# BAB II
# TINJAUAN PUSTAKA

## 2.1 Landasan Teori

### 2.1.1 Database Management System (DBMS)

Sistem Manajemen Basis Data (Database Management System/DBMS) adalah perangkat lunak yang dirancang untuk mengelola, menyimpan, dan memanipulasi data dalam suatu basis data secara efisien. (Annisa Rahmawita dkk., 2023; J. Patni, H. Sharma, Ravi Tomar, 2021). DBMS memungkinkan pengguna untuk melakukan berbagai operasi seperti penyisipan, pengambilan, pembaruan, dan penghapusan data melalui antarmuka yang terstruktur (J. Patni, H. Sharma, Ravi Tomar, 2021). Salah satu peran utama DBMS adalah menyediakan lingkungan yang terorganisir dan terpusat untuk menyimpan data sehingga dapat diakses oleh berbagai aplikasi secara bersamaan tanpa adanya konflik. Selain itu, DBMS memberikan keuntungan seperti integritas data, keamanan, dan skalabilitas dalam pengelolaan informasi yang besar (Widodo, 2024). DBMS adalah perangkat lunak yang berperan penting dalam pengelolaan data secara efisien dengan menyediakan mekanisme untuk penyimpanan, manipulasi, dan pengambilan data melalui antarmuka terstruktur. Sistem ini memungkinkan berbagai operasi seperti penyisipan, pembaruan, dan penghapusan data, serta memastikan data dapat diakses oleh berbagai aplikasi secara bersamaan tanpa konflik. Selain itu, DBMS menawarkan keunggulan dalam menjaga integritas, keamanan, dan skalabilitas data, sehingga menjadi solusi yang andal dalam pengelolaan informasi dalam skala besar.

Seiring perkembangan teknologi, DBMS tidak hanya terbatas pada model relasional tradisional tetapi juga mencakup model *NoSQL* untuk menangani data yang tidak terstruktur. *NoSQL* seperti *MongoDB*, Couchbase, dan *Amazon DynamoDB* menjadi populer dalam skenario yang membutuhkan skalabilitas horizontal dan fleksibilitas dalam pengelolaan data semi-terstruktur atau tidak terstruktur (Hasibuan & Nasution, 2023). DBMS tidak lagi terbatas pada model relasional tradisional, yang menggunakan tabel dengan skema tetap, tetapi juga mencakup DBMS *NoSQL* yang dirancang untuk menangani data yang semi-terstruktur atau tidak terstruktur. *NoSQL* seperti *MongoDB*, Couchbase, dan *Amazon DynamoDB* semakin populer karena kemampuannya dalam mengelola data dengan skema yang lebih fleksibel, mendukung skalabilitas horizontal, serta menangani beban kerja yang dinamis dan bervolume besar. Model *NoSQL* ini sangat cocok untuk aplikasi modern seperti big data, *Internet of Things* (*IoT*), dan layanan berbasis *cloud*, di mana data tidak selalu memiliki format yang konsisten dan membutuhkan pemrosesan yang cepat serta efisien.

### 2.1.2 *NoSQL* Database

*NoSQL* Database adalah jenis sistem basis data yang dirancang untuk menangani data yang tidak dapat diakomodasi dengan baik oleh model basis data relasional tradisional. *NoSQL* sering digunakan untuk mengelola data besar (big data) yang bersifat tidak terstruktur, semi-terstruktur, atau bervariasi. Sistem ini menawarkan fleksibilitas dalam penyimpanan dan pengolahan data, yang sering kali sulit dicapai dengan database relasional. Salah satu keunggulan utama dari *NoSQL* adalah kemampuannya dalam skalabilitas horizontal, di mana data dapat

dengan mudah didistribusikan ke beberapa server untuk meningkatkan performa (Ali & Naeem, 2023)

Menurut (N & Khasawneh, Mahmoud H. AL-Sahlee, 2020), *NoSQL* database dibagi menjadi beberapa kategori berdasarkan model data yang didukung, yaitu *document stores*, *key-value stores*, *column-family stores*, dan graph databases. *Document stores*, seperti *MongoDB* dan *CouchDB*, menyimpan data dalam format dokumen *JSON* atau *BSON* yang cocok untuk aplikasi yang memerlukan fleksibilitas *schema-less*. *Key-value stores*, seperti Redis, digunakan untuk operasi yang membutuhkan kecepatan tinggi dalam penyimpanan dan pengambilan data sederhana. Selain itu, *NoSQL* sangat cocok untuk aplikasi modern seperti media sosial, *e-commerce*, dan *Internet of Things* (*IoT*), yang membutuhkan respon cepat dan kemampuan untuk menangani data dalam jumlah besar yang terus berkembang.

Dengan berkembangnya teknologi *cloud*, *NoSQL* menjadi semakin relevan karena mampu mengintegrasikan data dari berbagai sumber secara efisien. Sistem seperti *Amazon DynamoDB* dan *Google Firestore* bahkan dirancang khusus untuk lingkungan *cloud*-native, memungkinkan pengelolaan data dengan *latency* rendah dan keandalan tinggi. Meskipun memiliki banyak keunggulan, *NoSQL* tidak sepenuhnya menggantikan database relasional, melainkan digunakan sebagai pelengkap untuk memenuhi kebutuhan aplikasi tertentu.

### 2.1.3 *MongoDB*

*MongoDB* adalah salah satu sistem basis data *NoSQL* berbasis document store yang dirancang untuk menyimpan data dalam format *JSON*-like (*BSON*). *MongoDB* menawarkan fleksibilitas tinggi dalam pengelolaan data tanpa memerlukan schema yang kaku, sehingga cocok untuk aplikasi modern yang membutuhkan skalabilitas horizontal dan kemampuan untuk menangani data yang tidak terstruktur atau semi-terstruktur. *MongoDB* mendukung fitur seperti query yang fleksibel, indeks kompleks, dan replikasi data untuk memastikan ketersediaan dan keandalan yang tinggi. Keunggulan ini menjadikan *MongoDB* sering digunakan dalam aplikasi berbasis big data, *Internet of Things* (*IoT*), dan analitik *real-time* (Maharani, 2022).

Menurut (Harshitha, 2020), *MongoDB* memungkinkan pengguna untuk melakukan operasi *CRUD (Create, Read, Update, Delete)* dengan performa yang optimal berkat pengindeksan bawaan dan desain penyimpanan dokumen. Selain itu, *MongoDB* mendukung transaksi multi-dokumen yang memenuhi standar *ACID*, sehingga memperluas penggunaannya untuk aplikasi yang membutuhkan integritas data. Dalam konteks distribusi data, *MongoDB* menggunakan pendekatan *sharding* untuk membagi data secara horizontal ke beberapa *node*, memungkinkan pengelolaan data dalam skala besar tanpa mengorbankan performa.

Dengan berkembangnya infrastruktur *cloud*, *MongoDB* juga menyediakan layanan berbasis *cloud* melalui *MongoDB* Atlas, yang memungkinkan pengguna untuk mengelola database mereka dengan mudah tanpa perlu konfigurasi manual.

Hal ini menjadikan *MongoDB* sebagai pilihan utama bagi pengembang yang membutuhkan solusi database modern yang efisien dan mudah diintegrasikan ke dalam berbagai platform (Rathore & Bagui, 2024)

### 2.1.4 *Cloud Computing*

*Cloud* computing merupakan paradigma yang menyediakan sumber daya komputasi sesuai permintaan melalui internet, memungkinkan pengguna untuk mengakses aplikasi, penyimpanan, dan daya pemrosesan tanpa harus memiliki infrastruktur sendiri (SIDHAARTH VISHNU.K.R. S, 2023; Supongmen Walling, 2020). Teknologi ini menawarkan berbagai keuntungan, seperti skalabilitas, efisiensi biaya, dan fleksibilitas yang sangat dibutuhkan oleh bisnis modern (Mishra & Panda, 2023; SIDHAARTH VISHNU.K.R. S, 2023). Dengan kemampuan ini, *cloud computing* telah menjadi salah satu inovasi paling signifikan dalam teknologi informasi.

Penyedia utama di pasar *cloud computing* saat ini adalah *Amazon Web Services* (*AWS*), Microsoft Azure, dan *Google Cloud* Platform. Ketiga penyedia ini mendominasi pasar dengan berbagai layanan dan solusi inovatif yang dirancang untuk memenuhi kebutuhan bisnis yang terus berkembang (SIDHAARTH VISHNU.K.R. S, 2023). Sejak komersialisasinya pada tahun 2006, teknologi ini terus mengalami perkembangan pesat yang dipelopori oleh perusahaan-perusahaan teknologi besar (Mishra & Panda, 2023; SIDHAARTH VISHNU.K.R. S, 2023).

Namun, meskipun menawarkan berbagai keuntungan, adopsi *cloud computing* juga menghadapi tantangan, khususnya dalam hal keamanan dan privasi data. Isu ini menjadi perhatian utama bagi banyak organisasi yang menggunakan layanan *cloud*, karena data yang disimpan di *cloud* rentan terhadap ancaman seperti serangan siber dan kebocoran data (SIDHAARTH VISHNU.K.R. S, 2023; Supongmen Walling, 2020). Oleh karena itu, para peneliti dan praktisi teknologi terus mencari solusi untuk meningkatkan keamanan dalam lingkungan *cloud*.

Selain itu, adopsi *cloud computing* yang terus meningkat juga mendorong permintaan tenaga profesional yang terampil di bidang ini. Kebutuhan ini menjadi peluang sekaligus tantangan bagi individu maupun organisasi dalam memanfaatkan teknologi *cloud* untuk meningkatkan daya saing mereka (SIDHAARTH VISHNU.K.R. S, 2023).

Dengan berbagai manfaat dan tantangan yang ditawarkannya, *cloud computing* telah menjadi komponen utama dalam transformasi digital. Hal ini memotivasi berbagai penelitian untuk mengatasi masalah yang ada, sehingga teknologi ini dapat digunakan secara lebih luas dan efektif di masa depan (Supongmen Walling, 2020).

### 2.1.5 *Amazon Web Services* (*AWS*)

*Amazon Web Services* (*AWS*) merupakan platform komputasi awan yang komprehensif, menawarkan lebih dari 140 layanan, termasuk komputasi, penyimpanan, basis data, dan *Internet of Things* (*IoT*) (Verma dkk., 2020). *AWS*

dirancang untuk menyediakan sumber daya yang fleksibel dan hemat biaya kepada bisnis dengan sistem pembayaran sesuai penggunaan (*pay-as-you-go*), memungkinkan organisasi untuk melakukan skalabilitas dengan cepat tanpa investasi awal yang signifikan (Abhishek Saini dkk., 2024). Hal ini menjadikan *AWS* sebagai salah satu solusi *cloud computing* yang paling populer dan andal.

Platform *AWS* menyediakan lingkungan yang aman dan dapat diandalkan, memungkinkan organisasi untuk berinovasi dan beroperasi secara efisien. Infrastruktur global *AWS* mendukung berbagai aplikasi, mulai dari penyimpanan data hingga distribusi konten, yang memberikan fleksibilitas tinggi untuk memenuhi kebutuhan teknologi informasi bisnis modern (Verma dkk., 2020). Dengan skala globalnya, *AWS* memungkinkan pengelolaan sumber daya yang optimal di berbagai lokasi geografis.

Namun, dengan meningkatnya permintaan pengguna, server dapat menghadapi beban kerja yang tinggi. Mengatasi tantangan ini, *AWS* menggunakan teknik *load balancing* yang mendistribusikan tugas ke berbagai server di wilayah yang berbeda. Pendekatan ini bertujuan untuk menjaga kinerja optimal dan pengalaman pengguna yang konsisten, meskipun dalam situasi beban tinggi (Hota & Pattanayak, 2021).

Kemampuan *AWS* untuk mendukung berbagai kebutuhan, mulai dari pengelolaan data hingga pengembangan aplikasi, menjadikannya platform yang dapat diandalkan bagi bisnis dari berbagai skala. *AWS* memberdayakan organisasi untuk memanfaatkan teknologi *cloud* sebagai alat untuk pertumbuhan dan inovasi,

membantu mereka tetap kompetitif di era digital yang terus berkembang (Abhishek Saini dkk., 2024).

### 2.1.6 *Yahoo! Cloud Serving Benchmark* (*YCSB*)

*Yahoo! Cloud Serving Benchmark* (*YCSB*) adalah sebuah *framework open-source* yang dirancang untuk mengevaluasi dan membandingkan performa berbagai sistem database, khususnya database *NoSQL*. Dikembangkan oleh para peneliti di Yahoo! pada tahun 2010, *YCSB* diciptakan untuk menyediakan sebuah standar industri guna mengukur kapabilitas sistem database modern dalam menangani beban kerja transaksional yang umum ditemui pada aplikasi berbasis *cloud*. *YCSB* telah menjadi tolok ukur standar di industri dan akademisi untuk melakukan perbandingan performa yang objektif antara berbagai arsitektur database yang berbeda, seperti *MongoDB*, *Cassandra*, dan Redis.

Arsitektur *YCSB* bersifat modular dan dapat diperluas, memungkinkannya untuk mendukung puluhan jenis database. Kerangka kerja ini, yang ditulis dalam bahasa Java, secara umum terdiri dari dua komponen utama:

1. *Database Interface Layer (Binding):* Ini adalah lapisan konektor yang spesifik untuk setiap database. *Binding* ini bertugas menerjemahkan operasi generik *YCSB* (seperti *read, update, insert*) menjadi perintah asli yang dimengerti oleh database target.
2. *Workload Generator:* Ini adalah inti dari *YCSB* yang bertugas menghasilkan beban kerja sintetis. Komponen ini mensimulasikan perilaku pengguna aplikasi dengan mengirimkan serangkaian operasi ke database target.

Proses *benchmarking* dengan *YCSB* berjalan dalam dua fase utama:

1. Fase *load***:** Pada fase ini, *YCSB* akan mengisi database target dengan sejumlah data awal yang telah ditentukan. Data ini biasanya terdiri dari beberapa *field* dengan ukuran tertentu.
2. Fase *run***:** Setelah data dimuat, fase ini akan mengeksekusi beban kerja yang sebenarnya sesuai dengan konfigurasi yang dipilih. Fase inilah yang digunakan untuk mengukur metrik performa utama seperti *throughput* (operasi per detik) dan *latency* (waktu respons).

*YCSB* menyediakan serangkaian beban kerja standar (dikenal sebagai *Core Workloads*) yang dirancang untuk mensimulasikan berbagai jenis skenario aplikasi nyata. Setiap *workload* memiliki proporsi operasi baca, tulis, dan *update* yang berbeda. Beberapa *workload* yang paling umum digunakan adalah:

1 Workload A (*Update Heavy*): Beban kerja dengan proporsi seimbang, terdiri dari 50% operasi baca dan 50% operasi *update*. Skenario ini mensimulasikan aplikasi di mana pengguna sering melihat dan memodifikasi data, seperti pada sistem jejaring sosial atau platform kolaborasi dokumen.

2 Workload B (*Read Mostly*): Beban kerja yang didominasi oleh operasi baca, terdiri dari 95% operasi baca dan 5% operasi *update*. Ini merepresentasikan aplikasi seperti portal berita atau kata*log* produk, di mana data lebih sering dibaca daripada diubah.

3 Workload C (*Read Only*): Terdiri dari 100% operasi baca. Skenario ini cocok untuk aplikasi yang bersifat *read-only*, seperti *cache* profil pengguna yang datanya dibuat di tempat lain.

4 Workload D (*Read Latest*): Beban kerja yang fokus pada membaca data terbaru. Skenario ini mensimulasikan aplikasi seperti linimasa media sosial, di mana pengguna paling tertarik pada pembaruan status terkini.

5 Workload E (*Short Ranges*): Beban kerja yang mensimulasikan *query* dengan jangkauan pendek (*scan*), seperti membaca rangkaian pesan dalam sebuah utas forum.

6 Workload F (*Read-Modify-Write*): Skenario di mana klien membaca sebuah data, memodifikasinya, lalu menulis kembali perubahan tersebut ke database.

Fleksibilitas untuk memilih dan mengonfigurasi *workload* ini menjadikan *YCSB* sebagai alat yang sangat kuat untuk menguji bagaimana sebuah database berperilaku di bawah tekanan yang berbeda, sehingga memungkinkan dilakukannya analisis *trade-off* performa secara mendalam.

### 2.1.7 *Amazon CloudWatch*

*Amazon CloudWatch* adalah layanan pemantauan dan manajemen yang dirancang khusus untuk mengawasi sumber daya *cloud* dan aplikasi yang berjalan di lingkungan *Amazon Web Services* (*AWS*). Layanan ini mengumpulkan data berupa metrik dan *log* dari berbagai komponen dalam ekosistem *cloud*, termasuk *instance Amazon EC2*, database, dan aplikasi itu sendiri. Dengan data ini, *CloudWatch* memberikan wawasan operasional yang komprehensif,

memungkinkan pengguna untuk memvisualisasikan data performa, mengidentifikasi masalah, dan bahkan melakukan analisis akar penyebab (Anand, 2017).

Fungsi utama dari *CloudWatch* meliputi pengumpulan metrik, pemantauan *log*, dan penggunaan alarm. Metrik adalah data kuantitatif yang dikumpulkan secara berkala dari sumber daya *AWS*. Dalam penelitian, metrik ini sangat penting untuk mengukur performa sistem dan efisiensi sumber daya secara *real-time*, seperti penggunaan *CPU*, *RAM*, dan *I/O Wait*. *CloudWatch* juga memungkinkan pengguna untuk memantau *log* dari aplikasi dan sistem, yang sangat membantu dalam proses *troubleshooting* (Malaraju, 2022).

Selain itu, *CloudWatch* memiliki fitur alarm yang dapat dikonfigurasi untuk mengirimkan peringatan ketika suatu metrik melebihi atau di bawah ambang batas yang ditentukan. Fitur ini penting untuk otomatisasi dan respons proaktif terhadap perubahan performa. Penggunaan *CloudWatch* dalam sebuah penelitian, memungkinkan pemantauan yang terpusat dan terperinci, memastikan data yang dikumpulkan valid dan dapat diandalkan untuk analisis. Hal ini menjadikan *CloudWatch* alat yang efektif untuk mengevaluasi kinerja sistem dan mengoptimalkan penggunaan sumber daya di lingkungan *cloud* (Anand, 2017).

## 2.2 Penelitian Terkait (*State Of The Art*)

Tabel 2.1 berisi penelitian terkait, yang menjadi acuan dalam penelitian.

*Tabel 2. 1 Penelitian Terkait*

| No | Penulis | Masalah | Solusi | Tantangan (Topik Penelitian Lanjutan) |
|---|---|---|---|---|
| 1 | (Krechowicz dkk., 2021) | Memperkenalkan *strong consistency* ke dalam *NoSQL* tanpa mengorbankan skalabilitas. | Mengusulkan arsitektur SD2DS dengan penjadwalan operasi atomik menggunakan sequence numbers. | Menerapkan model konsistensi ini pada operasi yang lebih kompleks seperti transaksi multi-dokumen atau operasi agregasi terdistribusi, serta mengukur dampaknya. |
| 2 | (Burdakov dkk., 2016) | Sulitnya memilih parameter replikasi (N, W, R) secara optimal tanpa eksperimen yang mahal. | Mengembangkan model matematika (LST) untuk memperkirakan probabilitas *stale read* dan waktu tunggu. | Mengembangkan model kalibrasi otomatis berdasarkan workload *real-time* atau mengintegrasikan model ini ke dalam *platform auto-tuning* untuk database *NoSQL*. |
| 3 | (Ouyang dkk., 2022) | Pengujian *causal consistency MongoDB* yang ada dinilai tidak cukup menyeluruh dan kredibel. | Merancang dan mengimplementasikan pengujian Jepsen yang lebih ketat, mencakup tiga varian konsistensi dan skenario kegagalan yang lebih luas. | Mendesain algoritma pengecekan yang dapat menangani histori dengan operasi tulis yang dilaporkan gagal namun datanya tetap tersimpan (kasus *ThinAirRead*). |
| 4 | (Krechowicz dkk., 2025) | Menjaga *strong consistency* dan *throughput* tinggi secara bersamaan saat menggunakan replikasi data. | Mengusulkan arsitektur TS2DS dengan replikasi data dinamis untuk data yang sering diakses. | Menyelidiki algoritma cerdas berbasis *AI/ML* untuk menentukan secara dinamis kapan harus mereplikasi atau menghapus replika data berdasarkan pola akses *real-time*. |

| | | | | |
|---|---|---|---|---|
| 5 | (Nwe dkk., 2019) | *Latency* tinggi pada algoritma seleksi replika berbasis kuorum karena level konsistensi yang statis. | Mengusulkan pendekatan seleksi replika dengan tingkat konsistensi dinamis berdasarkan staleness rate (PBS). | Menguji performa algoritma pada workload yang lebih bervariasi (misalnya *read-intensive*) dan mengukur dampak terhadap *throughput* transaksi secara lebih mendalam. |
| 6 | (Chauhan, 2019) | Keandalan *MongoDB* yang kurang karena sifat tulis asinkron dan tantangan desain *schema-less*. | Melakukan tinjauan literatur untuk mengidentifikasi area penelitian pada *MongoDB*. | Mengimplementasikan dan menguji mekanisme validasi skema hibrida di atas *MongoDB* untuk menegakkan integritas data tanpa mengorbankan fleksibilitas sepenuhnya. |
| 7 | (Diogo dkk., 2019) | Belum ada studi komparatif yang memadai mengenai implementasi model konsistensi pada berbagai database *NoSQL*. | Menganalisis dan membandingkan secara teoretis model konsistensi pada lima database *NoSQL* populer. | Melakukan evaluasi empiris (eksperimental) berskala besar untuk memvalidasi perbandingan teoretis dan mengukur dampak nyata berbagai model konsistensi. |
| 8 | (Gorbenko dkk., 2020) | Kurangnya pemahaman kuantitatif tentang bagaimana pengaturan konsistensi mempengaruhi performa *Cassandra*. | Mengusulkan metodologi berbasis *benchmarking* untuk mengoptimalkan performa *Cassandra* secara dinamis. | Membangun sistem *auto-tuning* yang sepenuhnya otomatis yang mengimplementasikan metodologi ini untuk menyesuaikan level konsistensi *Cassandra* di lingkungan produksi. |
| 9 | (Schultz dkk., 2019) | Menjaga *strong consistency* bisa sangat mahal, sementara banyak aplikasi bisa mentolerir konsistensi yang lebih lemah. | Menyediakan konsistensi yang dapat diatur (tunable) melalui *writeConcern* dan *readConcern* di *MongoDB*. | Penelitian lanjutan untuk mengurangi *latency* pada w: "*majority*" dan mendukung operasi baca snapshot yang berjalan lama tanpa membebani sistem secara signifikan. |
| 10 | (Ferreira dkk., 2024) | Kurangnya pemahaman | Menganalisis performa *MongoDB*, | Memperluas eksperimen dengan |

| | | | | |
|---|---|---|---|---|
| | | dampak performa dari tingkat konsistensi pada DBMS *NoSQL* di lingkungan *geo-replicated*. | Redis, dan *Cassandra* di *cloud* dalam skenario *single-region* dan *multi-region*. | jumlah *node* yang lebih besar dan beban kerja yang lebih tinggi di lingkungan non-terbatas untuk memvalidasi temuan awal. |
| 11 | (Dhanagari, 2024) | *Eventual consistency* pada *MongoDB* tidak cocok untuk aplikasi kritikal yang butuh reliabilitas tinggi. | Memberikan panduan best practices untuk menyeimbangkan performa dan reliabilitas di *MongoDB*. | Mengembangkan model konsistensi adaptif atau optimasi berbasis AI yang dapat secara dinamis menukar performa dengan reliabilitas sesuai kebutuhan transaksi. |
| 12 | (Hassan & Bashir, 2023) | Lemahnya dukungan *ACID* dan konsistensi kuat pada *NoSQL* membebani aplikasi untuk menanganinya. | Mengevaluasi model konsistensi pada *ScyllaDB* (model kolom) untuk mengukur *throughput* dan *latency*. | Mempelajari model konsistensi pada tipe database *NoSQL* yang berbeda (misalnya document store) dan menganalisis dampak tingkat konsistensi terhadap *throughput* transaksi. |
| 13 | (Ouyang dkk., 2022) | Klaim konsistensi kausal *MongoDB* belum terbukti secara menyeluruh. | Mendesain ulang pengujian Jepsen dan melakukan evaluasi kausalitas dengan workload *YCSB*. | Menguji implementasi di lingkungan produksi skala besar untuk menjembatani kesenjangan antara hasil simulasi dan perilaku dunia nyata. |
| 14 | (Andor, 2021) | Performa *NoSQL* DBMS dapat sangat bervariasi antar versi, sehingga sulit untuk mengetahui dampak pembaruan versi tanpa pengujian langsung. | Melakukan studi perbandingan performa (waktu eksekusi) pada enam database *NoSQL* (*MongoDB*, *Cassandra*, HBase, Redis, Couchbase, OrientDB) menggunakan *YCSB* untuk mengklasifikasikan mana yang optimal untuk operasi baca | Menganalisis metrik performa lain di luar *throughput*, terutama *latency* (*average*, *max*, *95th/99th percentile*), yang krusial untuk aplikasi dengan kebutuhan waktu respons cepat. |
| 15 | (Matallah dkk., 2020) | Kurangnya standardisasi | Melakukan studi perbandingan | Memperluas skala pengujian (lebih |

| | | | | |
|---|---|---|---|---|
| | | dan banyaknya variasi solusi *NoSQL* menyulitkan pemilihan database yang paling sesuai dengan kebutuhan pengguna, jenis data, dan tipe pemrosesan. | performa (waktu eksekusi) pada enam database *NoSQL* (*MongoDB*, *Cassandra*, HBase, Redis, Couchbase, OrientDB) menggunakan *YCSB* untuk mengklasifikasikan mana yang optimal untuk operasi baca (*read*) dan pembaruan (*update*). | banyak operasi & *record*), menggunakan data dunia nyata (*IoT*), menguji dalam lingkungan terdistribusi, dan menggunakan metrik lain seperti konsistensi, ketersediaan, dan *latency*. |
| 16 | (Abu Kausar dkk., 2022) | Kurangnya perbandingan ekstensif antara berbagai database *NoSQL*, terutama dengan jumlah operasi yang sangat besar, sehingga menyulitkan proses pemilihan. | Membandingkan performa (*throughput* dan *latency*) dari *MongoDB*, *Cassandra*, dan Redis menggunakan *YCSB* dengan beban kerja hingga 1 juta operasi. Hasilnya menemukan bahwa *MongoDB* memiliki performa paling unggul. | Mengevaluasi lebih banyak database *NoSQL* di lingkungan *cloud* dan menguji aspek performa lainnya di luar *throughput* dan *latency*. |
| 17 | (Carvalho dkk., 2023) | Sedikitnya studi akademis yang secara spesifik hanya membandingkan performa antar database *NoSQL* berjenis document store. | Mengevaluasi performa tiga database dokumen (Couchbase, *CouchDB*, *MongoDB*) menggunakan delapan workload *YCSB*. Ditemukan *MongoDB* unggul, kecuali pada operasi pemindaian (*scan*), di mana performanya sangat buruk. | Mengevaluasi lebih banyak database dokumen dengan beban kerja yang lebih besar dan membandingkannya dengan database relasional di lingkungan cluster dan *cloud*. |

Pesatnya perkembangan teknologi dan adopsi layanan berbasis komputasi awan telah menempatkan performa dan efisiensi database sebagai faktor krusial. Penelitian ini berfokus pada evaluasi performa dua mode konsistensi utama pada *MongoDB*, yaitu *eventual* dan *strong*, dengan menganalisis parameter kinerja kunci (*latency*, *throughput*, dan penggunaan sumber daya server). Kekuatan utama

penelitian ini adalah penggunaan model data yang realistis (profil pengguna) dan pengujian pada infrastruktur *cloud* skala kecil yang merepresentasikan lingkungan pengembangan banyak aplikasi modern.

1. Landasan Konseptual: Trade-off Konsistensi vs. Performa

Pendekatan untuk menyeimbangkan konsistensi dan performa telah menjadi dasar dari banyak penelitian. (Schultz dkk., 2019). secara fundamental menjelaskan mekanisme *tunable consistency* pada *MongoDB* melalui parameter *writeConcern* dan *readConcern*, serta menyoroti tingginya biaya performa untuk mencapai *strong consistency*. Senada dengan itu, (Dhanagari, 2024) memberikan panduan kualitatif berupa best practices untuk menyeimbangkan keduanya, namun tidak didukung oleh data *benchmark* empiris. Penelitian-penelitian ini memberikan fondasi teoretis yang kuat, namun tidak menyediakan data *benchmark* empiris yang spesifik untuk berbagai jenis beban kerja aplikasi.

2. Analisis Performa Komparatif dan Benchmarking

Beberapa penelitian telah melakukan *benchmarking* pada database *NoSQL*. (Ferreira dkk., 2024). melakukan analisis komparatif yang sangat relevan pada *MongoDB*, Redis, dan *Cassandra*, namun fokus utamanya adalah pada dampak lingkungan *geo-replicated* (*multi-region*) skala besar. Studi lain oleh (Gorbenko dkk., 2020) dan (Hassan & Bashir, 2023) juga melakukan *benchmarking*, tetapi pada database *NoSQL* yang berbeda (*Cassandra* dan *ScyllaDB*) namun tidak secara spesifik menganalisis dampak pada infrastruktur dengan sumber daya terbatas.

Meskipun penelitian sebelumnya telah membahas *trade-off* konsistensi, terdapat kesenjangan seperti Belum ada studi yang secara sistematis dan komprehensif mengevaluasi dampak jenis beban kerja (dominan baca/tulis) terhadap performa dan penggunaan sumber daya secara bersamaan untuk mode *eventual* dan *strong consistency* pada *MongoDB*, khususnya dalam konteks lingkungan *cloud* skala kecil (*t2.micro*) yang menggunakan model data aplikasi yang realistis.

Penelitian ini dirancang untuk mengisi kesenjangan tersebut. Kontribusi utamanya adalah menghasilkan sebuah kerangka kerja strategis berbasis bukti yang dapat digunakan oleh para arsitek sistem untuk memilih mode konsistensi yang paling optimal, tidak hanya berdasarkan teori, tetapi juga berdasarkan karakteristik beban kerja aplikasi dan realitas keterbatasan infrastruktur yang mereka hadapi.

## 2.3 Kebaruan Penelitian

Table 2.2 Gambaran mengenai ruang lingkup dari penelitian ini.

*Tabel 2.2 Matriks Penelitian*

| **No** | **Penelitian** | **Database** | **Model Konsistensi(*Eventual* & *Strong*)** | **Analisis Performa(*Latency* & *Throughput*)** | **Analisis Sumber Daya(*CPU* & *RAM*)** | **Kustom Record / Data** |
|---|---|---|---|---|---|---|
| 1 | *Tunable Consistency in MongoDB* | *MongoDB* | ✓ | ✓ | ✗ | ✗ |

| No | Penelitian | Database | Model Konsistensi(*Eventual* & *Strong*) | Analisis Performa(*Latency* & *Throughput*) | Analisis Sumber Daya(*CPU* & *RAM*) | Kustom Record / Data |
|---|---|---|---|---|---|---|
| 2 | *Impacts of Data Consistency Levels in Cloud-Based NoSQL for Data-Intensive Applications* | *MongoDB, Cassandra,* | ✓ | ✓ | ✓ | ✗ |
| 3 | *Interplaying Cassandra NoSQL Consistency and Performance: A Benchmarking Approach* | *Cassandra* | ✓ | ✓ | ✗ | ✗ |
| 4 | *Checking Causal Consistency of MongoDB* | *MongoDB* | ✗ (Fokus Causal) | ✗ | ✗ | ✗ |
| 5 | *Verifying Transactional Consistency of MongoDB* | *MongoDB* | ✗ (Fokus Transactional) | ✗ | ✗ | ✗ |
| 6 | *Consistency Models of NoSQL Databases* | Multiple (Survei) | ✓ | ✗ | ✗ | ✗ |
| 7 | *A Consistent Replica Selection Approach for Distributed Key-Value Storage System* | *Cassandra* | ✓ | ✓ | ✗ | ✗ |
| 8 | *Highly Scalable & Consistent Distributed Architecture for NoSQL Datastore* | Arsitektur Baru | ✓ | ✓ | ✗ | ✓ |
| 9 | *An Evaluation of Consistency Models in NoSQL Databases* | *ScyllaDB* | ✓ | ✓ | ✗ | ✗ |

| No | Penelitian | Database | Model Konsistensi(*Eventual* & *Strong*) | Analisis Performa(*Latency* & *Throughput*) | Analisis Sumber Daya(*CPU* & *RAM*) | Kustom Record / Data |
|---|---|---|---|---|---|---|
| 10 | *MongoDB and Data Consistency: Bridging the Gap between Performance and Reliability* | *MongoDB* | ✓ (Kajian) | ✗ | ✗ | ✗ |
| 11 | *A Review on Various Aspects of MongoDB Databases* | *MongoDB* | ✓ (Kajian) | ✗ | ✗ | ✗ |
| 12 | *Estimation Models for NoSQL Database Consistency Characteristics* | Riak | ✓ | ✓ | ✗ | ✗ |
| 13 | *Benchmarking Replication in Cassandra and MongoDB NoSQL Datastores* | *MongoDB, Cassandra* | ✓ | ✓ (*Through put*) | ✗ | ✗ |
| 14 | *Performance Benchmarking for NoSQL Database Management Systems* | *MongoDB, Cassandra* | ✗ | ✓ (*Through put*) | ✗ | ✓ |
| 15 | *Evaluation of NoSQL Databases: MongoDB, Cassandra, HBase, Redis, Couchbase, OrientDB* | *MongoDB, Cassandra,* | ✓ (Kajian) | ✓ (Waktu Eksekusi) | ✓ (Kajian) | ✓ |
| 16 | *Study of Performance and Comparison of NoSQL Databases: MongoDB,Cassandra, and Redis Using YCSB* | *MongoDB, Cassandra,* | ✗ | ✓ | ✗ | ✓ |
| 17 | *Performance Evaluation of NoSQL Document Databases: Couchbase, CouchDB, and MongoDB* | Couchbase, *CouchDB,* | ✗ | ✓ (Waktu Eksekusi) | ✓ | ✓ |

| No | Penelitian | Database | Model Konsistensi(*Eventual* & *Strong*) | Analisis Performa(*Latency* & *Throughput*) | Analisis Sumber Daya(*CPU* & *RAM*) | Kustom Record / Data |
|---|---|---|---|---|---|---|
| 18 | Analisis Performa Database *NoSQL* pada Layanan *Cloud* (Penelitian Ini) | *MongoDB* | ✓ | ✓ | ✓ | ✓ |

Untuk memetakan posisi dan kebaruan penelitian ini secara visual, Tabel 2.2 menyajikan perbandingan komprehensif antara ruang lingkup penelitian ini dengan studi-studi relevan lainnya. Matriks ini menyoroti bagaimana penelitian ini secara sistematis menggabungkan beberapa dimensi analisis yang sering kali dipelajari secara terpisah.

Jika dilihat dari matriks, banyak penelitian telah berfokus pada hubungan inti antara Model Konsistensi dan Analisis Performa (*Latency* & *Throughput*). Studi seperti yang dilakukan oleh (Schultz dkk., 2019) dan (Hassan & Bashir, 2023) memberikan kontribusi penting dalam area ini dengan membandingkan performa pada berbagai model konsistensi. Hanya sedikit penelitian, seperti yang dilakukan oleh (Ferreira dkk., 2024), yang memperluas analisis hingga mencakup Analisis Sumber Daya (*CPU* & *RAM*) secara bersamaan.

Namun, kesenjangan signifikan muncul ketika kita mempertimbangkan konteks pengujian. Kolom terakhir, "Analisis pada Skenario Praktis," menunjukkan kebaruan utama dari penelitian ini. Matriks menunjukkan dengan

jelas bahwa studi-studi sebelumnya cenderung berfokus pada *benchmark* generik atau pada infrastruktur skala besar.

Oleh karena itu, kebaruan utama dari penelitian ini terletak pada pendekatannya yang holistik dan relevan secara praktis. Penelitian ini adalah satu-satunya studi dalam matriks ini yang secara simultan:

1 Mengevaluasi model konsistensi *eventual* dan *strong*.

2 Mengukur performa *latency* dan *throughput*.

3 Menganalisis dampak pada sumber daya *CPU* dan *RAM*.

4 Melakukan semua analisis di atas dalam konteks skenario yang sangat umum di dunia nyata, menggunakan model data aplikasi yang realistis (profil pengguna) pada infrastruktur *cloud* dengan sumber daya terbatas (*t2.micro*).

Pendekatan multi-dimensi yang didasarkan pada skenario praktis inilah yang memungkinkan penelitian ini menghasilkan sebuah kerangka kerja strategis yang diharapkan dapat langsung diterapkan oleh pengembang.