Nama : Rionando Soeksin Putra
NIM : 11221063
MK : Sistem Paralel & Terdistribusi A (2025)

SOAL TEORI

T1) Jelaskan karakteristik utama sistem terdistribusi dan trade-off yang umum pada desain Pub-Sub log aggregator.

Jawaban :
Beberapa karakteristik dari sistem terdistribusi adalah sebagai berikut,

- *Resource Sharing*
  Target dari sistem terdistribusi adalah untuk memudahkan masing masing user (dan aplikasi) untuk mengakses dan membagikan *resources* yang bersifat *remote*. *Resources* yang dimaksud ini bisa berbentuk apa saja, seperti data, file, *network*. Dengan adanya hubungan antar user dan resource in, kolaborasi juga menjadi lebih mudah (Steen & Tanenbaum, 2023, p. 10).
- *Distribution Transparency*
  Salah satu target yang ingin dicapai saat merancang sistem terdistribusi adalah bagaimana cara menyembunyikan fakta bahwa proses dan *resource*-nya itu dibagikan ke beberapa komputer. Intinya, yang dilihat oleh aplikasi itu sama di mana saja, yang perlu dibuat “transparan” adalah fakta bahwa proses yang dilakukan dan *resource* yang digunakan oleh masing masing aplikasi itu tersebar (Steen & Tanenbaum, 2023, p. 11).
- *Openness*
  Target penting lain yang ingin dicapai oleh sebuah sistem terdistribusi adalah *openness. Openness* sendiri adalah bagaimana cara, saat kita menambahkan komponen baru ke sistem yang sudah ada, agar tidak mengganggu komponen komponen yang sudah digunakan (Steen & Tanenbaum, 2023, p. 15).
- *Dependability*
  Dependability adalah tentang seberapa banyak kita bisa berharap kepada sistem terdistribusi yang digunakan. *Dependability* harus memenuhi beberapa syarat, seperti Availability (bisa langsung digunakan), *Reliability* (bisa digunakan terus menerus tanpa adanya kegagalan), dan *safety* (Saat ada komponen yang gagal, tidak ada bencana yang terjadi) (Steen & Tanenbaum, 2023, p. 18-19)..
- *Security*
  Jika sistem terdistribusi tidak *secure*, maka sistem tersebut sama saja tidak *dependable*. Hal hal yang harus diperhatikan adalah *confidentiality* (kerahasian) dan integritas data (Steen & Tanenbaum, 2023, p. 21).
- *Scalability*
  Scalability adalah kemampuan sebuah sistem terdistribusi untuk tumbuh. Tumbuh yang dimaksud ini bisa dari segi *size scalability* (menambahkan user dan resource tanpa mempengaruhi performa), *Geographical scalability* (ketika user dan resource berjarak sangat jauh, tidak terlalu terasa *delay* dalam komunikasinya), dan Administrative scalability (bisa dengan mudah di-*manage* walaupun sistem digunakan beberapa kelompok yang berbeda) (Steen & Tanenbaum, 2023, p. 24).

Salah satu tradeoffs dari pub/sub log aggregator adalah dalam hal ordering. Karena sistem merupakan sebuah sistem yang terdistribusi yang terdiri dari banyak publisher, sulit untuk menentukan ordering tentang data mana yang datang lebih dulu karena ada banyak faktor, seperti latency dan tidak adanya jam global.

T2) Bandingkan arsitektur client-server vs publish-subscribe untuk aggregator. Kapan memilih Pub-Sub? Berikan alasan teknis.
Jawaban :
Salah satu perbedaan dari arsitektur client-server dan publish-subscribe adalah pada hubungannya. Ada hubungan yang jelas antara client dan server pada arsitektur client-server, sedangkan pada arsitektur publish-subscribe, tidak ada hubungan, secara eksplisit, antara publisher dan subscriber. Hal ini, karena pub-sub menggunakan Event bus untuk menghubungkan keduanya (Steen & Tanenbaum, 2023, p. 66-70). Salah satu contoh aplikasi dimana kita lebih baik memilih pub-sub adalah ketika kita ingin membuat sebuah dashboard dengan memanggil data dari beberapa sensor secara bersamaan, karena

- Sistem memerlukan komunikasi *asynchronous* di mana sensor tidak perlu menunggu dashboard mengolah data sebelum mengirim data baru.
- Bisa scalable (horizontal) dengan lebih mudah, seperti ketika ingin menambahkan sensor baru ataupun menambahkan dashboard yang menggunakan data tersebut.

T3) Uraikan at-least-once vs exactly-once delivery semantics. Mengapa idempotent consumer krusial di presence of retries?
Jawaban :

At-least-once semantics	*Exactly-once semantics*
Terus mengirimkan request ke server sampai client menerima respon dari server. Hal ini dilakukan karena mungkin saja server crash setelah memproses request tetapi sebelum server berhasil memberikan response kembali ke client (Steen & Tanenbaum, 2023, p. 510-511).	Mengirimkan request ke server tepat hanya sekali (exactly once). Hal tersebut akan tetap dilakukan walaupun hasilnya bisa jadi gagal atau memerlukan percobaan lagi (Steen & Tanenbaum, 2023, p. 510-511).

Dalam hal ini, idempoten menjadi sangat penting. Idempoten sendiri adalah suatu operasi yang bisa dilakukan berulang ulang tetapi tetap tidak akan menyakiti sistem (Steen & Tanenbaum, 2023, p. 80). Hal ini krusial karena dalam mekanisme *at-least-once delivery semantics*, pesan atau request yang sama dapat dikirim ulang beberapa kali akibat adanya *retry* untuk menjamin pengiriman. Tanpa idempoten, pengulangan ini dapat menyebabkan efek samping yang tidak diinginkan, seperti duplikasi data dan beban tambahan pada server. satu cara yang paling simple adalah dengan mengirimkan *event_id* di dalam payloadnya. Jika *event_id,* sudah pernah tercatat, maka request tidak akan dilakukan lagi walaupun client tetap melakukan request berkali kali. Dengan demikian, sistem bisa tetap aman, walaupun *client* mengirimkan *request* berulang kali.

T4) Rancang skema penamaan untuk topic dan event_id (unik, collision-resistant). Jelaskan dampaknya terhadap dedup.
Jawaban :
Topic = {topic}/{location}/{type}-{4-digit-numbers}
contohnya

- sensor/E103/temperature-0001
- web/lms-itk/scrape-0002

event_id = {topic}/{timestamp-utc}/{uuid}
contohnya

- sensor/E103/temperature-0001/2025-10-23T20:51:30Z/550e8400-e29b-41d4-a716-446655440000
- web/lms-itk/scrape-0002/2025-10-23T20:51:30Z/550e8400-e29b-41d4-a716-446655440000

Hal ini penting untuk dedup karena dengan adanya id yang unik dan collision-resistant, kemungkinan data baru ditolak karena sudah ada *topic* dan *event_id* yang sama di dalam dedup store menjadi sangat kecil dan bahkan mungkin tidak ada. Karena, jika *topic* dan *event_id* terlalu umum, akan ada kemungkinan data baru ditolak karena memiliki kombinasi *topic* dan *event_id* yang sama dengan yang sudah ada di *dedup store.*

T5) Bahas ordering: kapan total ordering tidak diperlukan? Usulkan pendekatan praktis (mis. event timestamp + monotonic counter) dan batasannya.
Jawaban :
Salah satu kasus dimana total ordering tidak perlu adalah ketika masing masing event bersifat independen (Steen & Tanenbaum, 2023, p. 274).. Contohnya, ketika kita ingin mengembangkan sistem monitoring suatu ruangan menggunakan 3 sensor (Suhu, kelembapan, dan gas). Urutan event suhu dan event kelembapan ini tidak perlu penting ordernya, karena keduanya diolah oleh client yang berbeda. Jadi, hanya dibutuhkan order per topik. Salah satu, pendekatan yang bisa dilakukan adalah kombinasi timestamp dan counter (cth : sensor1-20251023T200523Z-00057). Tetapi, ketika ada berbagai sumber, bisa menyebabkan timestamp menjadi tidak terlalu akurat.

T6) Identifikasi failure modes (duplikasi, out-of-order, crash). Jelaskan strategi mitigasi (retry, backoff, durable dedup store).
Jawaban :

Type of failure	*Description*	*Mitigation*
Crash	*Halts, but is working correctly until it halts* (Steen & Tanenbaum, 2023, p. 467).	Implementasi checkpoint agar sistem dapat melanjutkan dari state terakhir sebelum *crash*

Omission failure	*Fails to respond to incoming requests* *Fails to receive incoming messages* *Fails to send messages* (Steen & Tanenbaum, 2023, p. 467).	Gunakan *timeout* + *retry mechanism* dengan *exponential backoff* untuk mengirim ulang pesan yang hilang.
Timing failure	*Response lies outside a specified time interval* (Steen & Tanenbaum, 2023, p. 467).	Implementasikan QoS (*Quality of Service*) agar *latency* dapat dijaga di bawah batas tertentu.
Response failure	*Response is incorrect* *The value of the response is wrong* *Deviates from the correct flow of control* (Steen & Tanenbaum, 2023, p. 467).	Gunakan *data validation & sanity check* sebelum memproses hasil.

T7) Definisikan eventual consistency pada aggregator; jelaskan bagaimana idempotency + dedup membantu mencapai konsistensi.
Jawaban :
Eventual consistency adalah sebuah kemampuan sebuah sistem terdistribusi yang di mana, jika tidak ada perubahan dalam waktu yang lama, semua komponen akan, pada suatu waktu, menjadi konsisten (data yang tersimpan sama) (Steen & Tanenbaum, 2023, p. 407).
Untuk mencapai konsistensi, sistem bisa menggunakan beberapa hal, seperti

- Idempoten, yang memastikan, bahwa operasi yang sama dapat dieksekusi berulang kali tanpa mengubah hasil akhirnya. Jadi walaupun ada kegagalan saat pengiriman dan dilakukan pengiriman ulang, hasil akhirnya akan tetap sama dari 1 request ke request lain.
- Deduplication, yang memastikan, bahwa sistem tidak akan melakukan proses pada event yang sudah pernah diproses. Hal ini dilakukan dengan menyimpan kombinasi dari *topic & event_id* yang diharapkan akan selalu unik dari tiap request.

T8) Rumuskan metrik evaluasi sistem (throughput, latency, duplicate rate) dan kaitkan ke keputusan desain.
Jawaban :

Metrics	*Definition*	*Design*
Throughput	Jumlah event yang dapat dikelola sistem dalam satu waktu	Penggunaan *asyncio* dan *FastAPI* agar bisa melakukan proses asinkronus. *Batch publishing* pada

		endpoint /publish bisa meningkatkan throughput karena bisa mengurangi overhead per request
Latency	Waktu yang dibutuhkan untuk memproses suatu event	Penggunaan database berupa SQLite, sehingga bisa mengurangi overhead pada jaringan.
Dupilcate Rate	Persentase event duplikat yang dapat diiterima sistem dibandingkan dengan total semua event yang dikirimkan.	Penggunaan kombinasi *(topic, event_id)* sebagai kombinasi *primary key* untuk mencegah adanya duplikasi. Penggunaan *database* (SQLite) sebagai *dedup store* untuk memastikan idempotensi walalupun sistem *crash*.

Daftar Pustaka

Steen, M. van, & Tanenbaum, A. S. (2023). Distributed Systems (Fourth edition, version 4.01 (January 2023)). Maarten van Steen.