Lembar Jawaban UTS

Sistem Paralel dan Terdistribusi

Disusun Oleh :

Bagas Yoga Pratama Pramudika 11231014

07 Oktober 2025

Bagian Teori

1. T1

Sistem terdistribusi ditandai oleh berbagi *resource sharing*, *openness*, *distribution transparency*, *dependability*, dan *scalability*. Transparansi lokasi/replikasi/kesalahan memudahkan interaksi pengguna, tetapi selalu menimbulkan biaya kinerja dan sering bertentangan dengan skalabilitas geografis maupun administratif. Oleh karena itu, perancang harus melakukan 'bergadang' antara tingkat transparansi dan kinerja/skalabilitas yang dicapai. (Tanenbaum & Van Steen, 2023).

Untuk Pub-Sub aggregator, keunggulan utamanya adalah decoupling/pemisahan (temporal/referensial) yang memungkinkan *publisher* dan *subscriber* memiliki ketergantungan yang longgar; ini cocok untuk sistem yang berkembang secara dinamis dan topik/log dengan bervolume tinggi. Namun, decoupling ini memindahkan beban ke *broker/queue* (*matching, buffering*), sehingga menambah latensi, kompleksitas penamaan topik, dan kebutuhan observability untuk mendeteksi duplikasi/out-of-order. Trade-off penting: (i) *at-least-once via retry/ack* yang meningkatkan reliabilitas tetapi menimbulkan duplikasi; (ii) *exactly-once* sulit/mahal sehingga lebih realistis mengandalkan konsumen idempoten + *durable dedup store*; (iii) transparansi lokasi/repikasi memfasilitasi penskalaan tetapi menantang konsistensi. (Tanenbaum & Van Steen, 2023).

2. T2

Arsitektur client-server beroperasi berdasarkan prinsip *request-reply* sinkron: klien mengirim permintaan dan menunggu jawaban. Arsitektur ini sederhana dan efisien bila jaringan dapat stabil, namun kegagalan/kehilangan paket dapat memicu pengulangan permintaan yang dapat menyebabkan operasi *non-idempotent* dijalankan dua kali. (Tanenbaum & Van Steen, 2023).

Sebaliknya, model *publish-subscribe* memisahkan produsen dan konsumen melalui *subscription* pada atribut/topik; komunikasi direferensikan berdasarkan deskripsi peristiwa (bukan identitas endpoint) dan dapat dipisahkan secara temporal maupun referensial. Hal Ini mefasilitasi *multicast*, elastisitas dalam jumlah subscriber, dan *fan-out* tanpa terikat pada alamat tertentu—menjadikannya sangat cocok untuk agregasi log/telemetri. (Tanenbaum & Van Steen, 2023).

Pilih *Pub-Sub* ketika: produsen dan konsumen tidak selalu aktif secara bersamaan, penyebaran ke beberapa konsumen diperlukan, dan beban penanganan *burst* ingin dialihkan

ke *broker/queue* dengan *persistent asynchronous communication*. (Tanenbaum & Van Steen, 2023).

3. T3

   At-least-once adalah semantik pengiriman di mana sistem akan *retransmit* (mengirim ulang) sampai penerima mengakui pesan (*acknowledged*) tersebut, memastikan bahwa pesan benar-benar terkirim, tetapi pesan tersebut mungkin bisa ganda (duplikat). Duplikasi biasanya terjadi karena asumsi jaringan dan retransmisi (mis. TCP atau broker messaging), yang mengakibatkan satu pesan dapat muncul kembali setelah pengirim mengirimnya ulang. (Tanenbaum & Van Steen, 2023).

   Sebaliknya, **exactly-once** mensyaratkan bahwa setiap efek aplikasi dijalankan tepat satu kali. Secara umum, mencapai hal ini memerlukan koordinasi *end-to-end* hingga tingkat *commit* terdistribusi (mis. 2PC/3PC) untuk mencegah penerapan ganda atau hilangnya efek—pendekatan ini mahal dan rentan terhadap *blocking* saat terjadi gagal. Oleh karena itu, praktik industri biasanya mencapai "*exactly-once effects*" melalui *at-least-once* yang dikombinasikan dengan deduplikasi/idempotensi di sisi aplikasi. (Tanenbaum & Van Steen, 2023).

   Mengapa **idempotent consumer** krusial saat *retries*? Retransmisi dapat menyebabkan permintaan yang sama diproses lebih dari sekali. Operasi **idempotent** dapat diulang dengan aman karena tidak mengubah hasil setelah penerapan awal(misalnya: *read* blok file), sedangkan operasi non-idempotent (contoh: transfer uang) berisiko jika dieksekusi ulang. Dengan menandai dan menolak *event* yang sudah diproses—misalnya memakai *sequence number*/*event_id* dan *dedup store*—kondumen memastikan hanya efek pertama yang tercatat, sehingga *retries* tidak menyebabkan *over-apply*. (Tanenbaum & Van Steen, 2023).

4. T4

   Penamaan memisahkan *name/identifier/address* dan mendukung *location transparency*. Untuk *topic*, gunakan *human-friendly names* hierarkis (mis. logs.app.service.v1) untuk memastikan kemudahan pengelolaan dan menghindari pengkodean lokasi. (Tanenbaum & Van Steen, 2023).

   Untuk *event_id*, gunakan identifier yang sah (tidak dapat didaur ulang, merujuk tepat satu entitas), misalnya UUID v7 atau hash kriptografis dari (topic, producer_id, ts, nonce) untuk memperoleh keunikan dan ketahanan tabrakan; jangan menggunakan alamat yang dapat berubah. Hal ini mempermudah pengujian kesetaraan *event_id* di berbagai proses dan

menghindari ambiguitas yang timbul pada nama non-unik. (Tanenbaum & Van Steen, 2023).

Dampak terhadap deduplikasi: dengan pasangan kunci (topic, event_id) yang bersifat *location-independent*, *consumer* dapat melakukan *set-membership check* pada *durable dedup store* untuk menolak duplikasi tanpa mengetahui alamat sumber atau lokasi replika. (Tanenbaum & Van Steen, 2023).

5. T5

Tidak semua agregasi log memerlukan *total ordering*. Seringkali, *per-topic causal/partial order* sudah cukup untuk memastikan ketetapan analisis hilir. Lamport menunjukkan bahwa yang penting adalah kesepakatan urutan peristiwa (*happens-before*), bukan waktu absolut; *logical clocks* mengurutkan peristiwa ketika dibutuhkan sebab-akibat. (Tanenbaum & Van Steen, 2023).

Pendekatan praktis: sertakan timestamp (wall-clock) + *monotonic counter* lokal (atau *Lamport clock*) per publisher untuk *tie-break*; di sisi *consumer*, lakukan *stable sort* berdasarkan (topic, ts, counter) dengan toleransi *skew* (window). Kendala: jam nyata rentan *skew*; Lamport clock tidak membedakan peristiwa *concurrent* hanya berdasarkan nilai waktu; *vector clocks* menyediakan informasi kausal (sebab-akibat) yang lebih kaya tetapi meningkatkan overhead metadata. (Tanenbaum & Van Steen, 2023).

Jika suatu aplikasi benar-benar memerlukan serialisasi global (mis. debit/credit), hal itu termasuk dalam domain *sequential consistency* atau protokol konsensus; namun biaya performanya lebih tinggi. (Tanenbaum & Van Steen, 2023).

6. T6

Kegagalan jaringan/transmisi dan asumsi yang salah (reliability/latency/bandwidth /administrator tunggal) merupakan sumber utama anomali seperti duplikasi, *out-of-order*, dan *message loss*. Desain harus mengakui bahwa “jaringan tidak reliabel” dan *failure transparency* penuh tidak dapat dicapai. (Tanenbaum & Van Steen, 2023).

Mitigasi: (i) *Retry with backoff* + *ack* di MOM/AMQP untuk *at-least-once*, disertai *idempotent consumer* dan *durable dedup store* agar duplikasi dibuang; (ii) *persistent queues/durable messages* untuk menahan *crash/restart*; (iii) *flow control* dan *buffering* untuk mengatasi burst sehingga mengurangi *out-of-order* pada konsumen; (iv) *logging/metrics* agar insiden terlihat. (Tanenbaum & Van Steen, 2023).

7. T7

Dalam skala besar, banyak sistem mengadopsi *weak consistency* di mana semua replika pada akhirnya akan terkonsolidasi ketika tidak ada update—*eventual consistency*. Model ini umum dalam sistem terdistribusi yang direplikasi, DNS, dan cache web; *lazy propagation* sering kali cukup slama tidak terjadi *write-write conflicts* atau terdpaat mekanisme *winner*. (Tanenbaum & Van Steen, 2023).

Dalam sebuah *log aggregator*, *eventual consistency* dicapai ketika setiap *event* unik pada akhirnya diproses sekali dan dipublikasikan ke tampilan konsumen. *Idempotent consumer* memastikan bahwa *retry* tidak mengubah hasil; deduplikasi menolak duplikasi sehingga keadaan akhir tidak bergantung pada jumlah pengiriman ulang—hal ini memperpendek waktu konvergensi dan menghindari anomali akumulasi. Dengan demikian, kombinasi *at-least-once delivery* + *idempotency* + *dedup* menghasilkan *eventual consistency* pragmatis tanpa biaya *global total order*. (Tanenbaum & Van Steen, 2023).

8. T8

Throughput (event/detik) mencerminkan efisiensi jalur *ingest* (validasi skema, *enqueue*, *upsert* SQLite). **Latency** (p95/p99) menilai *processing delay* dari *publish* → *durable write*. **Duplicate rate** = duplicate_dropped / received mengukur dampak dari *retries* dan kualitas *producer id*; sebaiknya ≥20% dalam skenario pengujian. **Unique ratio** & **CPU/mem footprint** menunjukkan *headroom*. **Latensi deduplikasi** (lookup time) memengaruhi skala; memilih *UUIDv7/ULID* dan pengindeksan (topic, event_id) mengurangi biaya. **Uptime** dan *error rates* memvalidasi *fault handling*. Keputusan desain: *at-least-once* dipilih untuk reliabilitas; idempoten + deduplikasi untuk *exactly-once effects*; SQLite untuk *durability* lokal; FastAPI untuk *I/O-bound concurrency*. *(Bab 1–7).*

DAFTAR PUSTAKA

van Steen, M., & Tanenbaum, A. S. (2023). *Distributed systems* (4th ed., Version 4.01).