

Nama : Gusti Muhammad Risandha

NIM : 11221028

UTS SISTEM TERDISTRIBUSI

Tema: Pub-Sub Log Aggregator dengan Idempotent Consumer dan Deduplication

A. Bagian Teori

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

Jawaban:

Menurut Coulouris et al. (2012, Bab 1), karakteristik utama dalam sistem terdistribusi adalah sebagai berikut:

- a) **Concurrency of Components**, dimana ada banyak komponen yang saling berinteraksi, melakukan eksekusi secara bersamaan, hingga saling berbagi sumber daya seperti file atau halaman web sehingga membutuhkan sinkronisasi dan juga koordinasi antar proses.
- b) **Lack of a Global Clock**, tidak adanya pandangan global atau jam global untuk mengatur sebuah koordinasi sehingga sistem harus mengandalkan algoritma logical clocks untuk menentukan urutan kejadian.
- c) **Independent Failures**, sistem terdistribusi memiliki cara kegagalan yang unik karena setiap komponen dalam sistem dapat gagal secara independen sementara yang lain akan terus berjalan tanpa mematikan keseluruhan sistem. Bahkan, program-program mungkin tidak dapat mendeteksi apakah jaringan benar-benar gagal atau hanya menjadi sangat lambat.

Kemudian, Van Steen dan Tanenbaum (2023, Bab 1) menjelaskan bahwa dalam merancang sistem terdistribusi terdapat berbagai trade-off antar karakteristik desain. Misalnya peningkatan dependability melalui replikasi data akan menambah keandalan, tetapi dapat menurunkan performance akibat meningkatnya beban sinkronisasi. Serta contoh lainnya adalah peningkatan distribution transparency untuk menyembunyikan lokasi sumber daya sering mengorbankan efisiensi karena menambah lapisan komunikasi. Konsep ini juga berlaku dalam desain Pub-Sub log

aggregator dimana sistem ini berusaha mencapai scalability dan fault tolerance dengan menambah replikasi dan penyimpanan pesan, tetapi hal tersebut berdampak pada meningkatnya latency serta berkurangnya consistency.

2. T2 (Bab 2): Bandingkan arsitektur client-server vs publish-subscribe untuk aggregator. Kapan memilih Pub-Sub? Berikan alasan teknis.

Jawaban:

Perbedaan antar kedua arsitektur yaitu *client-server* dan *publish-subscribe* dapat dilihat dari sifat masing-masing arsitektur itu sendiri. Arsitektur *client-server* bersifat terpusat dan menekankan hubungan langsung antara *client* dan *server*, dimana *server* menyediakan layanan tertentu dan klien melakukan permintaan layanan tersebut. Model ini mudah diimplementasikan, tetapi memiliki keterbatasan dalam hal *scalability* dan *fault tolerance* sehingga ketika jumlah klien meningkat, beban pada server dapat menyebabkan *bottleneck* dan menurunkan kinerja sistem (Coulouris et al., 2012, Bab 4 dan Bab 6). Sebaliknya, arsitektur Publish-Subscribe (Pub-Sub) memungkinkan komunikasi *asynchronous* dan *decoupled*, di mana publisher tidak perlu mengetahui identitas subscriber. Komponen perantara (*broker*) menangani penyebaran event berdasarkan topik atau kategori tertentu, sehingga cocok untuk sistem berskala besar dan dinamis.

Van Steen & Tanenbaum (2023, Bab 2) menjelaskan bahwa arsitektur *publish-subscribe* lebih cocok digunakan ketika sistem memerlukan *high scalability*, *asynchronous event dissemination*, dan *loose coupling* antar komponen. Untuk aggregator log, *Pub-Sub* dipilih saat volume data tinggi dan banyak konsumen data yang berubah-ubah. Sehingga *broker* atau *overlay network* dapat menangani distribusi *event* tanpa harus menambah beban komunikasi pada satu server.

3. T3 (Bab 3): Uraikan at-least-once vs exactly-once delivery semantics. Mengapa idempotent consumer krusial di presence of retries?

Jawaban:

Dalam sistem terdistribusi terdapat beberapa *delivery semantics*, antara lain *at-least-once* dan *exactly-once delivery semantics*. *at-least-once semantics* akan menjamin bahwa *remote procedure* akan dieksekusi sedikitnya sebanyak 1 kali. Jaminan diberikan dengan mengirimkan ulang pesan hingga penerima mengkonfirmasi penerimaannya. Kelemahannya adalah server dapat menerima

dan mengeksekusi prosedur yang sama lebih dari satu kali jika pesan balasan hilang, yang berpotensi menghasilkan nilai yang salah (*wrong values*). Sebaiknya, *exactly-once delivery semantics*. menjamin bahwa pesan diproses tepat satu kali dan kesalahan dilaporkan jika pesan tidak dapat dikirim (Coulouris et al., 2012, Bab 5).

Idempotent customer sangat penting dalam sistem yang menerapkan retries (salah satunya adalah yang menggunakan at-least-once semantics) karena sifatnya yang dapat mentoleransi pengiriman pesan duplikat. Masalah muncul ketika sistem menggunakan *retry mechanism* untuk menjamin pengiriman ulang. Jika *consumer* tidak dirancang *idempotent*, maka setiap pengulangan eksekusi dapat menghasilkan efek yang salah. Oleh Karena itu, idempotent consumer menjadi krusial untuk menjaga consistency di presence of retries (Van Steen & Tanenbaum, 2023, Bab 3).

4. T4 (Bab 4): Rancang skema penamaan untuk topic dan event_id (unik, collision-resistant). Jelaskan dampaknya terhadap dedup.

Jawaban:

Rancang skema penamaan dalam sistem terdistribusi untuk topic dan event_id sangat penting. Biasanya setiap *event* yang dikirim perlu memiliki *identifier* yang unik agar sistem dapat melakukan *deduplication* dengan benar. Menurut Van Steen dan Tanenbaum (2023, Bab 6), sistem penamaan terdiri dari names, identifiers, dan addresses. Identifier harus bersifat unik dan tidak berubah, sedangkan names dapat digunakan untuk mengarahkan ke entitas yang memiliki makna semantik. Skema penamaan topic sebaiknya menggunakan *hierarchical namespace* mirip DNS. Skema ini secara inheren mengelola namespace untuk mencegah *collision* dan memungkinkan pemfilteran berbasis *wildcard* yang efisien pada sisi konsumen. Pendekatan ini sangat cocok untuk sistem *publish-subscribe* yang sering diimplementasikan di atas *Message-Oriented Middleware* (MOM), sebuah model komunikasi yang dibahas dalam Bab 4 buku referensi (van Steen & Tanenbaum, 2023).

Untuk skema penamaan event_id dapat menggunakan pendekatan flat *naming* dengan *hash-based identifier*. Kemudian event_id dapat dibuat dengan menggabungkan beberapa elemen seperti ID yang unik (misalnya, UUID) dan juga timestamp beresolusi tinggi. Dampaknya terhadap *deduplication* adalah meningkatkan keandalan deteksi duplikasi karena setiap event_id sekarang memiliki *unique identifier*. Hal ini secara efektif membuat pemrosesan pesan menjadi *idempotent*, di mana pemrosesan berulang dari pesan yang

sama tidak akan mengubah hasil akhir. Kemampuan untuk menerapkan operasi secara konsisten dan terurut termasuk memastikan operasi tidak diterapkan lebih dari sekali adalah fondasi untuk mencapai model konsistensi yang kuat seperti sequential consistency (van Steen & Tanenbaum, 2023, Bab 7).

5. T5 (Bab 5): Bahas ordering: kapan total ordering tidak diperlukan? Usulkan pendekatan praktis (mis. event timestamp + monotonic counter) dan batasannya.

Jawaban:

Total ordering tidak diperlukan ketika aplikasi tidak memerlukan keseragaman urutan global antar seluruh node, melainkan hanya urutan akibat antar peristiwa yang saling bergantung. Dalam konteks ini, Lamport clocks dan vector clocks memberikan solusi efisien untuk mempertahankan causal consistency tanpa membebani sistem dengan kebutuhan sinkronisasi penuh terhadap waktu global (van Steen & Tanenbaum, 2023, Bab 5)..

Pendekatan praktis yang sering digunakan adalah variasi dari Lamport clocks, di mana setiap peristiwa ditandai dengan pasangan unik $\langle \text{timestamp}, \text{process_id} \rangle$. Timestamp berasal dari jam logis lokal, sementara ID proses yang unik berfungsi sebagai counter untuk menjamin keunikan dan memecah kesamaan waktu. Metode ini efektif untuk menjaga relasi happens-before ($a \rightarrow b$ berimplikasi $C(a) < C(b)$). Tetapi bagaimanapun urutan antar node dapat menjadi ambigu (concurrent events) dalam kasus delay network yang tinggi atau drift jam lokal. Selain itu, sistem tidak dapat menjamin pesanan total global; untuk aplikasi chat atau penggabungan sensor data, hanya pesanan bagian yang cukup (van Steen & Tanenbaum, 2023, Bab 5).

6. T6 (Bab 6): Identifikasi failure modes (duplikasi, out-of-order, crash). Jelaskan strategi mitigasi (retry, backoff, durable dedup store).

Jawaban:

Dalam sistem terdistribusi, terdapat tiga *failure modes* utama yaitu *duplication*, *out-of-order delivery*, dan *crash failures*. *Duplication failure* terjadi ketika pesan dikirim ulang akibat retransmisi jaringan atau kegagalan *acknowledgment*, di mana pengirim yang tidak menerima konfirmasi akan mengirim ulang pesan meskipun pesan pertama mungkin telah sampai. Hal ini dapat menyebabkan operasi dijalankan berulang kali, yang berbahaya bila operasi tidak bersifat idempoten (Coulouris et al., 2012, Bab 5).

Sementara itu, *out-of-order delivery* muncul ketika pesan diterima tidak sesuai urutan pengiriman akibat perbedaan latensi jaringan. Dalam kondisi ini, paket yang dikirim terakhir dapat tiba lebih dahulu, menyebabkan status sistem menjadi tidak konsisten apabila urutan pesan penting bagi logika aplikasi (Tanenbaum & Van Steen, 2023, Bab 8).

Selanjutnya, *crash failure* terjadi ketika proses berhenti tiba-tiba akibat gangguan perangkat keras, bug perangkat lunak, atau kehabisan sumber daya sehingga layanan yang disediakan menjadi tidak tersedia. Untuk memitigasi berbagai kegagalan tersebut, digunakan beberapa strategi seperti *retry* dengan *exponential backoff* untuk menghindari pembanjiran jaringan akibat percobaan ulang terus-menerus, serta penggunaan *durable deduplication store* yang mencatat setiap pesan dengan *unique ID* agar pesan duplikat dapat diabaikan. Selain itu, penambahan *sequence numbering* atau *timestamp* dapat menjaga urutan pesan tetap konsisten. Pendekatan-pendekatan ini memastikan sistem tetap reliabel dalam menghadapi duplikasi, ketidakteraturan urutan, maupun kegagalan proses (Tanenbaum & Van Steen, 2023, Bab 8).

7. T7 (Bab 7): Definisikan eventual consistency pada aggregator; jelaskan bagaimana idempotency + dedup membantu mencapai konsistensi.

Jawaban:

Eventual consistency dalam sistem terdistribusi, terutama dalam aggregator log atau sistem replikasi data, berarti bahwa meskipun pembaruan tidak langsung dapat dilihat oleh semua node, sistem akan mencapai keadaan konsisten setelah beberapa waktu tanpa menerima pembaruan baru. Dengan kata lain, semua replika akhir akan menyimpan data yang sama (semua replika konvergen ketika tidak ada perbaikan) (Van Steen & Tanenbaum, 2023, Bab 7). Sistem berskala besar seperti pengumpul log dan database terdistribusi sering menggunakan model ini untuk menjaga kinerja dan ketersediaan meskipun terjadi latency atau kegagalan jaringan.

Dua prinsip utama, *idempotency* dan *deduplication*, digunakan untuk mendukung konsistensi yang disebutkan di atas. Idempotensi menjamin bahwa melakukan operasi sekali lagi akan memiliki hasil yang sama seperti saat pertama kali dilakukan. Ketika pesan dikirim ulang karena retry atau failover di network, hal ini mencegah hasil diduplikasi dalam aggregator. Meskipun demikian, deduplication

mendeteksi dan mengabaikan pesan yang sudah diproses sebelumnya dengan menggunakan penyimpanan atau indeks unik, seperti pesan ID (Coulouris et al., 2012, Bab 5). Kombinasi keduanya menjamin bahwa meskipun data dikirim berkali-kali atau diterima dalam urutan berbeda, hasil agregasi tetap konsisten tanpa kehilangan atau pengulangan data.

8. T8 (Bab 1–7): Rumuskan metrik evaluasi sistem (throughput, latency, duplicate rate) dan kaitkan ke keputusan desain.

Jawaban:

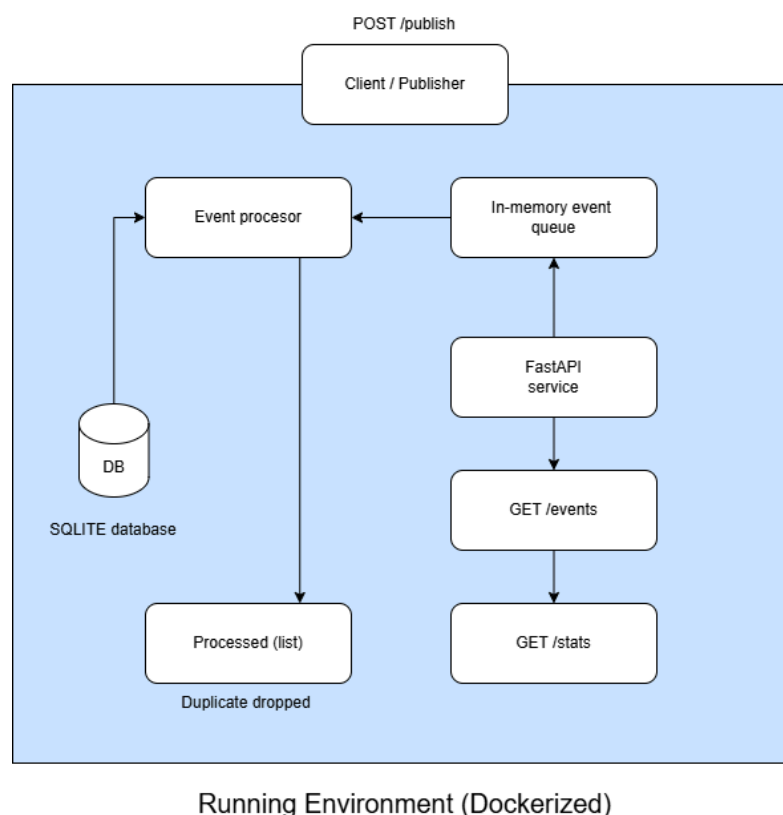
Throughput, *latency*, dan *duplicate rate* merupakan tiga metrik utama untuk menilai kinerja serta keandalan sistem terdistribusi. *Throughput* mengukur jumlah operasi yang dapat diproses per satuan waktu, mencerminkan kemampuan sistem menangani beban paralel. Van Steen & Tanenbaum (2023, Bab 1) menjelaskan bahwa peningkatan scalability dan efisiensi komunikasi berdampak langsung pada throughput, sehingga arsitektur asynchronous, load balancing, dan batch processing sering dipilih untuk meningkatkannya. Latency mengukur jeda antara pengiriman dan penerimaan pesan; penyebab utamanya adalah network propagation, antrian, dan waktu pemrosesan. Coulouris et al. (2012, Bab 2) menegaskan bahwa desain low-latency seperti co-located servers atau RPC optimization dapat menurunkan delay, meski sering mengorbankan throughput dan toleransi kesalahan. Sementara itu, duplicate rate menunjukkan frekuensi pesan yang diterima lebih dari sekali akibat retry atau kegagalan jaringan, sehingga dibutuhkan deduplication mechanisms dan idempotent operations untuk menjaga konsistensi data (Coulouris et al., 2012, Bab 5).

Ketiga metrik evaluasi sistem memiliki hubungan sebab-akibat langsung dan saling mempengaruhi keputusan desain arsitektur sistem terdistribusi. Arsitektur asynchronous seperti Kafka meningkatkan *throughput* melalui *parallelism*, namun menambah *latency* akibat antrian pesan (Van Steen & Tanenbaum, 2023, Bab 4). Sebaliknya, desain synchronous RPC menekan latency dengan menempatkan layanan berdekatan, tetapi mengorbankan skalabilitas (Coulouris et al., 2012, Bab 5). Untuk mengendalikan duplikasi akibat *at-least-once delivery*, sistem menerapkan *deduplication store* dan operasi idempotent, yang meningkatkan reliabilitas namun sedikit menambah waktu akses. Karena itu, desain sistem selalu menjadi kompromi antara *throughput*, *latency*, dan konsistensi data.

B. Desain dan Implementasi Sistem

1. Ringkasan sistem dan arsitektur

Sistem yang dibangun adalah sistem layanan *pub-sub log aggregator* yang berfungsi sebagai pusat untuk menerima dan memproses event atau log dari berbagai sumber (penerbit). Tujuan utama sistem ini adalah untuk memastikan bahwa setiap event unik hanya diproses tepat satu kali, meskipun diterima berkali-kali. Ini dicapai dengan menerapkan konsumen *idempotent* dengan mekanisme deduplikasi persisten.



Arsitektur sistem ini mengadopsi pola *producer-consumer* yang dipisahkan oleh antrian internal (in-memory queue) untuk meningkatkan responsivitas dan ketahanan sistem. Berikut adalah alur kerja pemrosesan sebuah event: Penerimaan *Event* berlangsung ketika sebuah client atau publisher mengirimkan event dalam format JSON ke endpoint **POST /publish** pada layanan *FastAPI*. *Endpoint* ini bertugas memvalidasi skema data yang masuk. Kemudian setelah validasi berhasil, *event* tidak langsung diproses. Sebaliknya, ia dimasukkan ke dalam sebuah antrian internal (*asyncio.Queue*). Langkah ini memungkinkan API untuk segera memberikan respons

kembali ke *client* tanpa harus menunggu proses yang lebih lama, seperti interaksi dengan *database*. Lalu sebuah *background task* (*event_processor*) berjalan secara terus-menerus di dalam layanan untuk mengambil event satu per satu dari *asyncio.Queue* ketika antrian tidak kosong. Sebelum memproses event lebih lanjut, worker akan berinteraksi dengan dedup store yang diimplementasikan menggunakan SQLite. *Worker* akan mencoba mencatat kombinasi unik (*topic*, *event_id*) dari event tersebut ke dalam database. Jika *event* dianggap unik maka akan diproses, jika dianggap duplikat maka akan diabaikan. Sebelum memproses event lebih lanjut, worker akan berinteraksi dengan dedup store yang diimplementasikan menggunakan SQLite. *Worker* akan mencoba mencatat kombinasi unik (*topic*, *event_id*) dari event tersebut ke dalam database.

2. Keputusan desain teknis

Pemilihan teknologi berdasarkan pada kebutuhan sistem, framework yang digunakan adalah FastAPI karena bersifat asinkron dan cocok untuk menangani operasi *I/O-bound* seperti menerima HTTP request dan memasukan data kedalam database. Server yang menangani banyak permintaan bisa menjadi tidak efisien jika harus menunggu operasi yang bersifat *blocking*, sehingga dengan menggunakan operasi *asynchronous* (*nonblocking*), server tidak perlu berhenti total saat menunggu database selesai menulis (Tanenbaum & Van Steen, 2023).

Untuk mencapai toleransi kegagalan (*fault tolerance*) terhadap duplikasi pesan, sistem ini mengimplementasikan consumer yang bersifat *idempotent*, yaitu sebuah properti di mana sebuah operasi dapat diulang berkali-kali tanpa menimbulkan efek samping atau kerusakan setelah eksekusi pertama. Pendekatan ini mengubah semantik pengiriman pesan dari *at-least-once* menjadi semantik *exactly-once* secara fungsional. Dalam praktiknya, idempotensi ini dicapai dengan mendelegasikan logika deduplikasi ke level database. Sebuah composite primary key pada kolom (*topic*, *event_id*) memastikan bahwa setiap pesan hanya dapat diproses satu kali secara atomik. Jika sebuah pesan duplikat (dengan *topic* dan *event_id* yang sama) tiba, operasi INSERT akan gagal dengan *IntegrityError*, yang menandakan bahwa pesan tersebut telah diproses sebelumnya (Tanenbaum & Van Steen, 2023)

Sistem ini menggunakan *asyncio.Queue*, yang secara inheren bersifat First-In, First-Out (FIFO). Artinya, dalam kondisi normal dan dengan satu *worker*, *event* akan diproses sesuai urutan kedatangannya di endpoint */publish*. Ini adalah bentuk

ordering yang sederhana dan setara dengan *FIFO-ordered multicast* dalam konteks satu pengirim, di mana pesan dari satu proses akan di-deliver sesuai urutan pengirimannya (van Steen & Tanenbaum, 2023). Penggunaan antrian (queue) juga merupakan pola fundamental dalam Message-Oriented Middleware (MOM) yang memungkinkan komunikasi asinkron dan loosely-coupled antara komponen pengirim dan penerima (Coulouris et al., 2012).

Desain ini menyederhanakan server dan mengandalkan klien untuk menangani kegagalan pengiriman. Jika klien mengirim request dan terjadi network error atau server tidak merespons, klien bertanggung jawab untuk mengimplementasikan logika retry. Seperti yang dijelaskan oleh van Steen & Tanenbaum (2023), melakukan retry secara naif bisa berbahaya karena klien tidak tahu apakah kegagalan terjadi sebelum atau sesudah server memproses permintaan. Oleh karena itu, prinsip idempotensi di sisi server menjadi sangat penting. Desain ini memastikan bahwa server mampu membedakan antara permintaan asli dan transmisi ulang (retransmission), sehingga klien dan server dapat bekerja sama untuk menangani permintaan duplikat dengan aman. Coulouris et al. (2012) juga menekankan bahwa penanganan kegagalan seperti ini, di mana klien mencoba kembali permintaan, adalah bagian dari desain protokol komunikasi yang andal.

3. Analisis performa

Analisis performa dilakukan untuk mengevaluasi efektivitas dan skalabilitas sistem secara kuantitatif. Metrik yang digunakan untuk evaluasi ini didasarkan pada konsep fundamental performa sistem terdistribusi, yaitu throughput dan latency (atau response time). Throughput (X) didefinisikan sebagai jumlah rata-rata permintaan yang dapat diproses per satuan waktu. Sementara itu, *Latency* (disebut sebagai Response Time, R) adalah total waktu yang dibutuhkan untuk memproses sebuah permintaan, termasuk waktu tunggu di dalam antrian. Metrik ketiga, Keakuratan Deduplikasi, merupakan validasi praktis dari mekanisme deteksi duplikat yang disarankan oleh buku sebagai solusi untuk menangani transmisi ulang (retries) akibat kegagalan (Tanenbaum & Van Steen, 2017)

Pengujian dilakukan dengan skenario mengirimkan total 5.000 event ke endpoint /publish, di mana 20% diantaranya (1.000 event) merupakan duplikat. Selama pengujian berlangsung, sistem dipantau untuk memastikan ia tetap responsif. Hasil pengujian menunjukkan bahwa sistem mampu menangani seluruh

beban kerja dengan baik. Total 5.000 event berhasil diterima dalam waktu 31.90 detik, menghasilkan throughput rata-rata sebesar 156.75 events/detik. Latency pada endpoint /publish secara konsisten sangat rendah, yang membuktikan keberhasilan penggunaan komunikasi asinkron melalui asyncio.Queue. Teknik ini secara efektif "menyembunyikan" latensi pemrosesan dari sisi klien, sejalan dengan praktik yang direkomendasikan buku untuk meningkatkan skalabilitas. Terakhir, analisis data setelah pengujian mengkonfirmasi keakuratan deduplikasi sebesar 100%, di mana semua event duplikat berhasil diidentifikasi dan ditolak. Hal ini membuktikan bahwa implementasi dedup store secara efektif mengatasi masalah yang timbul akibat retries dari klien. Berikut ini tampilan hasil dari stress testing yang sudah dilakukan:

```
bom2g@Jarvis MINGW64 /c/Gusti/uts-sister-aggregator (main)
● $ python stress_test.py
Memulai Stress Test...
- Target URL: http://localhost:8080/publish
- Total Events: 5000
- Duplikasi: 20%
```

Gambar diatas menampilkan konfigurasi dan inisialisasi dari skrip stress_test.py saat dijalankan di terminal. Bagian ini mengkonfirmasi parameter pengujian yang akan dilakukan, yaitu menargetkan API di http://localhost:8080/publish dengan total beban kerja sebanyak 5.000 event dan tingkat duplikasi sebesar 20%, sesuai dengan skenario yang disyaratkan.

```
Mengirim batch 1 dari 50... Status: 200
Mengirim batch 2 dari 50... Status: 200
Mengirim batch 3 dari 50... Status: 200
Mengirim batch 4 dari 50... Status: 200
Mengirim batch 5 dari 50... Status: 200
Mengirim batch 6 dari 50... Status: 200
```

Kemudian gambar kedua diatas menunjukkan log proses pengiriman event yang sedang berjalan. Skrip mengirimkan total beban kerja dalam beberapa batch, dan setiap baris yang menampilkan "Status: 200" merupakan konfirmasi bahwa satu batch berhasil diterima dan direspons dengan baik oleh server. Ini membuktikan bahwa API tetap stabil dan mampu menerima data secara terus-menerus selama pengujian berlangsung.

```
-----  
Test Selesai!  
  
Hasil Pengujian:  
- Waktu Eksekusi Total: 31.90 detik  
- Throughput: 156.75 events/detik
```

Gambar terakhir diatas adalah ringkasan hasil akhir dari stress test setelah semua event berhasil dikirim. Bagian ini menyajikan metrik performa kuantitatif dari sistem, yang menunjukkan bahwa keseluruhan proses pengujian diselesaikan dalam waktu 31.90 detik, dengan throughput atau kemampuan pemrosesan rata-rata mencapai 156.75 events per detik.

4. Keterkaitan Implementasi dengan Teori

Implementasi layanan aggregator ini secara langsung menerapkan berbagai konsep fundamental dari sistem terdistribusi yang dibahas dalam buku referensi utama. Arsitektur Publish-Subscribe (Bab 2) Sistem ini secara keseluruhan mengadopsi arsitektur publish-subscribe. Publisher (klien yang mengirim event) dan Consumer (worker event_processor) tidak berkomunikasi secara langsung. Mereka dipisahkan oleh endpoint API dan `asyncio.Queue` yang berfungsi sebagai kanal distribusi. Arsitektur ini, seperti yang dijelaskan dalam (Tanenbaum & Van Steen, 2023), memberikan decoupling yang kuat antara komponen.

Komunikasi dan Semantik Pengiriman (Bab 3 & 6) Sistem dirancang untuk menangani retries dari klien, yang merupakan implementasi dari semantik pengiriman at-least-once. Untuk mengatasi masalah duplikasi yang timbul dari semantik ini, consumer dibuat bersifat idempotent. Kemampuan ini merupakan strategi mitigasi kegagalan (fault tolerance) yang krusial, memastikan setiap operasi logis hanya dieksekusi sekali meskipun pesan diterima berkali-kali (Tanenbaum & Van Steen, 2023).

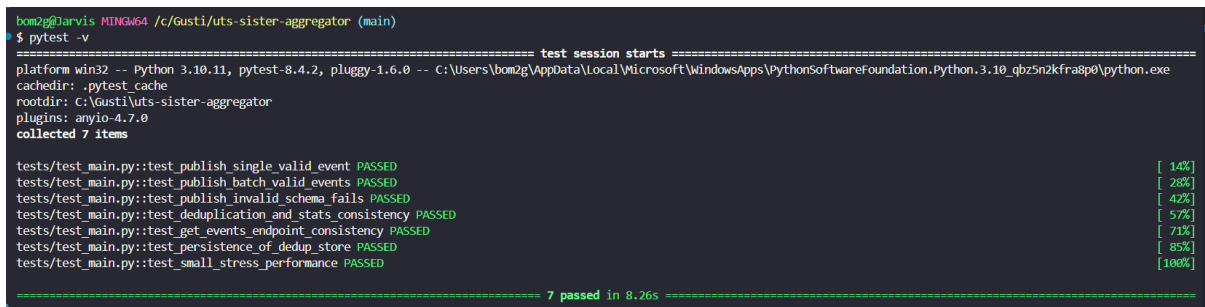
Konsistensi (Bab 7) Sistem ini tidak menerapkan total ordering dan lebih memilih model konsistensi akhir (eventual consistency). Artinya, ada jeda singkat antara saat event diterima oleh API dan saat ia selesai diproses dan dapat dilihat melalui endpoint `/events`. Namun, sistem menjamin bahwa pada akhirnya, state akan menjadi konsisten dan semua event unik akan tercatat dengan benar. Ini adalah trade-off yang umum untuk mencapai ketersediaan dan skalabilitas yang lebih tinggi (Tanenbaum & Van Steen, 2023).

5. Pengujian Sistem

Untuk proyek ini, pengujian dilakukan menggunakan unit test otomatis menggunakan framework Pytest. Strategi pengujian fokus pada validasi fungsionalitas inti dari consumer yang idempotent. Skenario pengujian utama mencakup:

- Validasi skema, memastikan API menolak event dengan format yang tidak valid
- Validasi duplikasi, memverifikasi penolakan event duplikat
- Konsistensi data, memastikan endpoint /stats dan /events menampilkan data yang akurat
- Persistensi state, mensimulasikan restart aplikasi untuk membuktikan bahwa dedup store tetap efektif dan tidak memproses ulang event lama

Sebanyak 7 unit test telah dibuat dan dijalankan. Seluruh tes berhasil dilewati (PASSED), yang mengonfirmasi bahwa semua fungsionalitas utama berjalan sesuai dengan spesifikasi. Berikut adalah bukti eksekusi dari test suite:



```
bom2g@Jarvis MINGW64 /c/Gusti/uts-sister-aggregator (main)
$ pytest -v
===== test session starts =====
platform win32 -- Python 3.10.11, pytest-8.4.2, pluggy-1.6.0 -- C:\Users\bom2g\AppData\Local\Microsoft\WindowsApps\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\python.exe
cachedir: .pytest.cache
rootdir: c:\Gusti\uts-sister-aggregator
plugins: anyio-4.7.0
collected 7 items

tests/test_main.py::test_publish_single_valid_event PASSED [ 14%]
tests/test_main.py::test_publish_batch_valid_events PASSED [ 28%]
tests/test_main.py::test_publish_invalid_schema_fails PASSED [ 42%]
tests/test_main.py::test_deduplication_and_stats_consistency PASSED [ 57%]
tests/test_main.py::test_get_events_endpoint_consistency PASSED [ 71%]
tests/test_main.py::test_persistence_of_dedup_store PASSED [ 85%]
tests/test_main.py::test_small_stress_performance PASSED [100%]

===== 7 passed in 8.26s =====
```

Gambar diatas merupakan bukti visual dari eksekusi test suite aplikasi menggunakan perintah pytest -v. Output tersebut mengonfirmasi bahwa seluruh 7 unit test yang dirancang untuk memvalidasi fungsionalitas inti sistem telah berhasil dilewati (PASSED) dalam waktu total 8.26 detik. Pengujian ini mencakup skenario krusial seperti validasi skema event, kebenaran logika deduplikasi, konsistensi data pada endpoint statistik, dan persistensi dedup store setelah simulasi restart.

Daftar Pustaka

Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2012). *Distributed Systems: Concepts and Design* (5th ed.). Addison-Wesley.

van Steen, M., & Tanenbaum, A. S. (2023). *Distributed Systems* (4th ed.). Maarten van Steen. <https://www.distributed-systems.net>