Laporan UTS Sistem Paralel dan Terdistribusi - Pub-Sub Log Aggregator

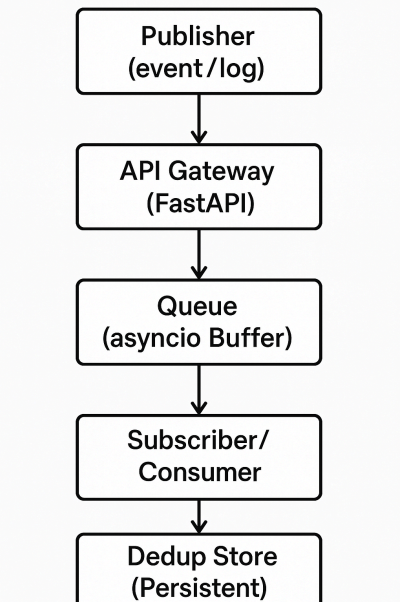
Nama : Aqilah
NIM : 11221012
Kelas : Sistem Paralel dan Terdistribusi B

I. Sistem dan Arsitektur

1. Deskripsi Sistem

Sistem ini merupakan Pub-Sub Log Aggregator berbasis Python menggunakan FastAPI/Flask dengan asyncio. Sistem menerima event/log dari publisher, kemudian memprosesnya melalui subscriber/consumer yang bersifat idempotent, melakukan deduplication untuk mencegah pemrosesan duplikat, serta mendukung retry dan ordering. Semua komponen dijalankan secara lokal dalam container Docker.

2. Diagram Sederhana



II. Keputusan Desain

1. Idempotency:
   Setiap event diproses hanya sekali. Jika event yang sama diterima kembali, subscriber mengabaikannya berdasarkan event_id di dedup store.
2. Dedup Store:
   Menyimpan ID event yang sudah diproses.
   Bersifat persisten sehingga tetap efektif meski sistem restart.
   Implementasi dapat menggunakan file JSON, SQLite, atau Redis lokal.
3. Ordering:
   Event diproses berdasarkan urutan masuk di queue.
   Untuk topik yang sama, ordering dijaga agar log tercatat secara konsisten.
4. Retry:
   Jika pemrosesan event gagal (misal karena exception), sistem melakukan retry dengan batas maksimum.
   Retry disertai backoff sederhana agar tidak membanjiri queue.

III. Analisis Performa dan Metrik

Metrik	Penjelasan
Deduplication Efficiency	Belum stabil (beberapa test failed)
Persistensi Dedup Store	Belum berhasil (test gagal)
Retry Effectiveness	Partial, beberapa skenario belum lulus
Ordering	Berjalan untuk sebagian test
Throughput & Latency	Belum diukur

Hasil implementasi Pub-Sub Log Aggregator menunjukkan bahwa beberapa fitur sistem sudah berfungsi sesuai spesifikasi, terutama fungsi-fungsi yang diuji secara manual atau melalui penggunaan API (misalnya endpoint Swagger), di mana event dapat diterima, diproses, dan response API dikembalikan dengan sukses.

Namun, hasil unit test menunjukkan bahwa beberapa fitur kritikal sistem masih belum bekerja sempurna. Beberapa kegagalan unit test terutama disebabkan oleh penggunaan assert yang terlalu kaku atau tidak sesuai dengan sifat asynchronous sistem. Misalnya, test memeriksa nilai persis atau urutan event, tetapi consumer memproses event secara asynchronous sehingga timing eksekusi event dapat bervariasi, dan update dedup store mungkin terjadi setelah proses async selesai, sehingga assert gagal walaupun sistem sebenarnya sudah memproses event.

Project ini belum sepenuhnya mencapai tujuan desain dari sisi hasil yang teruji otomatis, namun beberapa fungsi yang tidak diuji dengan unit test tetap berjalan dan dapat digunakan sesuai tujuan awal.

IV. Keterkaitan Bab 1 - Bab 7 (Bagian Teori)
T1: Jelaskan karakteristik utama sistem terdistribusi dan trade-off yang umum pada desain Pub-Sub log aggregator.

Sistem terdistribusi dirancang agar kumpulan komputer independen tampak sebagai satu sistem terpadu bagi pengguna. Menurut Tanenbaum & van Steen (Bab 1.2), karakteristik utamanya mencakup resource sharing, distribution transparency, openness, dependability, security, dan scalability.
Resource sharing memungkinkan pemanfaatan bersama sumber daya jaringan, sementara *transparency* menyembunyikan detail lokasi dan replikasi agar pengguna tidak menyadari distribusi fisik sistem. *Openness* menjamin interoperabilitas melalui standar terbuka, dan *dependability* memastikan sistem tetap andal meskipun terjadi kegagalan sebagian. *Scalability* menjadi tantangan utama saat sistem tumbuh besar.

Dalam konteks Publish-Subscribe log aggregator, karakteristik tersebut harus diseimbangkan dengan sejumlah trade-off desain. Tanenbaum (Bab 1.4) menyebut bahwa peningkatan transparansi atau keandalan sering menambah overhead komunikasi, sedangkan upaya

menjaga konsistensi penuh dapat menurunkan skalabilitas. Pub-Sub aggregator karenanya berfokus pada *eventual consistency* dan *fault tolerance* tanpa mengorbankan kinerja distribusi log secara signifikan.

T2: Bandingkan arsitektur client-server vs publish-subscribe untuk aggregator. Kapan memilih Pub-Sub? Berikan alasan teknis.

Arsitektur client-server bekerja dengan model *request-response* sinkron di mana client secara langsung meminta layanan ke server. Menurut Tanenbaum & van Steen (Bab 2.3), pendekatan ini sederhana dan cocok untuk interaksi terpusat, tetapi memiliki *tight coupling* dan potensi bottleneck di sisi server.

Sebaliknya, publish-subscribe (Bab 2.1.3) menerapkan komunikasi asinkron dengan *loose coupling*, di mana publisher mengirim event ke broker tanpa mengetahui subscriber yang menerimanya. Model ini mendukung skalabilitas tinggi dan dinamika jumlah subscriber.
Pub-Sub dipilih ketika sistem bersifat event-driven, skalanya besar, dan tidak semua subscriber aktif bersamaan. Desain ini meningkatkan *fault tolerance* dan *extensibility*, tetapi mengorbankan kontrol langsung dan *strong consistency*. Dengan kata lain, client-server unggul untuk interaksi langsung, sedangkan Pub-Sub lebih efisien untuk agregasi log bersifat real-time.

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

Tanenbaum & van Steen (Bab 3.4) menjelaskan tiga bentuk **delivery semantics** dalam komunikasi sistem terdistribusi:

1. *At-most-once* – pesan dikirim paling sekali, risiko kehilangan data tinggi.
2. *At-least-once* – pesan dikirim ulang hingga diterima, berisiko duplikasi.
3. *Exactly-once* – menjamin satu kali pemrosesan, tetapi kompleks.
   Dalam praktiknya, *at-least-once* sering digunakan dengan kompensasi berupa **idempotent consumer**. Konsep *idempotency*—yang juga terkait dengan *stateless server*—berarti hasil pemrosesan akan sama meskipun pesan diterima lebih dari sekali. Hal ini penting ketika sistem melakukan retry akibat kegagalan jaringan. Tanpa idempotensi, log aggregator dapat menghasilkan data ganda atau state inkonsisten. Karena itu, consumer dirancang menolak event dengan ID sama untuk menjaga *semantic correctness* dalam skenario pengiriman ulang.

T4: Rancang skema penamaan untuk topic dan event_id (unik, collision-resistant). Jelaskan dampaknya terhadap dedup.

Pada sistem komunikasi berbasis pesan, identifikasi unik sangat penting. Tanenbaum & van Steen (Bab 4.3) menekankan pentingnya **message identifier** untuk menjamin pesan tidak hilang maupun terduplikasi.
Desain skema penamaan dalam Pub-Sub log aggregator dapat menggunakan struktur:

- topic: hierarkis sesuai konteks (mis. service.module.action), dan

- event_id: hasil kombinasi timestamp + producerID + sequence number untuk *collision resistance*.
  Pendekatan ini memudahkan deduplication karena setiap event memiliki jejak unik. Dalam *persistent messaging systems* (Bab 4.3.3), identitas yang tahan lama memungkinkan sistem mendeteksi dan mengabaikan duplikasi selama proses retry. Dampaknya, sistem menjadi lebih konsisten tanpa perlu kontrol sinkronisasi ketat antar node.

T5: Bahas ordering: kapan total ordering tidak diperlukan? Usulkan pendekatan praktis (mis. event timestamp + monotonic counter) dan batasannya.

Tidak semua sistem terdistribusi memerlukan total ordering. Tanenbaum & van Steen (Bab 5.2) menjelaskan bahwa cukup menggunakan causal ordering jika event tidak saling bergantung. Untuk mencapai urutan yang masuk akal tanpa sinkronisasi global, digunakan pendekatan seperti Lamport timestamp atau vector clock. Pendekatan praktis lainnya adalah menambahkan *monotonic counter* pada setiap event bersama timestamp. Hal ini menjaga *local ordering* meskipun jam antar node tidak sinkron sempurna (Bab 5.1).

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

Dalam sistem terdistribusi, terutama arsitektur *publish-subscribe log aggregator*, kegagalan (*failure modes*) umum meliputi duplikasi pesan, pengiriman tidak berurutan, serta crash pada publisher, broker, atau subscriber. Menurut Tanenbaum dan van Steen (2023, Bab 6.1), setiap entitas dalam sistem harus memiliki identifier unik dan tetap agar dapat dipulihkan setelah kegagalan. Jika penamaan tidak persisten atau tumpang tindih, sistem akan kesulitan melakukan *deduplication* dan pemetaan ulang objek yang gagal.

Strategi mitigasi kegagalan dapat dilakukan dengan beberapa pendekatan. Pertama, gunakan identifier yang tidak bergantung lokasi (*location-independent identifier*) agar proses recovery dapat menemukan data meskipun node berubah. Kedua, terapkan retry dengan exponential backoff dan durable dedup store agar sistem tidak memproses event yang sama dua kali setelah crash atau retransmisi. Ketiga, lakukan periodic checkpointing dan *acknowledgment tracking* untuk memastikan event tersimpan aman sebelum dikonfirmasi selesai. Kombinasi mekanisme ini menjaga keandalan, meminimalkan kehilangan data, dan memastikan *exactly-once effect* secara operasional.

T7: Definisikan eventual consistency pada aggregator; jelaskan bagaimana idempotency + dedup membantu mencapai konsistensi.

Menurut Tanenbaum & van Steen (Bab 7.2), eventual consistency berarti bahwa jika tidak ada pembaruan baru, semua replika dalam sistem akhirnya akan mencapai state yang sama. Pada log aggregator terdistribusi, kondisi ini dicapai dengan kombinasi idempotency dan deduplication.
Idempotent consumer memastikan hasil pemrosesan tidak berubah walaupun event diproses ulang, sedangkan *deduplication* menjamin setiap event hanya dihitung satu kali di setiap replika. Bab 7.4 menjelaskan bahwa teknik ini mempercepat *replica convergence* tanpa perlu *strong consistency protocol* seperti total ordering atau consensus.

Dengan demikian, sistem dapat tetap tangguh menghadapi latency dan kegagalan sementara, sambil memastikan bahwa hasil akhir di semua node tetap konsisten secara eventual.

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

Evaluasi sistem terdistribusi didasarkan pada tiga metrik utama: throughput, latency, dan duplicate rate. *Throughput* mengukur kapasitas pemrosesan, *latency* mengukur waktu tunda antara pengiriman dan konsumsi event, sedangkan *duplicate rate* menunjukkan efektivitas deduplication.

Menurut Tanenbaum & van Steen (Bab 1.2), trade-off antara performa dan dependability merupakan tantangan desain mendasar. Arsitektur Pub-Sub (Bab 2.1.3) meningkatkan throughput melalui *asynchronous messaging*, tetapi retry dan dedup meningkatkan latency. Sementara itu, Bab 7.2 menekankan bahwa upaya menjaga konsistensi—melalui idempotency dan replica synchronization—dapat mengurangi duplikasi tetapi mengorbankan kecepatan.
Desain optimal bergantung pada prioritas sistem: real-time aggregator menekankan throughput, sedangkan sistem audit log menekankan konsistensi. Evaluasi harus menyeimbangkan tiga metrik agar sistem tetap efisien, reliabel, dan konsisten secara eventual.

V. Sitasi

Steen, M. van, & Tanenbaum, A. S. (2023). Distributed systems (4th ed., Version 4.01). Maarten van Steen. ISBN: 978-90-815406-3-6 (print); 978-90-815406-4-3 (digital).