan tingkat kesalahan pada beban sedang. Meskipun dirancang untuk menangani lalu lintas tinggi, tantangan terkait keterbatasan sumber daya dan overhead komunikasi menjadi jelas seiring meningkatnya beban. Hal ini tidak berarti bahwa microservices tidak dapat menangani jumlah pengguna yang tinggi, tetapi menyoroti perlunya peningkatan manajemen sumber daya.

Oleh karena itu, upaya optimasi dan pemantauan berkelanjutan sangat penting untuk memastikan kinerja optimal selama periode peningkatan aktivitas pengguna.

Ucapan Terima Kasih

Penelitian untuk artikel ini didanai oleh DIPA dari
Badan Layanan Umum Universitas Sriwijaya 2024.
Nomor SP DIPA: 023.17.2.677515/2024, tanggal 24 November 2023.
Sesuai dengan Keputusan Rektor nomor: 0013/UN9/LP2M.PT/2024, tanggal 20 Mei 2024.

Referensi

[1] R. Bolscher dan M. Daneva, “Merancang arsitektur perangkat lunak untuk mendukung pengiriman berkelanjutan dan DevOps: Tinjauan literatur sistematis,” dalam *ICSOFT 2019 - Prosiding Konferensi Internasional ke-14 tentang Teknologi Perangkat Lunak,* SciTePress, 2019, hlm. 27–39. doi: 10.5220/0007837000270039.

[2] FH Khoso, A.Lakhan, AA Arain, MA Soomro, SZ Nizamani, dan K. Kanwar, “Sistem Berbasis Layanan Mikro untuk Internet of Things Industri dalam Jaringan yang Dibantu Fog-Cloud”, *Eng. Technol. Appl. Sci. Res.,* vol. 11, no. 2, hlm. 7029–7032, April 2021. https://doi.org/10.48084/etasr.4077

[3] F. Dai, G. Liu, X. Xu, Q. Mo, Z. Qiang, dan Z. Liang, “Pemeriksaan kompatibilitas untuk sistem siber-fisik berbasis layanan mikro,” *Softw Pract Exp,* vol. 52, no. 11, hlm. 2393–2410, November 2022, doi: 10.1002/spe.3131.

[4] P. Di Francesco, P. Lago, dan I. Malavolta, “Merancang Arsitektur dengan Layanan Mikro: Sebuah Studi Pemetaan Sistematis,” *Jurnal Sistem dan Perangkat Lunak,* vol. 150, hlm. 77–97, April 2019, doi: 10.1016/j.jss.2019.01.001.

[5] S. Hassan, R. Bahsoon, dan R. Buyya, “Analisis Skalabilitas Sistematis untuk Keputusan Desain Adaptasi Granularitas Layanan Mikro,” *Softw Pract Exp,* 2022, 52(6): 1378–1401, doi: 10.1002/spe.3069.

[6] JH Duarte Correia dan AR Silva, “Identifikasi Refactoring Fungsionalitas Monolit untuk Layanan Mikro”

Migrasi," *Softw Pract Exp,* 2022, 52(12): 2664–2683, doi: 10.1002/spe.3141.

[7] R. Gebler, "Mendukung Manajemen Pandemi Regional dengan Memungkinkan Pelaporan Mandiri—Sebuah Laporan Kasus," *PLoS One,* 2024, 31;19(1):e0297039, doi: 10.1371/journal.pone.0297039.

[8] J. Kazanaviÿius dan D. Mažeika, "Evaluasi Komunikasi Layanan Mikro Saat Menguraikan Monolit", *Comput. Inform.,* vol. 42, no. 1, hlm. 1–36, Mei 2023. https://doi.org/10.31577/cai_2023_1_1

[9] FH Vera-Rivera, C. Gaona, dan H. Astudillo, "Mendefinisikan dan mengukur granularitas microservice—tinjauan literatur," *PeerJ Comput Sci,* vol. 7, hlm. e695, Sep. 2021, doi: 10.7717/peerj-cs.695.

[10] B. Salles dan J. Cunha, "Dekomposisi Monolit ke Mikroservis dengan Bantuan Visual," *Simposium IEEE 2023 tentang Bahasa Visual dan Komputasi Berpusat pada Manusia (VL/HCC),* Washington, DC, AS, 2023, hlm. 293-295, doi: 10.1109/VL-HCC57772.2023.00057.

[11] H.-P. Huang, Y.-Y. Fanjiang, C.-H. Hung, H.-F. Tsai, dan B.-H. Lin, "Evaluasi Sistem Mikroservis Interkom Cerdas Berbasis Cloud of Things," *Elektronik (Basel),* 2023, vol 12, no. 11 hal. 2406, doi: 10.3390/electronics12112406.

[12] S. Hassan *dkk.,* "Dari Monolit ke Mikroservis: Arsitektur Perangkat Lunak untuk Inspeksi Infrastruktur UAV Otonom," *Softw Pract Exp,* vol. 52, 2022, doi: 10.14569/ijacsa.2017.081236.

[13] J. Kazanavicius *dkk.,* "Identifikasi Layanan Mikro dengan Mempartisi Aplikasi Web Monolitik Berdasarkan Kasus Penggunaan," *Softw Pract Exp,* 2022, doi: 10.1002/spe.3141.

[14] AJ Lauwren, "Perbandingan Kinerja Microservice dan Monolith dalam Aplikasi Transaksi," *Jurnal Informatika,* 2024, doi: 10.24167/proxies.v5i2.12447.

[15] G. Blinowski, A. Ojdowska, dan A. Przybylek, "Arsitektur Monolitik vs. Arsitektur Mikroservis: Evaluasi Kinerja dan Skalabilitas," *IEEE Access,* vol. 10, hlm. 20357–20374, 2022, doi: 10.1109/ACCESS.2022.3152803.

[16] A. Barczak dan M. Barczak, "Perbandingan kinerja aplikasi berbasis monolit dan mikroservis.", *Konferensi Multi-Dunia ke-25 tentang Sistemik, Sibernetika dan Informatika, WMSCI 2021,* vol. 1, hlm. 120–125. Institut Internasional Informatika dan Sistemik, IIIS.

[17] M. Seedat, Q. Abbas, dan N. Ahmad, "Pemetaan Sistematis Aplikasi Monolitik ke Arsitektur Layanan Mikro," September 2023, [Online]. Tersedia: http://arxiv.org/abs/2309.03796

[18] CH Kristantyo, IA Putranto, ES Soegoto, R. Setiawan, dan R. Jumansyah, "Dampak Penerapan E-Ticketing Terhadap Transportasi Bus di Bandung," *Kne Social Sciences,* 2020, doi: 10.2991/aebmr.k.200108.008.

[19] N. Bumanis, G. Vitols, I. Arhipova, dan I. Mozga, "Manajemen Siklus Hidup Tiket Seluler: Studi Kasus Transportasi Publik Latvia," doi: Transport 2017, 10.22616/erdev2017.16.n015.

[20] J. Jordanov dan SK Jaiswal, "Pendekatan Desain Berbasis Domain dalam Arsitektur Layanan Cloud Native," *Jurnal Internasional Penelitian Inovatif di bidang Teknik & Manajemen,* 2023, doi: 10.18421/tem124-09.

[21] SK Jaiswal, "Domain-Driven Design (DDD)- Menjembatani Kesenjangan Antara Persyaratan Bisnis dan Pemodelan Berorientasi Objek," *International Journal of Innovative Research in* 2024, doi: *Engineering Management,* ijirem.2024.11.2.16. & 10.55524/

[22] R. Mikkilineni, "Kelas Baru Mesin Cerdas Dengan Alur Proses Berbasis Peristiwa yang Mengatur Diri Sendiri untuk Mendesain, Menerapkan, dan Mengelola Aplikasi Perangkat Lunak Terdistribusi," 2023, doi: 10.20944/preprints202311.1104.v1.

[23] K. Farias dan L. Lazzari, "Arsitektur Berbasis Peristiwa dan Gaya Arsitektur REST: Sebuah Studi Eksplorasi tentang Modularitas," *Jurnal Penelitian dan Teknologi Terapan,* 2023, doi: 10.22201/icat.24486736e.2023.21.3.1764.

[24] A. Rahmatulloh, F. Nugraha, R. Gunawan, dan I. Darmawan, "Arsitektur Berbasis Peristiwa untuk Meningkatkan Kinerja dan Skalabilitas dalam Sistem Berbasis Layanan Mikro," 2022, doi: 10.1109/icadeis56544.2022.10037390.

[25] Ayoubi, "Pendekatan Arsitektur Berorientasi Layanan Berbasis Peristiwa untuk Sistem E-Governance," *Kjet,* 2019, doi: 10.31841/kjet.2021.1.

[26] R. Bhattacharya dan T. Wood, "BLOC: Menyeimbangkan Beban dengan Kontrol Kelebihan Beban dalam Arsitektur Layanan Mikro," dalam *Konferensi Internasional IEEE tentang Komputasi Otonom dan Sistem Pengorganisasian Mandiri (ACSOS),* 2022, hlm. 91–100. doi: 10.1109/ACSOS55765.2022.00027.

[27] KG Sukadharma, "Implementasi CI/CD Pada Microservices Untuk Meningkatkan Availability Pada Pemrosesan Big Data," *Jeliku (Jurnal Elektronik Ilmu Komputer Udayana),* 2024, doi: 10.24843/jlk.2023.v12.i03.p12.