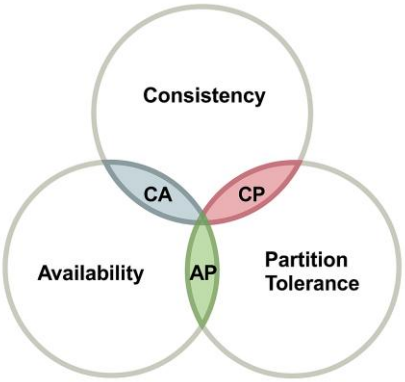
BAB II

TINJAUAN PUSTAKA

2.1 Landasan Teori

2.1.1 Teorema CAP (*Consistency, Availability, dan Partition Tolerance*)

Teorema CAP yang dikemukan oleh Eric Brewer pada tahun 2000 menyatakan bahwa sistem *database* terdistribusi, hanya dua dari tiga aspek utama yaitu: *Consistency, Availability* dan *Partition Tolerance* yang dapat dicapai secara bersamaan (Khasawneh dkk., 2020). *Consistency* berarti seluruh *node* dalam sistem menyajikan data yang sama pada waktu tertentu setelah suatu operasi tulis dikonfirmasi. *Availability* menunjukkan bahwa setiap permintaan yang diterima sistem akan memperoleh respons dalam batas waktu tertentu, baik berupa keberhasilan maupun kegagalan yang terdefinisi. *Partition Tolerance* adalah kemampuan sistem untuk tetap beroperasi meskipun terjadi kegagalan komunikasi atau pemisahan jaringan antar *node* (Suriansyah dkk., 2024). lustrasi hubungan ketiga aspek tersebut ditunjukkan pada Gambar 2.1



Gambar 2.1 Teorema CAP

Gambar 2.1 mengilustrasikan hubungan ketiga properti tersebut. Sistem dengan kombinasi CA (*Consistency dan Availability*) tidak dapat bertahan ketika terjadi partisi jaringan. Sistem CP (*Consistency dan Partition Tolerance*) akan mengorbankan *Availability* dengan menolak sebagian permintaan untuk menjaga *consistency*. Sementara itu, sistem AP (*Availability* dan *Partition Tolerance*) tetap melayani permintaan meskipun terdapat risiko inkonsistensi data (Meghwar & Meghji, 2025).

Penelitian ini secara khusus difokuskan pada aspek consistency dan partition tolerance dalam Teorema CAP. Pemilihan kedua aspek tersebut didasarkan pada relevansinya terhadap mekanisme replikasi pada MongoDB Replica Set, terutama dalam kondisi terjadinya gangguan jaringan. Consistency berkaitan langsung dengan proses sinkronisasi data antar node, sedangkan partition tolerance berkaitan dengan kemampuan sistem untuk tetap beroperasi ketika terjadi pemisahan jaringan.

Sementara itu, aspek *availability* tidak dijadikan fokus utama dalam penelitian ini karena dalam konfigurasi yang digunakan, ketersediaan layanan direpresentasikan secara operasional melalui parameter *request success rate*. Selain itu, pada skenario pengujian yang diterapkan, sistem tetap mampu melayani seluruh permintaan selama *quorum* mayoritas terpenuhi, sehingga tidak terjadi penurunan *availability* yang signifikan. Oleh karena itu, penelitian ini lebih menitikberatkan pada analisis *trade-off* antara *consistency* dan *partition tolerance* yang menunjukkan dinamika perubahan yang lebih terukur.

Teorema CAP dalam penelitian ini tidak hanya dipahami secara konseptual, tetapi dioperasionalkan melalui enam parameter terukur. Aspek *consistency* direpresentasikan oleh *replication lag, stale reads*, dan latensi pengakuan tulis mayoritas. Aspek *partition tolerance* direpresentasikan oleh *request success rate, recovery time*, dan *rollback data*. Keenam parameter tersebut digunakan untuk mengevaluasi *trade-off* CAP pada implementasi MongoDB *Replica Set*.

2.1.2 *Consistency*

Consistency dalam sistem *database* terdistribusi merujuk pada jaminan bahwa seluruh node menyajikan keadaan data yang selaras setelah suatu operasi tulis dikonfirmasi (Carrara dkk., 2020). Pada model *strong consistency* atau *linearizable consistency*, setiap operasi baca yang dilakukan setelah operasi tulis akan selalu mengembalikan nilai terbaru. Sebaliknya, pada *eventual consistency*, pembaruan data membutuhkan waktu untuk dipropagasikan ke seluruh *node*, sehingga pembacaan pada *node* tertentu dapat menghasilkan data yang belum diperbarui (Tseng dkk., 2020).

Dalam MongoDB, tingkat konsistensi dipengaruhi oleh konfigurasi *writeConcern, readConcern*, dan *readPreference.* Pengaturan *writeConcern = majority* memastikan bahwa operasi tulis dianggap berhasil setelah direplikasi dan diakui oleh mayoritas anggota *replica set*. Pembacaan dari *primary* dengan konfigurasi *readConcern* yang sesuai memberikan jaminan konsistensi yang lebih kuat dibandingkan pembacaan dari *secondary*, yang bersifat *eventual consistency*.

Pengukuran *strong consistency* dalam penelitian ini dilakukan dengan menghitung waktu lantensi tulis menggunakan rumus pada persamaan 1 berikut:

$$T_{\text{strong}} = t_{\text{ack}} - t_{\text{write}} \quad (1)$$

Keterangan:

t_{write} = waktu ketika data pertama kali dikirim ke *primary node*.

t_{ack} = waktu ketika *primary* mengirimkan pengakuan bahwa data telah direplikasi ke mayoritas node.

Nilai ini merepresentasikan waktu yang dibutuhkan sistem untuk mencapai jaminan *majority write*. Dalam konteks MongoDB, *consistency* kuat tidak secara langsung diukur melalui nilai latensi, melainkan melalui jaminan bahwa operasi tulis telah direplikasi dan diakui oleh mayoritas node dalam *replica set*. Oleh karena itu, latensi pengakuan tulis mayoritas pada penelitian ini digunakan sebagai indikator biaya waktu untuk mencapai *consistency* mayoritas, bukan sebagai bukti konsistensi *linearizable* secara penuh.

Sedangkan *eventual consistency* diukur berdasarkan selisih waktu antara penulisan di *primary* dan data tersedia di *secondary*:

$$\text{Delay}_i = t_{\text{read},i} - t_{\text{write},i} \quad (2)$$

Keterangan:

$t_{\text{read},i}$ = waktu pembacaan data pada *secondary node* ke-i

$t_{\text{write},i}$ = waktu penulisan data di *primary node*.

Nilai ini mencerminkan *replication lag* atau keterlambatan propagasi data. Semakin besar nilai *delay*, semakin tinggi kemungkinan terjadinya pembacaan data usang (*stale read*) pada *secondary*.

Dengan demikian, *consistency* dalam penelitian ini direpresentasikan melalui tiga indikator empiris: latensi pengakuan tulis mayoritas, *replication lag*, dan frekuensi *stale reads*. Ketiga metrik tersebut digunakan untuk menggambarkan dinamika sinkronisasi data dalam MongoDB *Replica Set* pada kondisi normal maupun saat terjadi partisi jaringan.

2.1.3 *Partition Tolerance*

Partition Tolerance adalah kemampuan sistem *database* terdistribusi untuk tetap beroperasi walaupun terjadi kegagalan komunikasi antar *node* (Raikwar dkk., 2020). Di MongoDB partisi jaringan dapat terjadi jika koneksi antara *primary* dan *secondary* terputus sebagian atau seluruhnya. Sistem ini memiliki *partition tolerance* tinggi dapat terus memproses permintaan, walaupun beberapa *node* tidak terhubung. Namun partisi sering kali menimbulkan risiko inkonsistensi data, tergantung pada konfigurasi CAP yang dipilih. Pengujian *partition tolerance* dalam penelitian ini dilakukan dengan simulasi *network partition* menggunakan aturan *firewall* (*iptables*) atau memutuskan jaringan ke salah satu *node*. Selama partisi, sistem diuji untuk melihat keberhasilan operasi baca/tulis dan perilaku sinkronisasi data. Parameter yang akan diuji mencakup *persentase request sukses,* waktu pemulihan (*recovery time*), dan tingkat kehilangan data. Hasil pengujian dapat menunjukan sejauh mana sistem mampu bertahan terhadap gangguan jaringan. *Partition tolerance* yang baik memastikan

layanan tetap berjalan meskipun *node* terisolasi. Peningkatan *partition tolerance* biasanya mengorbankan tingkat *consistency*.

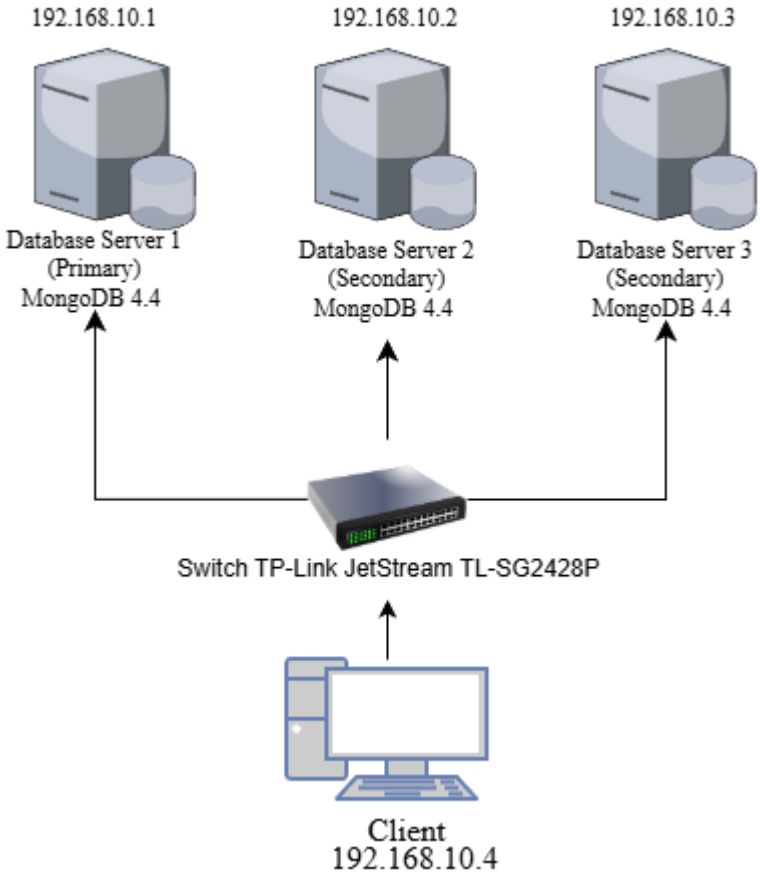
2.1.4 Database NoSQL

NoSQL (*Not Only SQL*) adalah *database* non-relasional yang dirancang untuk menangani data bervolume besar, struktur bervariasi, dan kebutuhan skalabilitas tinggi (Wibyantoro & Asmoro, 2020). *Database* relasional, NoSQL tidak memerlukan skema tetap sehingga lebih fleksibel dalam menyimpan data semi terstruktur. NoSQL memiliki beberapa model penyimpanan, antara lain *document-oriented, key-value, column family*, dan *graph database* (Tripathi, 2025). MongoDB termasuk dalam kategori *document-oriented database* yang menyimpan data dalam bentuk dokumen BSON. Keunggulan NoSQL meliputi kemudahan skala *horizontal* melalui *sharding* dan *replication*, sistem ini cocok digunakan pada aplikasi yang membutuhkan ketersediaan tinggi dan respon cepat. Pada Teorema CAP banyak *database* NoSQL mengutamakan *Availability* dan *partition tolerance*. Namun konfigurasi tertentu memungkinkan peningkatan *consistency*, seperti pada MongoDB. NoSQL banyak digunakan dalam *big data analytics*, IoT, dan layanan *web real-time*. Penggunaannya didorong oleh meningkatnya kebutuhan akan sistem yang mampu memproses data secara cepat dan fleksibel. Dalam penelitian ini, NoSQL dipilih karena kemampuannya mendukung replikasi otomatis dan pengaturan *consistency*.

2.1.5 Replikasi *Database*

Replikasi *database* adalah mekanisme penyalinan data dari satu (*primary*) ke satu atau (*secondary*) dengan tujuan menjamin ketersediaan data, redundansi, dan kontinuitas layanan dalam sistem *database* terdistribusi (Wibowo dkk., 2025). Mekanisme ini memungkinkan *system* tetap melayani permintaan meskipun terjadi kegagalan pada salah satu *node*. Dalam sistem terdistribusi *modern*, replikasi juga bertindak sebagai strategi untuk menjaga kontinuitas layanan saat terjadi kerusakan pada komponen sistem (Uriawan dkk., 2024).

Semakin banyak *node* yang terlibat dalam proses replikasi, maka semakin besar pula kebutuhan terhadap sumber daya seperti CPU, RAM, *bandwidth*, dan waktu sinkronisasi (Lestandy dkk., 2020). Oleh karena itu, penting untuk melakukan pengaturan dan pengujian performa agar proses replikasi tidak mengganggu kestabilan sistem secara keseluruhan. Seperti yang dikemukakan oleh Ahmed pengujian terhadap beban sistem saat proses replikasi sangat penting untuk mengetahui titik optimal konfigurasi replikasi yang efisien (Ahmed dkk., 2023). Topologi replikasi nya ditunjukkan pada gambar 2.2.



Gambar 2. 2 Topologi replikasi

Gambar 2.2 menjelaskan topologi replikasi yang digunakan pada penelitian ini. Satu *client* berbasis windows berada diluar segmen mesin virtual dan terhubung ke jaringan uji melalui *switch*. Di dalam segmen *Virtual Machine* terdapat tiga *server*: *Server* 1 (*Primary*) sebagai penerima seluruh operasi tulis, serta *Server* 2 (*Secondary*) dan *Server* 3 (*Secondary*) yang mereplikasi perubahan dari *primary* secara asinkron melalui oplog.

2.1.6 JSON

JavaScript Object Notation (JSON) adalah format data berbasis teks yang digunakan secara luas dalam pertukaran data antara *server* dan JSON

mendukung struktur data kompleks seperti objek bersarang dan *array*, membuatnya sangat fleksibel dan mudah dibaca. MongoDB menggunakan BSON (*Binary* JSON) sebagai implementasi biner dari JSON untuk meningkatkan efisiensi dalam penyimpanan transmisi data (Truică dkk., 2021). Penggunaan JSON dalam sistem *database* non-relasional memberikan efisiensi tinggi dan kecepatan transfer data karena keduanya mendukung format data semi-terstruktur yang fleksibel dan mudah diproses (Rianto dkk., 2021).

Penggunaan JSON dalam replikasi memengaruhi ukuran *payload* dan proses sinkronisasi antar *node*. Format ini mendukung perubahan data yang dinamis dan tidak membutuhkan skema tetap, yang membuatnya ideal untuk sistem berbasis dokumen seperti Mongo. Ukuran dokumen yang besar dapat memperlambat sinkronisasi jika tidak ditangani dengan baik. Efisiensi format data sangat mempengaruhi *bandwidth* dan latensi dalam proses replikasi, terutama ketika digunakan dalam sistem yang memiliki partisi jaringan atau koneksi tidak stabil (Latták & Koupil, 2022).

2.1.7 MongoDB

MongoDB adalah sistem manajemen *database* NoSQL berbasis dokumen yang dirancang untuk skala besar dan fleksibilitas tinggi (Medina dkk., 2022). MongoDB menyimpan data dalam format dokumen BSON, yang memungkinkan fleksibilitas skema dan kecepatan dalam pengambilan data. Fitur-fitur seperti index dinamis, *agregasi pipeline*, dan sistem replikasi otomatis menjadikan MongoDB pilihan utama dalam sistem big data dan arsitektur mikroservis. MongoDB dalam menangani data semi terstruktur sangat relevan

dalam aplikasi *real-time* yang menuntut skalabilitas horizontal. MongoDB menyediakan konfigurasi untuk mengontrol *consistency* dan performa sistem, seperti *writeConcern* untuk menentukan jumlah *node* yang harus mengakui penulisan data, dan *readPreference* untuk mengatur preferensi pembacaan dari *primary* atau *secondary node*. MongoDB menjadi objek eksperimen karena menyediakan fleksibilitas konfigurasi *replica set* yang memungkinkan pengujian performa dan *consistency* dalam berbagai skenario. (Khan dkk., 2023) menyatakan bahwa MongoDB sangat cocok untuk pengujian berbasis CAP *Theorem* karena kemampuannya dalam menyeimbangkan *trade-off* antar aspek *consistency*, ketersediaan, dan *partition tolerance*.

2.2 Penelitian Terkait

2.2.1 State Of The art

Tabel 2.1 State Of The Art

No	Peneliti	Judul	Objek Penelitian	Algoritma/Metode	Hasil Penelitian
1	(Ahmed dkk., 2023)	*Consistency Issue and Related Trade-Offs in Distributed Replicated Systems and Databases: A Review*	Sistem/DB terdistribusi bereplikasi	Tinjauan literatur (CAP, PACELC, quorum)	Penelitian ini menunjukkan bahwa saat partisi jaringan, sistem praktis harus memilih C atau A semakin kuat mekanisme koordinasi (*quorum*, sinkronisasi) makin menambah latensi namun mengurangi risiko *stale/rollback*, sehingga metrik seperti latensi tulis/baca dan ketersediaan perlu diukur eksplisit. Penelitian ini menunjukkan bahwa saat partisi jaringan, sistem praktis harus memilih C atau A semakin kuat mekanisme koordinasi (*quorum*, sinkronisasi) makin menambah latensi namun mengurangi risiko *stale/rollback*, sehingga metrik seperti latensi tulis/baca dan ketersediaan perlu diukur eksplisit.

2	(Campêlo dkk., 2020)	*A Brief Survey on Replica Consistency in Cloud Environments*	*Cloud storage/NoSQL*	Survei replikasi & *consistency*	Mengklasifikasikan pendekatan *consistency* replika dan menekankan *tunable consistency* penguatan *consistency* cenderung menaikkan waktu respons dan mengurangi *throughput* pada beban/latensi jaringan tinggi, sehingga perlu pengamatan *replication lag* dan *propagation/visibility delay* saat uji
3	(Bhosale, dkk., 2023)	*Data Consistency Models in Distributed Systems: CAP Theorem Revisited*	Sistem terdistribusi	Review model *consistency* & CAP	Penelitian ini memetakan spektrum model (*linearizable, eventual/causal*) dan kaitannya dengan koordinasi/latensi. Relevan untuk memilih *readConcern/writeConcern* sesuai jaminan yang diukur.
4	(Dhanagari, 2024)	*MongoDB and Data Consistency: Bridging the Gap between Performance and Reliability*	MongoDB (*replica set*)	Ulasan/praktik (*readPreference, readConcern, writeConcern*)	Penelitian ini Menganjurkan *w:"majority"* untuk durabilitas, serta *read* dari *primary* untuk meminimalkan *stale read* ke *secondary* meningkatkan skalabilitas namun bergantung *replication lag*, sehingga *stale-read rate* dan *lag* jadi indikator penting saat pengujian.
5	((Alflahi dkk.,	*Enhancement of*	MongoDB/NoSQL	MongoDB/NoSQL	Menunjukkan dampak penguatan

	2024)	*database access performance by improving data consistency in a non-relational database system (NoSQL)*			*consistency* pada metrik performa seperti dari *baseline* ke model usulan *throughput/latensi* baca/update berubah menjadi 2864.726 ms / 32806.275 ms / 51845.629 ms dari 2914.110 ms / 26510.930 ms / 32457.662 ms, menandakan biaya latensi seiring kontrol *consistency* yang lebih ketat.
6	(Suriansyah dkk., 2024)	Peningkatan Kinerja Database melalui Teknik Batch Loading dan Parallel Processing pada Proses Load Data	Proses load data ke DB	Eksperimen batch & parallel	Membuktikan batch/parallel mempercepat proses load dibanding serial laporan baseline ≈120 detik tanpa optimasi dipakai sebagai acuan, sehingga teknik ini cocok sebagai generator beban yang stabil saat mengukur *replication lag* dan *visibility delay* pada replikasi.
7	(Ferreira dkk., 2024)	*Impacts of Data Consistency Levels in Cloud-based NoSQL for Data-Intensive Applications*	Cassandra, MongoDB, Redis (*single- vs multi-region cloud*)	*Eksperimen nyata di cloud variasi consistency level*	Ketika *consistency* dinaikkan dari *weak* menjadi *strong*, beban baca Cassandra di *single-region* mengalami lonjakan waktu respons (175%) disertai penurunan *throughput* (62%). Dampak yang lebih tajam terlihat pada MongoDB di *multi-region*

					throughput baca turun sekitar 57%, sedangkan waktu respons melonjak ~237%. Pola ini menunjukkan *trade-off consistency* –kinerja kian nyata pada skenario antar-wilayah.
8	(Uriawan dkk., 2024)	*Implementing Replica Set: Strategy to Improve the Performance of NoSQL Database Cluster in MongoDB*	MongoDB (*single node vs 3-node replica set)*	Eksperimen CRUD (1000 operasi)	Pada uji 1.000 operasi CRUD, konfigurasi *single-node* lebih efisien pada jalur tulis—Create 1,754 s, Update 2,256 s, Delete 1,908 s—dibanding *replica set* yang memerlukan 3,984 s, 4,754 s, dan 4,399 s. Sebaliknya, pada jalur baca, *replica set* sedikit lebih cepat (0,038 s) daripada *single-node* (0,047 s). Temuan ini menegaskan adanya *overhead* koordinasi *quorum*/replikasi pada *write* serta keuntungan *read-scaling* dan *high availability* pada *replica set*.
9	(Khan dkk., 2023)	*SQL and NoSQL Database Software Architecture Performance Analysis and Assessments—A*	Arsitektur & performa SQL vs NoSQL	Arsitektur & performa SQL vs NoSQL	NoSQL unggul pada skalabilitas horizontal dan ketersediaan, sementara tingkat *consistency* yang dipilih berdampak pada latensi dan ketahanan saat partisi. SLR ini

		Systematic Literature Review			memperkuat perlunya mengukur metrik *latensi/throughput* dan *availability* sesuai setelan *consistency* yang dipakai.
10	(Wibowo dkk., 2025)	*A Comparative Study of Multi-Master Replication of NoSQL Database Server with Varying Data Formats*	CouchBase & CouchDB (*multi-master*)	Eksperimen replikasi *multi-master*, perbandingan format data	Menilai dampak skema replikasi dan format data terhadap performa. Contoh ringkas yang dilaporkan: transfer rate terendah CSV di CouchBase ≈111,41 kbps; penggunaan CPU terendah XML di CouchBase ≈13,89%; memori terendah JSON ≈1,68%; waktu delete tercepat di CouchDB ≈1,5 s; insert tercepat CSV di CouchBase ≈33,28 s rata-rata skenario penulis. Ini menguatkan bahwa desain replikasi memengaruhi kinerja yang kemudian berelasi dengan observasi *consistency*.

2.2.2 Matrix Penelitian

Tabel 2.2 Parameter penelitian

No	Peneliti	Judul Penelitian	***Consistency***			***Partition Tolerance***		
			Replication Lag	*Stale Reads*	*Latency Strong*	*Request Success Rate*	*Recovery Time*	*Rollback Data*
1	(Dhanagari, 2024)	*MongoDB and Data Consistency: Bridging the Gap between Performance and Reliability*	✓	✓	✓	-	-	-
2	(Ferreira dkk., 2024)	*Impacts of data consistency levels in cloud-based NoSQL for data-intensive applications*	✓	-	✓	-	-	-
3	(Uriawan dkk., 2024)	*Implementing Replica Set: Strategy toImprove the Performance of NoSQLDatabase Cluster in MongoDB*	✓	✓	-	-	-	-
4	(Wibowo dkk., 2025)	A Comparative Study of Multi-Master Replication of NoSQL Database Server with Varying Data Formats	✓	-	-	-	✓	-
5	(Campêlo dkk., 2020)	*A brief survey on replica consistency in cloud environments*	✓	✓	✓	-	-	-
6	(Bhosale, 2023)	*Data Consistency Models in Distributed Systems: CAP Theorem Revisited*	-	-	✓	-	-	-
7	(Gorbenko	*Interplaying cassandra nosql consistency and*	✓	✓	-	-	✓	-

	dkk., 2020)	*performance: A benchmarking approach*						
8	(Ahmed dkk., 2023)	*CONSISTENCY ISSUE AND RELATED TRADE-OFFS IN DISTRIBUTED REPLICATED SYSTEMS AND DATABASES: A REVIEW*	✓	✓	✓	-	-	-
9	(Khan dkk., 2023)	*SQL and NoSQL Database Software Architecture Performance Analysis and Assessments—A Systematic Literature Review*	✓	✓	✓	✓	-	-
10	**Our Research**		✓	✓	✓	✓	✓	✓

Berdasarkan matriks penelitian pada Tabel 2.2, dapat dilihat bahwa penelitian-penelitian sebelumnya umumnya masih menitikberatkan pada salah satu aspek saja. Sebagian penelitian berfokus pada aspek *consistency* dengan mengukur parameter seperti *replication lag*, *stale reads*, dan *latency strong*, sementara penelitian lainnya mulai menambahkan pengukuran terkait *partition tolerance* seperti *request success rate* atau *recovery time*. Namun, kajian yang mengintegrasikan kedua aspek tersebut secara bersamaan dalam satu rangkaian eksperimen, khususnya dengan simulasi partisi jaringan secara langsung, masih terbatas. Oleh karena itu, penelitian ini menggunakan enam parameter utama, yaitu *replication lag*, *stale reads*, *latency strong*, *request success rate*, *recovery time*, dan *rollback data*, yang merepresentasikan aspek *consistency* dan *partition tolerance* dalam Teorema CAP. Pendekatan ini diharapkan dapat memberikan gambaran yang lebih komprehensif mengenai perilaku replikasi basis data NoSQL ketika menghadapi kondisi gangguan jaringan.

Pemilihan enam parameter pengukuran dalam penelitian ini didasarkan pada representasi empiris dari dua aspek utama dalam Teorema CAP. Pada sisi *consistency*, penelitian-penelitian sebelumnya menunjukkan bahwa tingkat *consistency* replikasi direfleksikan melalui waktu propagasi data *(replication lag),* probabilitas ketidaksesuaian pembacaan (*stale reads*), serta waktu pengakuan penulisan data pada *primary* (*write latency/latency strong*) (Dhanagari, 2024; Ferreira dkk., 2024; Campêlo dkk., 2020). Ketiga metrik tersebut telah digunakan sebagai indikator stabilitas sinkronisasi data dalam sistem terdistribusi.

Pada *partition tolerance*, penelitian terkait *partition tolerance* jaringan menekankan dua aspek utama, yaitu keberhasilan operasi selama partisi (*request success rate*) dan kecepatan pemulihan setelah koneksi diperbaiki (*recovery time*) (Khan dkk., 2023; Wibowo dkk., 2025; Gorbenko dkk., 2020). Namun, belum banyak penelitian yang mengukur risiko kehilangan atau pembatalan data setelah *re-election*.

Dengan demikian, enam parameter tersebut dipilih karena secara bersama-sama mampu memetakan kedua sisi *trade-off* Teorema CAP secara komprehensif: tiga mengukur *consistency* data (*replication lag, stale reads, latency strong*), dan tiga mengukur *partition tolerance* (*request success rate, recovery time, rollback data*). Hal ini menjadikan pengukuran lebih lengkap dibanding penelitian sebelumnya yang hanya berfokus pada salah satu aspek.