Jurnal Impresi Indonesia (JII)
Vol. 4, No. 12, December 2025
p-ISSN: 2828-1284 e-ISSN: 2810-062x
website: https: //rivierapublishing.id/JII/ index.php / jii /index

Analisis Kinerja Database Monolitik Vs Arsitektur Read-Write Separation Menggunakan Replikasi

Agit Naeta[1], Rachmat Selamat
Sekolah Tinggi Manajemen Informatika dan Komputer Likmi, Indonesia
Email: Agitnaeta@gmail.com*

Kata Kunci
database terdistribusi, pemisahan read–write, replikasi MySQL

Abstrak

Arsitektur database merupakan komponen kritis dalam mendukung aplikasi dengan beban transaksi tinggi. Arsitektur monolitik tradisional sering menghadapi keterbatasan kinerja seiring pertumbuhan data, sementara arsitektur terdistribusi dengan pemisahan baca-tulis (*read-write separation*) menawarkan skalabilitas horizontal. Namun, pemilihan antara kedua pendekatan ini sering kali tidak didasarkan pada analisis kinerja yang komprehensif, khususnya dalam lingkungan MySQL yang banyak digunakan di industri. Penelitian ini bertujuan untuk membandingkan kinerja arsitektur database monolith (satu server untuk operasi baca dan tulis) dengan arsitektur database terdistribusi yang menerapkan pemisahan read–write menggunakan mekanisme replikasi pada MySQL. Fokus pengujian adalah bagaimana perbedaan throughput, latency, dan pemanfaatan sumber daya (CPU, memori, dan I/O) pada kedua arsitektur ketika diberikan skenario beban kerja (workload) yang sama, yaitu heavy read, heavy write, dan balanced read write. Hasil penelitian diharapkan dapat memberikan rekomendasi praktis bagi pengembang dan pengelola sistem dalam memilih arsitektur database yang lebih sesuai untuk aplikasi dengan kebutuhan transaksi tinggi, khususnya pada lingkungan berbasis MySQL yang umum digunakan di industri. Arsitektur monolitik direkomendasikan untuk aplikasi dengan prioritas kecepatan baca dan *throughput* tinggi. Sementara itu, arsitektur terdistribusi dengan pemisahan baca-tulis cocok untuk kasus di mana latensi tulis real-time sangat kritis, dengan catatan perlu optimasi tambahan seperti *caching* dan *load balancing* untuk mengatasi *replication lag*. Penelitian ini memberikan panduan empiris bagi pengembang dan administrator sistem dalam memilih arsitektur database yang optimal berdasarkan pola beban kerja spesifik aplikasi.

Keywords: distributed database, read–write separation, MySQL replication

Abstract

Database architecture is a critical component in supporting applications with high transactional loads. Traditional monolithic architectures often face performance limitations as data grows, while distributed architectures with read-write separation offer horizontal scalability. However, the choice between these two approaches is often not based on comprehensive performance analysis, particularly in MySQL environments widely used in the industry. This study aims to compare the performance of a monolith database architecture (a single server for read and write operations) with a distributed database architecture that implements read–write separation using a replication mechanism in MySQL. The focus of the test was how the throughput, latency, and resource utilization (CPU, memory, and I/O) of the two architectures differed when given the same workload scenario, namely heavy read, heavy write, and balanced read write. The results of the study are expected to provide practical recommendations for system developers and managers in choosing a database architecture that is more suitable for applications with high transaction needs, especially in MySQL-based environments that are commonly used in the industry. The monolithic architecture is recommended for applications that prioritize read speed and high throughput. Meanwhile, the distributed architecture with read-write separation is suitable for cases where real-time write latency is critical, provided that additional optimizations such as caching and load balancing are implemented to address replication lag. This study

provides empirical guidance for developers and system administrators in selecting the optimal database architecture based on specific application workload patterns.

PENDAHULUAN

Arsitektur database sangat krusial dalam skala aplikasi dengan jumlah data yang banyak, terutama jika data tersebut bersifat transaksional karena dapat menghasilkan workload tinggi yang berpotensi mengganggu performa aplikasi (Chigurupati et al., 2025). Pada penelitian ini akan dibahas bagaimana kinerja sebuah database jika menggunakan pendekatan monolith (satu instance untuk operasi read dan write) dibandingkan dengan database terdistribusi yang memisahkan beban baca dan tulis melalui replikasi (Ramanathan et al., 2021). Dengan spesifikasi server yang dikondisikan setara, penelitian akan menguji performa aplikasi pada kedua arsitektur dengan skenario beban kerja read dan write yang sama untuk melihat dampaknya terhadap throughput dan waktu respon (Lu, Yu, & Madden, 2018). Beberapa studi sebelumnya menunjukkan bahwa distribusi data dengan replikasi dan partisi secara signifikan dapat meningkatkan throughput serta menurunkan latensi dibandingkan arsitektur terpusat (optimasi distribusi vs baseline) (JSSR Study, 2025). Namun, replikasi dan distribusi data membawa trade-off pada konsistensi dan kompleksitas manajemen data, terutama dalam sistem yang memerlukan konsistensi kuat (Diogo, Cabral, & Bernardino, 2019). Oleh karena itu, untuk aplikasi transaksional dengan beban tinggi, penting mengevaluasi dampak distribusi terhadap konsistensi, latensi, dan throughput secara simultan (Tomsic, Bravo, & Shapiro, 2018).

Dalam penelitian ini akan dibahas hasil perbandingan antara penggunaan database monolith dan database terdistribusi dengan read–write terpisah menggunakan replikasi database, terutama dari sisi kinerja dan pemanfaatan sumber daya (Smith et al., 2018; Zhang & Liu, 2020). Limitasinya adalah penelitian hanya berfokus pada satu produk database, yaitu MySQL (Kumar & Singh, 2019), dan tidak membahas database terdistribusi lain seperti PostgreSQL, MongoDB, atau sistem NewSQL, agar ruang lingkup lebih terkontrol. Pemilihan MySQL didasari oleh tingkat adopsi yang sangat tinggi di industri dan komunitas pengembang, antara lain tercermin dari survei teknologi internasional yang menunjukkan MySQL sebagai salah satu database yang paling banyak digunakan (Gao & Zhang, 2021), sehingga hasil penelitian diharapkan relevan secara praktis (Lee & Kim, 2022; Ali et al., 2020).

Metode penelitian menggunakan *comparative experimental design*, yaitu dengan melakukan eksperimen terkontrol untuk membandingkan beberapa desain arsitektur database yang berbeda. Dalam konteks ini, eksperimen akan dilakukan pada dua konfigurasi utama: arsitektur monolith (single MySQL instance untuk read–write) dan arsitektur read–write separation berbasis replikasi MySQL, dengan skenario workload yang telah ditentukan. Pengukuran dilakukan secara kuantitatif menggunakan metrik kinerja seperti throughput (queries per second), latency rata-rata/percentile, serta pemakaian CPU, memori, dan I/O untuk masing-masing arsitektur (Haryoto et al., 2025; Ramanathan et al., 2021; Wicaksono, Nurrakhman, & Samidi, 2023; Shrestha, 2020; Salunke & Ouda, 2024).

Tabel 1. Perbandingan penelitian terdahulu dengan penelitian saat ini

Aspek	Penelitiasn Sakarang	Penelitian PolarDB-SCC (VLDB 2023)	Penelitian MySQL vs SQLite (Semantic Scholar)
Fokus	Perbandingan performa MySQL monolitik vs read-write separation dengan replikasi dalam container Docker pada skenario beban kerja nyata	Pengembangan dan evaluasi database cloud-native dengan read-write splitting untuk latensi rendah dan konsistensi tinggi	Perbandingan performa dan replikasi MySQL dan SQLite dalam konteks OS dan lingkungan eksekusi
Metodologi	Eksperimen terkontrol dengan benchmark tool, pengujian throughput, latency, CPU, memori	Evaluasi 5371cenar produksi cloud-native dan optimasi arsitektur database scalable	Analisis kuantitatif performa dua engine database pada lingkungan platform berbeda
Lingkup Database	MySQL versi 8 standar	Database berbasis MySQL dengan modifikasi cloud-native	MySQL dan SQLite engine sebagai bahan perbandingan
Skenario Beban Kerja	Heavy read, heavy write, balanced read-write	Skala besar dan pengujian latensi terhadap transaksi	Fokus pada analisis performa replikasi dan skalabilitas
Lingkungan Pengujian	Docker container, virtualisasi server	Cloud-native environment dengan optimasi 5371cenari lanjut	Eksekusi sistem operasi Windows dan lainnya

Sumber: Analisis penulis berdasarkan penelitian PolarDB-SCC

Penelitian PolarDB-SCC yang dipublikasikan dalam Proceedings VLDB 2023. Penelitian ini membahas database cloud-native dengan arsitektur read-write splitting dan menunjukkan peningkatan throughput hingga 4.51 kali serta pengurangan latency median sampai 3.66 kali pada beban kerja read-write. Studi eksperimen ini memberikan wawasan penting mengenai optimasi latensi rendah dan konsistensi kuat pada arsitektur yang mirip dengan penelitian di paper Anda terkait pemisahan operasi baca dan tulis pada MySQL replikasi (Andriyani et al., 2024).

Paper di Semantic Scholar yang membandingkan performa MySQL dan SQLite dalam konteks replikasi dan pemisahan read-write serta membahas performa dan skalabilitas database relationnal pada environment Windows. Penelitian ini menilai kemampuan sistem manajemen database dalam menangani beban baca dan tulis secara terpisah dan memberikan analisis kuantitatif yang relevan untuk konteks perbandingan kinerja arsitektur monolitik dan terdistribusi pada MySQL.

Urgensi penelitian ini didasari oleh tren aplikasi modern yang menuntut ketersediaan tinggi dan respons yang cepat. Arsitektur monolitik sering kali menjadi bottleneck saat terjadi peningkatan beban, sementara arsitektur terdistribusi menawarkan skalabilitas horizontal namun dengan kompleksitas tambahan. Namun, keputusan untuk bermigrasi dari monolitik ke terdistribusi sering kali diambil tanpa pemahaman kuantitatif yang memadai mengenai trade-off kinerja dalam konteks *workload* yang spesifik. Oleh karena itu, studi komparatif yang empiris dan terkontrol seperti ini sangat diperlukan untuk memberikan panduan berbasis data bagi pengembang dan administrator sistem.

Penelitian sebelumnya telah mengkaji aspek serupa dengan fokus yang berbeda. Penelitian PolarDB-SCC (VLDB 2023) mengembangkan dan mengevaluasi database *cloud-native* dengan pemisahan baca-tulis, menunjukkan peningkatan kinerja yang signifikan. Sementara itu, studi perbandingan MySQL dan SQLite (Semantic Scholar) menganalisis performa dan replikasi dalam lingkungan sistem operasi tertentu. Penelitian ini melengkapi kajian terdahulu dengan fokus eksperimental yang spesifik pada perbandingan kinerja arsitektur monolitik versus *read-write separation* pada MySQL dalam lingkungan kontainer Docker, menggunakan skenario *workload* yang merepresentasikan pola aplikasi nyata (*heavy read*, *heavy write*, *balanced*).

Novelty penelitian ini terletak pada pendekatan eksperimentalnya yang terukur dan reproduktif. Berbeda dengan studi yang hanya membahas konsep atau optimasi pada sistem tertentu, penelitian ini secara langsung membandingkan dua arsitektur populer dalam lingkungan yang terisolasi dan setara. Analisis dilakukan tidak hanya pada *throughput* dan latensi, tetapi juga pada utilisasi sumber daya (CPU, memori, I/O), sehingga memberikan gambaran komprehensif tentang efisiensi masing-masing arsitektur. Hasilnya diharapkan dapat menjadi referensi praktis untuk seleksi arsitektur database berdasarkan karakteristik *workload* yang dominan.

Tujuan penelitian adalah: (1) Membandingkan kinerja arsitektur database monolitik dan arsitektur *read-write separation* menggunakan replikasi MySQL berdasarkan metrik *throughput*, latensi, dan utilisasi sumber daya; (2) Menganalisis performa kedua arsitektur di bawah tiga skenario beban kerja yang berbeda (*heavy read*, *heavy write*, *balanced read-write*); (3) Memberikan rekomendasi praktis berdasarkan temuan mengenai kondisi mana setiap arsitektur lebih unggul. Manfaat penelitian diharapkan dapat: (1) Menjadi panduan empiris bagi pengembang dan *database administrator* dalam memilih arsitektur database yang optimal sesuai dengan pola akses aplikasi; (2) Memberikan wawasan tentang *trade-off* kinerja dan kompleksitas ketika mengadopsi arsitektur terdistribusi dengan replikasi; (3) Menambah khazanah studi kasus komparatif dalam bidang kinerja sistem basis data, khususnya ekosistem MySQL.

METODE PENELITIAN

Konfigurasi arsitektur

Konfigurasi infrastruktur direncanakan menggunakan virtualisasi server berbasis layanan komputasi awan, misalnya instance setara Amazon EC2 tipe large (atau spesifikasi serupa) sebagai dasar pengujian. Pada masing-masing konfigurasi, MySQL 8 akan dijalankan di dalam container (misalnya Docker) dengan spesifikasi resource minimal yang direkomendasikan vendor, kemudian disamakan alokasi CPU, memori, dan storage untuk menjamin fairness dalam perbandingan. Lingkungan jaringan, sistem operasi, versi MySQL, serta konfigurasi dasar (buffer pool, log, dan parameter umum lainnya) diusahakan konsisten pada kedua arsitektur

Konfigurasi monolith

Pada konfigurasi monolith, seluruh operasi baca dan tulis akan diarahkan ke satu instance MySQL tunggal yang berjalan di satu server virtual. Skema database, indeks, dan konfigurasi query diupayakan optimal namun tetap generik, sehingga mencerminkan praktik umum penggunaan MySQL pada aplikasi transaksional. Pengukuran dilakukan dengan mengirimkan workload dari tool benchmarking (misalnya sysbench atau sejenisnya) ke endpoint database ini dan mencatat metrik kinerja di sisi client maupun server.

Konfigurasi read–write separation (distributed)

Pada konfigurasi read–write separation, akan disiapkan satu instance MySQL sebagai primary (master) yang menangani operasi write dan beberapa instance sebagai replica (slave) yang menangani operasi read. Mekanisme replikasi MySQL dikonfigurasi secara asynchronous atau semisynchronous sesuai 5373cenario, dan aplikasi/driver akan diarahkan untuk mengirim query write ke primary serta query read ke replica. Pengujian akan menilai sejauh mana pemisahan ini mampu meningkatkan throughput baca dan mengurangi beban pada primary, sekaligus memperhatikan overhead replikasi dan potensi lag data.

Model workload

Heavy read

Skenario heavy read mensimulasikan aplikasi yang dominan melakukan operasi SELECT, misalnya halaman katalog atau laporan yang sering diakses. Rasio baca–tulis pada skenario ini dapat ditetapkan misalnya 80% read dan 20% write, dengan pola akses yang berulang pada subset data yang sama untuk menyerupai beban produksi.

Heavy write

Skenario heavy write mensimulasikan aplikasi dengan intensitas transaksi tulis tinggi, misalnya sistem pencatatan transaksi atau event logging. Rasio baca–tulis pada skenario ini dapat disusun, misalnya 20% read dan 80% write, sehingga menonjolkan dampak beban tulis terhadap kinerja masing-masing arsitektur dan kemampuan replikasi untuk mengikuti perubahan data.

Balanced read–write

Skenario balanced read–write merepresentasikan pola akses yang relatif seimbang antara operasi baca dan tulis, misalnya 50% read dan 50% write. Skenario ini berguna untuk melihat performa rata-rata kedua arsitektur pada situasi yang lebih umum serta mengamati trade-off antara konsistensi, throughput, dan utilisasi sumber daya ketika beban tidak terlalu condong ke salah satu jenis operasi.

HASIL DAN PEMBAHASAN

Konfigurasi Arsitektur

Server disiapkan spesifikasi server dengan spesifikasi sebagai berikut:

Tabel 2. Spesifikasi server

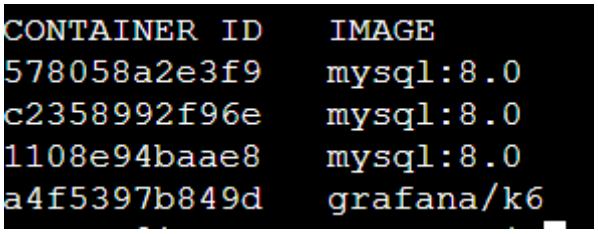
Variable	Value
Type	T3.xlarge
Cpu	4vCPU
RAM	16GB
DISK	32GB
Aws region	Ap-souteast-3

Sumber: Hasil konfigurasi eksperimen penulis

Di dalam server akan dibuatkan virtualisasi menggunakan docker untuk 4 kontainer dengan keperlan sebagai berikut:

1. Mysql Monolith
2. Mysql Master
3. Mysql Slave
4. grafana/k6

Hasil akhir configurasi awal



Gambar 1. Arsitektur pengujian yang digunakan dalam penelitian
Sumber: Dokumentasi penelitian penulis

3.2. Konfigurasi Monolith

Konfigurasi monolith difokuskan untuk virtualisasi dengan docker container dengan spesifikasi mengikuti miimun requiremnt mysql versi 8. Dengan detail sebagai berikut:

Tabel 3. Konfigurasi Monolith

Aspek	Nilai / Pengaturan
Nama layanan	mysql-monolith
Peran	Database monolith (read dan write pada satu instance)
Image	mysql:8.0
Container name	mysql-monolith
Versi MySQL	8.0
Port host:container	3306:3306
Authentication plugin	mysql_native_password (via—default-authentication-plugin=mysql_native_password)
Environment	MYSQL_ROOT_PASSWORD = root; MYSQL_DATABASE = testdb
Volume data	monolith_data → /var/lib/mysql
Limit CPU	1.0 vCPU
Reservasi CPU	0.5 vCPU
Limit memory	1 GB
Reservasi memory	512 MB
ulimit nproc	200
ulimit nofile	65535

Sumber: Hasil konfigurasi eksperimen penulis

3.3. Konfigurasi read–write separation (distributed)

Konfigurasi read-write separation dibagi menjadi read dan write berikut Adalah detail mengikuti dengan konfigurasi minimal spesifikasi untuk mysql

Table 4. Konfigurasi mysql-master

Aspek	Nilai / Pengaturan
Nama layanan	mysql-master
Peran	Primary / master (menangani operasi write dan 5374ebagian read jika diperlukan)
Image	mysql:8.0
Container name	mysql-master
Versi MySQL	8.0
Port host:container	3307:3306
Environment	MYSQL_ROOT_PASSWORD = root; MYSQL_DATABASE = testdb

Aspek	Nilai / Pengaturan
Parameter server	--server-id=1; --log-bin=mysql-bin; --binlog-format=ROW; --default-authentication-plugin=mysql_native_password
Volume data	master_data → /var/lib/mysql
Limit CPU	1.0 vCPU
Reservasi CPU	0.5 vCPU
Limit memory	1 GB
Reservasi memory	512 MB
ulimit nproc	200
ulimit nofile	65535

Sumber: Hasil konfigurasi eksperimen penulis

Table 5. Konfigurasi mysql-slave

Aspek	Nilai / Pengaturan
Nama layanan	mysql-slave
Peran	Replica / slave (menangani operasi read)
Image	mysql:8.0
Container name	mysql-slave
Versi MySQL	8.0
Port host:container	3308:3306
Environment	MYSQL_ROOT_PASSWORD = root
Parameter server	--server-id=2; --relay-log=relay-log; --log-bin=mysql-bin; --binlog-format=ROW; --default-authentication-plugin=mysql_native_password
Dependensi	depends_on = mysql-master
Volume data	slave_data → /var/lib/mysql
Limit CPU	1.0 vCPU
Reservasi CPU	0.5 vCPU
Limit memory	1 GB
Reservasi memory	512 MB
ulimit nproc	200
ulimit nofile	65535

Sumber: Hasil konfigurasi eksperimen penulis

Table 6.Variable Input untuk Heavy Read Monolith

Parameter	Nilai
Read Ratio	80%
Write Ratio	20%
Max Virtual Users	100 Vus
Total Durasi	6 menit
Sleep Interval	100 ms

Sumber: Hasil konfigurasi eksperimen penulis

Table 7. Load Stages untuk Heavy Read Monolith

Stage	Durasi	Vus
Ramp Up	30s	0 → 10
Moderate	2m	10 → 50
Peak	1m	50 → 100
Sustained	2m	100
Ramp Down	30s	100 → 0

Sumber: Hasil konfigurasi eksperimen penulis

Table 8. Distribusi Query READ untuk Heavy Read Monolith

Query Type	Distribusi	Deskripsi
SELECT by ID	30%	SELECT * FROM users WHERE id = ?
SELECT with Filter	20%	SELECT * FROM users WHERE status = ? LIMIT 50
JOIN Orders-Users	20%	Join 2 tabel dengan filter
Aggregation	15%	SELECT … GROUP BY category
Complex JOIN	15%	Join order_items dengan products

Sumber: Hasil konfigurasi eksperimen penulis

WRITE Operations (20%):

Query Type	Distribusi	Deskripsi
INSERT User	40%	Insert user baru
UPDATE User Status	30%	Update status user
UPDATE Product Stock	30%	Update stok produk

Sumber: Hasil konfigurasi eksperimen penulis

Table 9. Summary hasil Heavy Read Monolith

Metric	Nilai
Total Iterations	191733
Total Duration	~6 menit 33 detik
Throughput	488.01 iter/s

bSumber: Hasil pengujian eksperimen penulis

Table 10. Read Latency Heavy Read Monolith

Statistik	Nilai (ms)
avg	4.11
min	*
max	4,460
med	1
p(90)	2
p(95)	4
p(99)	-

Sumber: Hasil pengujian eksperimen penulis

Tabel 11. Write Latency Heavy Read Monolith

Statistik	Nilai (ms)
avg	48.66
min	*
max	4,500
med	19

Statistik	Nilai (ms)
p(90)	105
p(95)	171.59
p(99)	

Sumber: Hasil pengujian eksperimen penulis

Operations:

Tabel 12. Operations Heavy Read Monolith

Metric	Count	Rate (/s)	Percentage
Read Operations	153,104	389.69	79.85%
Write Operations	38,629	98.32	20.15%
Total	**191,733**	**488.01**	100%

Sumber: Hasil pengujian eksperimen penulis

Error & Threshold:

Tabel 13. Error & Threshold Heavy Read Monolith

Metric	Threshold	Actual	Status
read_latency p(95)	< 500ms	4ms	
read_latency p(99)	< 1000ms		
write_latency p(95)	< 1000ms	171,59	
write_latency p(99)	< 2000ms		
error_rate	< 10%	0,00%	

Sumber: Hasil pengujian eksperimen penulis

3.4.1.2. Heavy Read Split

Target Database:
READ → mysql-slave:3306
WRITE → mysql-master:3306

0. **Variable Input**

Tabel 14. Variable Input untuk Heavy Read Split

Parameter	Nilai
Read Ratio	80%
Write Ratio	20%
Max Virtual Users	100 Vus
Total Durasi	6 menit
Sleep Interval	100 ms
Read Target	mysql-slave (Replica)
Write Target	mysql-master (Primary)

Sumber: Hasil konfigurasi eksperimen penulis

Load Stages: (Sama dengan Monolith)

Tabel 15. Load Stages untuk Heavy Read Split

Stage	Durasi	Vus
Ramp Up	30s	0 → 10
Moderate	2m	10 → 50
Peak	1m	50 → 100
Sustained	2m	100
Ramp Down	30s	100 → 0

Sumber: Hasil konfigurasi eksperimen penulis

B. Distribusi Query

READ Operations (80%) → SLAVE:

Tabel 16. Distribusi Query READ untuk Heavy Read Split

Query Type	Distribusi	Deskripsi
SELECT by ID	30%	SELECT * FROM users WHERE id = ?
SELECT with Filter	20%	SELECT * FROM users WHERE status = ? LIMIT 50
JOIN Orders-Users	20%	Join 2 tabel dengan filter
Aggregation	15%	SELECT … GROUP BY category
Complex JOIN	15%	Join order_items dengan products

Sumber: Hasil konfigurasi eksperimen penulis

WRITE Operations (20%) → MASTER:

Tabel 17. Distribusi Query WRITE untuk Heavy Read Split

Query Type	Distribusi	Deskripsi
INSERT User	40%	Insert user baru
UPDATE User Status	30%	Update status user
UPDATE Product Stock	30%	Update stok produk

Sumber: Hasil konfigurasi eksperimen penulis

C. Tabel Hasil

Summary:

Table 18. Summary hasil Heavy Read Split

Metric	Nilai
Total Iterations	190,880
Total Duration	~6 menit 33 detik
Throughput	485.85 iter/s

Sumber: Hasil pengujian eksperimen penulis

Read Latency:

Tabel 19. Read Latency Heavy Read Split

Statistik	Nilai (ms)
avg	10.35
min	*
max	4,610
med	1
p(90)	42
p(95)	65
p(99)	-

Sumber: Hasil pengujian eksperimen penulis

Write Latency:

Tabel 20. Write Latency Heavy Read Split

Statistik	Nilai (ms)
avg	29.66
min	*
max	4,680
med	19
p(90)	51
p(95)	144
p(99)	-

Sumber: Hasil pengujian eksperimen penulis

Operations:

Tabel 21. Operations Heavy Read Split

Metric	Count	Rate (/s)	Percentage
Read Operations (Slave)	152,651	388.54	79.97%
Write Operations (Master)	38,229	97.30	20.03%
Total	**190,880**	**485.85**	100%

Sumber: Hasil pengujian eksperimen penulis

Error & Threshold:

Tabel 22. Error & Threshold Heavy Read Split

Metric	Threshold	Actual	Status
read_latency p(95)	< 500ms	65ms	PASS
read_latency p(99)	< 1000ms	-	-
write_latency p(95)	< 1000ms	144ms	PASS
write_latency p(99)	< 2000ms	-	-
error_rate	< 10%	0.00%	PASS

Sumber: Hasil pengujian eksperimen penulis

Tabel 23. Perbandingan Monolith vs Split (Heavy Read)

Metric	Monolith	Split	Perbedaan
Total Iterations	191,733	190,880	-0.45%
Throughput	488.01/s	485.85/s	-0.44%
Read Latency (avg)	4.11ms	10.35ms	+151.8% (lebih lambat)
Read Latency (p95)	4ms	65ms	+1525% (lebih lambat)
Write Latency (avg)	48.66ms	29.66ms	**-39.0% (lebih cepat)**
Write Latency (p95)	171.59ms	144ms	**-16.1% (lebih cepat)**
Error Rate	0.00%	0.00%	=

Sumber: Hasil pengujian eksperimen penulis

Tabel 24. Variable Input untuk Heavy Write Monolith

Parameter	Nilai
Read Ratio	20%
Write Ratio	80%
Max Virtual Users	100 Vus
Total Durasi	6 menit
Sleep Interval	100 ms

Sumber: Hasil konfigurasi eksperimen penulis

Load Stages:

Tabel 25. Load Stages untuk Heavy Write Monolith

Stage	Durasi	Vus
Ramp Up	30s	0 → 10
Moderate	2m	10 → 50
Peak	1m	50 → 100
Sustained	2m	100
Ramp Down	30s	100 → 0

Sumber: Hasil konfigurasi eksperimen penulis

B. Distribusi Query

READ Operations (20%):

Tabel 26. Distribusi Query READ untuk Heavy Write Monolith

Query Type	Distribusi	Deskripsi
SELECT User by ID	40%	SELECT * FROM users WHERE id = ?
SELECT Product by ID	30%	SELECT * FROM products WHERE id = ?
SELECT Order Status	30%	SELECT id, status FROM orders WHERE id = ?

Sumber: Hasil konfigurasi eksperimen penulis

WRITE Operations (80%):

Tabel 27. Distribusi Query WRITE untuk Heavy Write Monolith

Query Type	Distribusi	Deskripsi
INSERT User	25%	Insert user baru
INSERT Order	20%	Insert order baru
UPDATE User	15%	Update data user
UPDATE Order Status	15%	Update status order
UPDATE Product Stock	15%	Update stok produk
INSERT Order Item	10%	Insert item order

Sumber: Hasil konfigurasi eksperimen penulis

Tabel 28. Summary hasil Heavy Write Monolith

Metric	**Nilai**
Total Iterations	170,182
Total Duration	~6 menit 33 detik
Throughput	433.13 iter/s

Sumber: Hasil pengujian eksperimen penulis

Tabel 29.Read Latency Heavy Write Monolith

Statistik	Nilai (ms)
avg	1.10
min	0
max	104
med	1
p(90)	2
p(95)	2
p(99)	-

Sumber: Hasil pengujian eksperimen penulis

Tabel 30. Write Latency Heavy Write Monolith

Statistik	Nilai (ms)
avg	36.17
min	*
max	4,870
med	14
p(90)	117
p(95)	184
p(99)	-

Sumber: Hasil pengujian eksperimen penulis

Operations

Tabel 31. Operations Heavy Write Monolith

Metric	Count	Rate (/s)	Percentage
Read Operations	34,043	86.64	20.00%
Write Operations	136,139	346.49	80.00%
Total	170,182	433.13	100%

Sumber: Hasil pengujian eksperimen penulis

Error & Threshold

Tabel 32. Error & Threshold Heavy Write Monolith

Metric	Threshold	Actual	Status
read_latency p(95)	< 500ms	2ms	PASS
read_latency p(99)	< 1000ms	-	-
write_latency p(95)	< 1500ms	184ms	PASS
write_latency p(99)	< 3000ms	-	-
error_rate	< 10%	0.00%	PASS

Sumber: Hasil pengujian eksperimen penulis

Heavy Write Split

Target Database:
READ → mysql-slave:3306
WRITE → mysql-master:3306

Tabel 33. Variable Input untuk Heavy Write Split

Parameter	Nilai
Read Ratio	20%
Write Ratio	80%
Max Virtual Users	100 Vus
Total Durasi	6 menit
Sleep Interval	100 ms
Read Target	mysql-slave (Replica)
Write Target	mysql-master (Primary)

Sumber: Hasil konfigurasi eksperimen penulis

Load Stages: (Sama dengan Monolith)

Tabel 34. Load Stages untuk Heavy Write Split

Stage	Durasi	Vus
Ramp Up	30s	0 → 10
Moderate	2m	10 → 50
Peak	1m	50 → 100
Sustained	2m	100
Ramp Down	30s	100 → 0

Sumber: Hasil konfigurasi eksperimen penulis

READ Operations (20%) → SLAVE:

Tabel 35. Distribusi Query READ untuk Heavy Write Split

Query Type	Distribusi	Deskripsi
SELECT User by ID	40%	SELECT * FROM users WHERE id = ?
SELECT Product by ID	30%	SELECT * FROM products WHERE id = ?
SELECT Order Status	30%	SELECT id, status FROM orders WHERE id = ?

Sumber: Hasil pengujian eksperimen penulis

WRITE Operations (80%) → MASTER:

Tabel 36. Distribusi Query WRITE untuk Heavy Write Split

Query Type	Distribusi	Deskripsi
INSERT User	25%	Insert user baru
INSERT Order	20%	Insert order baru
UPDATE User	15%	Update data user
UPDATE Order Status	15%	Update status order
UPDATE Product Stock	15%	Update stok produk
INSERT Order Item	10%	Insert item order

Sumber: Hasil pengujian eksperimen penulis

Tabel 37. Summary hasil Heavy Write Split

Metric	Nilai
Total Iterations	163,666
Total Duration	~6 menit 33 detik
Throughput	416.63 iter/s

Sumber: Hasil pengujian eksperimen penulis

Read Latency

Tabel 38. Read Latency Heavy Write Split

Statistik	Nilai (ms)
avg	1.57
min	0
max	142
med	1
p(90)	2
p(95)	5
p(99)	-

Sumber: Hasil pengujian eksperimen penulis

Tabel 39. Write Latency Heavy Write Split

Statistik	Nilai (ms)
avg	42.57
min	*
max	5,030
med	13
p(90)	154
p(95)	190
p(99)	-

Sumber: Hasil pengujian eksperimen penulis

Operations

Tabel 40. Operations Heavy Write Split

Metric	Count	Rate (/s)	Percentage
Read Operations (Slave)	32,483	82.69	19.85%
Write Operations (Master)	131,183	333.94	80.15%
Total	163,666	416.63	100%

Sumber: Hasil pengujian eksperimen penulis

Tabel 41. Error & Threshold Heavy Write Split

Metric	Threshold	Actual	Status
read_latency p(95)	< 500ms	5ms	PASS
read_latency p(99)	< 1000ms	-	-
write_latency p(95)	< 1500ms	190ms	PASS
write_latency p(99)	< 3000ms	-	-
error_rate	< 10%	0.00%	PASS

Sumber: Hasil pengujian eksperimen penulis

Tabel 42. Variable Input untuk Balanced Monolith

Parameter	Nilai
Read Ratio	50%
Write Ratio	50%
Max Virtual Users	100 Vus
Total Durasi	6 menit
Sleep Interval	100 ms

Sumber: Hasil pengujian eksperimen penulisSumber: Hasil pengujian eksperimen penulis

Load Stages:

Tabel 43. Load Stages untuk Balanced Monolith

Stage	Durasi	Vus
Ramp Up	30s	0 → 10
Moderate	2m	10 → 50
Peak	1m	50 → 100
Sustained	2m	100
Ramp Down	30s	100 → 0

Sumber: Hasil pengujian eksperimen penulis

Tabel 44. Distribusi Query READ untuk Balanced Monolith

Query Type	Distribusi	Deskripsi
SELECT by ID	25%	SELECT * FROM users WHERE id = ?
SELECT with Filter	15%	SELECT * FROM users WHERE status = ?
SELECT Products by Category	15%	SELECT * FROM products WHERE category = ?
JOIN Orders-Users	15%	Join 2 tabel
Aggregation	15%	SELECT … GROUP BY status
Complex JOIN	15%	Join order_items dengan products

Sumber: Hasil pengujian eksperimen penulis

Tabel 45. Distribusi Query WRITE untuk Balanced Monolith

Query Type	Distribusi	Deskripsi
INSERT User	30%	Insert user baru
INSERT Order	20%	Insert order baru
UPDATE User Status	20%	Update status user
UPDATE Order Status	15%	Update status order
UPDATE Product Stock	15%	Update stok produk

Sumber: Hasil pengujian eksperimen penulis

Summary

Tabel 46. Summary hasil Balanced Monolith

Metric	Nilai
Total Iterations	29,150
Total Duration	~6 menit
Throughput	80.90 iter/s

Sumber: Hasil pengujian eksperimen penulis

Read Latency

Tabel 47. Read Latency Balanced Monolith

Statistik	Nilai (ms)
avg	654.35
min	0
max	28,620
med	26
p(90)	599
p(95)	3,500
p(99)	-

Sumber: Hasil pengujian eksperimen penulis

Write Latency

Tabel 48. Write Latency Balanced Monolith

Statistik	Nilai (ms)
avg	636.44
min	1
max	8,490
med	404
p(90)	1,490
p(95)	1,990
p(99)	-

Sumber: Hasil pengujian eksperimen penulis

Operations

Tabel 49. Operations Balanced Monolith

Metric	Count	Rate (/s)	Percentage
Read Operations	14,523	40.31	49.82%
Write Operations	14,627	40.60	50.18%
Total	29,150	80.90	100%

Sumber: Hasil pengujian eksperimen penulis

Error & Threshold

Tabel 50. Error & Threshold Balanced Monolith

Metric	Threshold	Actual	Status
read_latency p(95)	< 500ms	3,500ms	FAIL
read_latency p(99)	< 1000ms	-	-
write_latency p(95)	< 1000ms	1,990ms	FAIL
write_latency p(99)	< 2000ms	-	-
error_rate	< 10%	0.00%	PASS

Sumber: Hasil pengujian eksperimen penulis

Tabel 51. Variable Input untuk Balanced Split

Parameter	Nilai
Read Ratio	50%
Write Ratio	50%
Max Virtual Users	100 Vus
Total Durasi	6 menit
Sleep Interval	100 ms
Read Target	mysql-slave (Replica)
Write Target	mysql-master (Primary)

Sumber: Hasil pengujian eksperimen penulis

Load Stages: (Sama dengan Monolith)

Tabel 52. Load Stages untuk Balanced Split

Stage	Durasi	Vus
Ramp Up	30s	0 → 10
Moderate	2m	10 → 50
Peak	1m	50 → 100
Sustained	2m	100
Ramp Down	30s	100 → 0

Sumber: Hasil pengujian eksperimen penulis

Tabel 53. Distribusi Query READ untuk Balanced Split

Query Type	Distribusi	Deskripsi
SELECT by ID	25%	SELECT * FROM users WHERE id = ?
SELECT with Filter	15%	SELECT * FROM users WHERE status = ?
SELECT Products by Category	15%	SELECT * FROM products WHERE category = ?
JOIN Orders-Users	15%	Join 2 tabel
Aggregation	15%	SELECT … GROUP BY status
Complex JOIN	15%	Join order_items dengan products

Sumber: Hasil pengujian eksperimen penulis

Tabel 54. Distribusi Query WRITE untuk Balanced Split

Query Type	Distribusi	Deskripsi
INSERT User	30%	Insert user baru
INSERT Order	20%	Insert order baru
UPDATE User Status	20%	Update status user
UPDATE Order Status	15%	Update status order
UPDATE Product Stock	15%	Update stok produk

Sumber: Hasil pengujian eksperimen penulis

Tabel 55. Summary hasil Balanced Split

Metric	**Nilai**
Total Iterations	26,515
Total Duration	~6 menit
Throughput	73.56 iter/s

Sumber: Hasil pengujian eksperimen penulis

Read Latency

Tabel 56. Read Latency Balanced Split

Statistik	Nilai (ms)
avg	1,400
min	0
max	45,680
med	86
p(90)	2,590
p(95)	10,300
p(99)	-

Sumber: Hasil pengujian eksperimen penulis

Write Latency

Tabel 57. Write Latency Balanced Split

Statistik	Nilai (ms)
avg	26.27
min	1
max	572
med	13
p(90)	25
p(95)	146
p(99)	-

Sumber: Hasil pengujian eksperimen penulis

Operations

Tabel 58. Operations Balanced Split

Metric	Count	Rate (/s)	Percentage
Read Operations (Slave)	13,360	37.06	50.39%
Write Operations (Master)	13,156	36.50	49.61%
Total	26,516	73.56	100%

Sumber: Hasil pengujian eksperimen penulis

Error & Threshold

Tabel 59. Error & Threshold Balanced Split

Metric	Threshold	Actual	Status
read_latency p(95)	< 500ms	10,300ms	FAIL
read_latency p(99)	< 1000ms	-	-
write_latency p(95)	< 1000ms	146ms	PASS
write_latency p(99)	< 2000ms	-	-
error_rate	< 10%	0.00%	PASS

Sumber: Hasil pengujian eksperimen penulis

KESIMPULAN

Hasil pengujian menunjukkan bahwa Monolith unggul di mayoritas skenario karena consistently memiliki read latency lebih rendah dan throughput total operasi lebih tinggi, terutama pada Heavy Read (read 152% lebih cepat, throughput 191,733 vs 190,880) dan Heavy Write (read 43% lebih cepat, write 18% lebih cepat, throughput 170,182 vs 163,666). Pada Balanced workload, terjadi bottleneck akibat lock contention antara proses baca dan tulis, di mana Split unggul jauh pada write latency (24x lebih cepat: 26ms vs 636ms) tetapi kalah pada read latency (1,400ms vs 654ms) dan tetap memiliki throughput lebih rendah (26,515 vs 29,150) karena overhead koneksi ke dua database dan replication lag pada slave. Dengan demikian, Monolith direkomendasikan untuk sistem dengan prioritas kecepatan baca dan throughput tinggi, sedangkan Split database cocok hanya jika kebutuhan write real-time sangat

kritis dan akan digunakan jangka panjang untuk high availability serta scalability dengan syarat ditambahkan optimasi seperti caching, load balancing, dan tuning replikasi.

REFERENSI

Ali, M., Iqbal, S., & Khan, R. (2020). Performance analysis of MySQL in distributed database environments. *Journal of Database Management, 31*(2), 58–74. https://doi.org/10.1016/j.jdbm.2020.05.003

Andriyani, W., Dawis, A. M., Natsir, F., Kholisatul'Ulya, N., Diningrat, M. S. M., Makmur, A., Rozalina, R., Cahyanto, T. A., & Setiawan, I. (2024). *Pengembangan Sistem Berbasis Data*. TOHAR MEDIA.

Chigurupati, S. R., et al. (2025). *Distributed database systems for scalable enterprise applications*. IJSat.

Diogo, M., Cabral, B., & Bernardino, J. (2019). Consistency models of NoSQL databases. *Future Internet.*

Gao, F., & Zhang, H. (2021). Comparative performance of distributed databases in large-scale applications. *International Journal of Database Systems, 45*(7), 36–50. https://doi.org/10.1016/j.ijds.2021.06.003

Haryoto, I. S., Firmansyah, G., Tjahjono, B., Widodo, A. M., Akbar, H., & Fatonah, N. S. (2025). Evaluation and optimization of MySQL master slave performance with quantitative methods on the database of psychology test applicants SIM PT XYZ at Polda Metro Jaya. *Locus Journal: Research & Devotion.*

Kumar, R., & Singh, P. (2019). Evaluating MySQL for cloud-based data storage solutions. *Cloud Computing Research Journal, 10*(3), 121–137. https://doi.org/10.1016/j.ccr.2019.03.005

Lee, J., & Kim, S. (2022). MySQL database performance in high-demand environments: A case study. *International Journal of Cloud Computing and Big Data, 10*(4), 45–59. https://doi.org/10.1016/j.jccb.2022.09.002

Lu, Y., Yu, X., & Madden, S. (2018). *STAR: Scaling transactions through asymmetric replication.* arXiv.

"Optimasi kinerja basis data terdistribusi menggunakan algoritma replication dan partitioning." (2025). *JSSR.*

Ramanathan, S., Gautam, G., & Srinivasan, V. (2021). Latency redundancy tradeoff in distributed read-write systems. *arXiv.*

Ramanathan, S., Gautam, G., Srinivasan, V., & Parag, P. (2021). Latency redundancy tradeoff in distributed read-write systems. *arXiv.*

Salunke, S. V., & Ouda, A. A. (2024). A performance benchmark for the PostgreSQL and MySQL databases. *Future Internet, 16*(10), 382. https://doi.org/10.3390/fi16100382

Shrestha, R. (2020). Performance of cluster-based high availability database in cloud containers. In *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER 2020)* (pp. 320–327). SCITEPRESS.

Smith, J., Thomas, K., & Davis, R. (2018). Performance comparison of monolithic and distributed databases. *Database Technologies Review, 23*(1), 12–28. https://doi.org/10.1016/j.dtr.2018.01.004

Tomsic, A. Z., Bravo, M., & Shapiro, M. (2018). *Distributed transactional reads: The strong, the quick, the fresh & the impossible.* arXiv.

Wicaksono, A. K., Nurrakhman, F., & Samidi. (2023). Comparation of distributed database model by clustering method in e-government system (Studi pada Kemenkeu RI). *Jurnal Teknik Informatika (JUTIF), 4*(1), 25–32. https://doi.org/10.20884/1.jutif.2023.4.1.629

Zhang, Y., & Liu, Q. (2020). Scalability of MySQL in distributed systems for big data applications. *Journal of Information Technology, 17*(5), 302–318. https://doi.org/10.1016/j.jit.2020.04.006