Nama : Arief Reno Fathurrahman
NIM : 11201014

Bagian Teori

Jawab ringkas (150–250 kata/poin) dan sertakan sitasi APA 7th. Soroti Bab 8–9 (transaksi/konkurensi) disertai contoh dari rancangan Anda.

T1 (Bab 1): Karakteristik sistem terdistribusi dan trade-off desain Pub-Sub aggregator.

= Sistem terdistribusi adalah sistem di mana komponen-komponen yang terletak di komputer yang terhubung dalam jaringan berkomunikasi dan mengoordinasikan tindakan mereka hanya dengan mengirim pesan. Definisi ini menunjukkan beberapa karakteristik penting dari sistem terdistribusi, antara lain: adanya konkurensi antar komponen, tidak adanya jam global, dan kemungkinan terjadinya kegagalan secara independen pada setiap komponen.Sistem terdistribusi mencakup berbagai perkembangan teknologi signifikan dalam beberapa tahun terakhir, sehingga pemahaman tentang teknologi yang mendasarinya menjadi sangat penting dalam dunia komputasi modern.Selain itu, sistem ini mencakup berbagai aplikasi yang bervariasi, mulai dari sistem yang lebih lokal, seperti yang ditemukan di mobil atau pesawat, hingga sistem skala global yang melibatkan jutaan node. Terdapat juga layanan yang terfokus pada data hingga tugas yang membutuhkan daya proses tinggi, serta sistem yang dibangun dari sensor kecil yang relatif sederhana hingga sistem yang memanfaatkan elemen komputasi yang kuat. Bahkan, sistem ini dapat mencakup sistem tertanam hingga yang mendukung pengalaman interaktif yang kompleks (Baldoni et al., 2005).

T2 (Bab 2): Kapan memilih arsitektur publish–subscribe dibanding client–server? Alasan teknis.

= Pemilihan arsitektur publish-subscribe dibandingkan client-server tergantung pada kebutuhan sistem aplikasi yang akan dibangun. Arsitektur publish-subscribe lebih tepat digunakan dalam situasi di mana banyak produser (penerbit) ingin menyebarkan informasi (peristiwa) kepada banyak konsumen (abonemen) secara efisien. Ini karena publish-subscribe memungkinkan komunikasi yang tidak terikat, di mana produser tidak perlu mengetahui siapa yang menerima informasi, sehingga mengurangi kompleksitas pengelolaan koneksi langsung antara setiap produser dan konsumen. Secara teknis, arsitektur ini memberikan fleksibilitas dengan memisahkan penghasil dan konsumen informasi, memungkinkan mereka untuk beroperasi secara independen. Selain itu, sistem ini mendukung skalabilitas, karena pertumbuhan jumlah produser atau konsumen tidak memerlukan pengaturan ulang yang signifikan. Misalnya, dalam aplikasi perdagangan keuangan, informasi harus disebarluaskan secara real-time kepada banyak peserta pasar tanpa latency yang tinggi. Dalam situasi yang memerlukan pembaruan informasi secara terus-menerus dan sinkronisasi dengan cepat, publish-subscribe memberikan keunggulan dibandingkan arsitektur client-server yang lebih terstruktur dan kaku, seperti dalam kasus pencarian informasi di mesin pencari web yang melibatkan banyak crawler yang beroperasi fleksibel (Shoch and Hupp, 1982).

T3 (Bab 3): At-least-once vs exactly-once delivery; peran idempotent consumer.

= At-least-once delivery adalah mekanisme pengiriman pesan yang menjamin bahwa pesan akan sampai ke penerima setidaknya satu kali. Namun, pendekatan ini memiliki risiko bahwa pesan yang sama dapat diterima lebih dari sekali, menyebabkan terjadinya duplikasi. Sebaliknya, exactly-once delivery berusaha memastikan bahwa pesan yang dikirim hanya diproses satu kali tanpa mengalami duplikasi, tetapi implementasinya lebih kompleks dan memerlukan kontrol yang lebih ketat terhadap pengiriman dan penerimaan pesan. Dalam konteks ini, peran idempotent consumer menjadi sangat penting. Konsumen idempotent memiliki kemampuan untuk memproses pesan yang sama berulang kali tanpa mengubah hasil akhir, sehingga jika pesan terkirim ulang, konsumen dapat mengolah pesan duplikatif tersebut tanpa efek samping yang merugikan. Dengan demikian, konsumen idempotent menjamin konsistensi dan integritas data, sehingga menjadi solusi efektif untuk mengatasi tantangan yang muncul dari mekanisme pengiriman at-least-once.

T4 (Bab 4): Skema penamaan topic dan event_id (unik, collision-resistant) untuk dedup.

= Skema penamaan topik dan event_id berfungsi untuk memastikan keunikan dan mencegah tabrakan (collision) dalam sistem pengelolaan data serta deduplikasi. Topik biasanya merepresentasikan kategori atau jenis data yang disampaikan, dan harus diberi nama secara konsisten agar mudah dikenali. Penggunaan konvensi penamaan, seperti menggunakan awalan yang menunjukkan konteks (misalnya, user_signup), dapat meningkatkan pemahaman. Sementara itu, event_id adalah identifier unik yang diberikan untuk masing-masing peristiwa (event) dalam sistem. Keunikan event_id dapat dicapai melalui penggunaan timestamp, UUID (Universally Unique Identifier), atau kombinasi elemen lainnya yang memastikan tidak ada dua peristiwa yang memiliki идентификатор sama. Dengan menerapkan skema ini, pengelolaan dan penyimpanan data menjadi lebih efisien, serta mengurangi risiko terjadinya duplikasi data yang berpotensi mengganggu analisis dan pengambilan keputusan.

T5 (Bab 5): Ordering praktis (timestamp + monotonic counter); batasan dan dampaknya.

= Ordering praktis mengacu pada penggunaan timestamp yang dikombinasikan dengan pencacah monotonik (monotonic counter) untuk memastikan urutan peristiwa dalam sistem distributed. Timestamp merepresentasikan waktu ketika peristiwa terjadi, sementara pencacah monotonik berfungsi untuk menyelesaikan konflik saat beberapa peristiwa terjadi dalam waktu yang sama. Batasan dari skema ini meliputi ketergantungan terhadap presisi waktu sistem dan potensi untuk menyimpan status saat terjadi kegagalan. Jika timestamp yang digunakan tidak cukup presisi, akan ada kemungkinan terjadinya ambiguitas dalam urutan peristiwa. Dampak dari penggunaan metode ini adalah meningkatnya keandalan pengurutan peristiwa, namun dapat menyebabkan overhead dalam pengelolaan dan penyimpanan data. Selain itu, dalam situasi dengan latensi tinggi, pengaturan ulang timestamp dapat membebani sistem secara signifikan. Oleh karena itu, pemilihan metode ini harus mempertimbangkan kebutuhan spesifik sistem yang bersangkutan (Parrington et al., 1995).

T6 (Bab 6): Failure modes dan mitigasi (retry, backoff, durable dedup store, crash recovery).

= Failure modes merujuk pada berbagai bentuk kegagalan yang dapat terjadi dalam sistem atau aplikasi, mengakibatkan performa yang tidak optimal atau bahkan kerusakan. Untuk menangani failure modes, beberapa strategi mitigasi dapat diterapkan. Salah satunya adalah metode retry, yang melibatkan pengulangan operasi yang gagal untuk mencapai keberhasilan. Namun, perlu diwaspadai agar tidak menyebabkan peningkatan beban pada sistem. Selain itu, ada teknik backoff, di mana sistem menunggu periode tertentu sebelum mencoba kembali, sehingga mengurangi tekanan dan memberi waktu untuk pemulihan. Strategi lain yang penting adalah penggunaan durable dedup store, yang memastikan bahwa setiap permintaan unik hanya diproses sekali, mencegah pengulangan akibat kegagalan. Terakhir, crash recovery mencakup pemulihan sistem dari keadaan crash, sehingga integritas data dan kelanjutan operasional tetap terjaga, biasanya melalui pencadangan data dan log transaksi. Dengan menerapkan metode-metode ini, risiko dan dampak dari failure modes dapat diminimalisir (Baldoni et al., 2005).

T7 (Bab 7): Eventual consistency pada aggregator; peran idempotency + dedup.

= Eventual consistency adalah model konsistensi di mana sistem menjamin bahwa, setelah beberapa waktu, semua salinan data akan mencapai keadaan yang konsisten meskipun pada saat tertentu, salinan tersebut mungkin tidak sama. Ini sangat relevan dalam sistem aggregator, di mana data dari berbagai sumber digabungkan. Di sini, perubahan pada satu sumber mungkin tidak langsung terlihat oleh sumber lainnya, tetapi sistem akan berkomitmen untuk menstabilkan keadaan data pada akhirnya. Dua konsep penting yang mendukung model ini adalah idempotency dan deduplication. Idempotency memastikan bahwa operasi yang sama dapat dilakukan berulang kali tanpa mengubah hasil, sehingga mencegah efek samping yang tidak diinginkan ketika permintaan yang sama diproses lebih dari sekali. Sementara itu, deduplication berfungsi untuk mengidentifikasi dan menghapus entri duplikat, memastikan bahwa setiap data yang disimpan hanya ada satu kali di sistem. Dengan menerapkan kedua konsep ini, sistem aggregator mampu menjaga integritas data dan meningkatkan reliability, meskipun berjalan dalam model eventual consistency (Accetta et al. 1986) (Rozier et al. 1988, 1990).

T8 (Bab 8): Desain transaksi: ACID, isolation level, dan strategi menghindari lost-update.

= Desain transaksi yang tepat sangat penting untuk menjaga integritas data dalam sistem basis data. Model ACID (Atomicity, Consistency, Isolation, Durability) menjadi prinsip dasar yang menjamin transaksi dijalankan dengan aman dan konsisten. Pertama, atomicity memastikan bahwa seluruh operasi dalam transaksi dilakukan secara utuh; jika salah satu bagian gagal, seluruh transaksi akan dibatalkan. Kedua, consistency menjamin bahwa transaksi membawa data dari satu keadaan yang valid ke keadaan lain, sehingga aturan dan constraint yang telah ditetapkan tetap terjaga. Ketiga, isolation mengatur sejauh mana transaksi dapat terpisah dari transaksi lain, mencegah pengaruh yang tidak diinginkan di antara transaksi yang berjalan secara bersamaan. Selain itu, untuk menghindari fenomena lost update, strategi seperti penggunaan tingkat isolasi yang lebih tinggi—seperti Serializable—dapat diterapkan, di mana transaksi satu tidak dapat melihat perubahan yang dilakukan oleh transaksi lain hingga proses tersebut selesai. Ini membantu menjaga integritas data dan mencegah kehilangan informasi yang penting ([Bruneton et al. 2006).

T9 (Bab 9): Kontrol konkurensi: locking/unique constraints/upsert; idempotent write pattern.

= Kontrol konkurensi adalah mekanisme yang digunakan dalam basis data untuk memastikan bahwa transaksi yang dilakukan secara bersamaan tidak mengganggu satu sama lain, sehingga integritas data tetap terjaga. Salah satu metode adalah locking, di mana sistem mengunci data yang sedang diakses oleh suatu transaksi, sehingga transaksi lain tidak dapat mengubahnya sampai transaksi pertama selesai. Selain itu, unique constraints digunakan untuk mencegah penyisipan data yang duplikat, memastikan bahwa setiap entri dalam tabel memiliki nilai unik untuk kolom tertentu. Ini berfungsi sebagai langkah pencegahan dalam menjaga konsistensi data. Metode lain yang penting adalah upsert, yang merupakan gabungan dari operasi insert dan update. Upsert memungkinkan sistem untuk memasukkan data baru jika belum ada, atau memperbarui data yang sudah ada jika kunci uniknya ditemukan. Terakhir, idempotent write pattern adalah strategi di mana penulisan data yang sama dapat dilakukan berkali-kali tanpa mengubah hasil akhir. Dengan cara ini, meskipun permintaan yang sama diproses beberapa kali karena kegagalan atau kesalahan jaringan, integritas dan konsistensi data tetap terjamin. Kombinasi dari semua teknik ini membantu mengelola kontrol konkurensi dengan efektif (Bruneton et al. 2006).

T10 (Bab 10–13): Orkestrasi Compose, keamanan jaringan lokal, persistensi (volume), observability.

= Dalam konteks arsitektur perangkat lunak, orkestrasi Compose adalah pendekatan yang mengelola interaksi antar layanan dalam sistem yang terdistribusi. Dengan menggunakan alat orkestrasi seperti Docker Compose, pengembang dapat mengatur, menjalankan, dan mengelola beberapa kontainer secara bersamaan, yang meningkatkan efisiensi dalam pengembangan dan penyebaran aplikasi. Keamanan jaringan lokal sangat penting untuk melindungi data dan layanan yang berjalan di dalam jaringan tersebut. Ini mencakup pengaturan firewall, penggunaan enkripsi untuk data yang sedang ditransmisikan, serta penerapan kontrol akses yang ketat untuk memastikan hanya pengguna yang berwenang yang dapat mengakses sumber daya tertentu. Persistensi, atau volume, mengacu pada cara data disimpan dalam konteks kontainerisasi. Dengan menggunakan volume, data dapat disimpan secara permanen dan tidak akan hilang meskipun kontainer dihentikan atau dihapus. Ini memfasilitasi konsistensi data dan integrasi yang lebih baik antara kontainer. Observability adalah kemampuan untuk memonitor dan memahami status sistem secara real-time. Ini melibatkan pengumpulan, analisis, dan visualisasi metrik serta log dari berbagai komponen sistem. Observability memungkinkan tim pengembang dan operasi untuk mendeteksi dan menyelesaikan masalah dengan cepat, memastikan kinerja yang optimal dan mendukung pengambilan keputusan yang lebih baik dalam pengelolaan infrastruktur. Dengan menggabungkan semua elemen ini, organisasi dapat menciptakan lingkungan yang lebih aman, efisien, dan dapat diandalkan.