РЕЦЕНЗІЯ на статтю «Architectural Solutions for Implementing Real-Time Applications in the Digital Environment»

Актуальність теми. Затримка у 50–100 мс на платформі фінансових транзакцій — це не технічна деталь, це бізнес-ризик. І саме вибір архітектурного рішення визначає, чи залишиться система стабільною під час пікового навантаження, чи почне деградувати в найвідповідальніший момент. Стаття розглядає саме ці виклики: архітектурні підходи до побудови застосунків реального часу в розподілених системах. Предмет дослідження має очевидну практичну цінність. Системи на кшталт стрімінгових платформ, IoT-інфраструктур або мобільних банківських сервісів щосекунди обробляють тисячі подій — і архітектурна помилка на рівні проектування відразу дають про себе знати в production. Для провідних інженерів, що проектують подібні системи, тема статті безпосередньо корисна. Автор не обмежується одним підходом, а системно розглядає кілька конкурентних архітектурних стилів — що дозволяє читачеві сформувати цілісне уявлення про доступний інструментарій.

Технічний зміст. Автор розглядає п'ять основних архітектурних підходів: мікросервісну архітектуру, event-driven системи, стрімінгові платформи, реактивні моделі та edge computing. Охоплення широке, і для статті оглядового характеру — виправдане. Одна з сильних сторін — чітке різниця між hard real-time і soft real-time системами. Це розмежування не очевидним для читачів без вузькоспеціалізованого досвіду: система керування гальмами автомобіля є прикладом hard RT, де запізнення відповіді навіть на кілька мілісекунд є неприпустимим; стрімінг аудіо — soft RT, де буферизація компенсує мережеві флуктуації. Такий поділ правильно задає напрямок для

подальшого обговорення. Розділ про мікросервіси охоплює ключові переваги — незалежне масштабування, ізоляцію компонентів — і коректно вказує на складності: мережеву затримку, моніторинг, управління консистентністю. Варто зазначити, що внутрішній in-process виклик між модулями займає близько 0.01–0.1 мс, тоді як HTTP-виклик між сервісами в тій самій мережі — вже 1–5 мс. Ця різниця на перший погляд незначна, але в ланцюжку з десяти послідовних викликів вона стає відчутною. Автор цю проблему позначає, і це правильно. Розгляд event-driven архітектури — технічно грамотний. Асинхронна комунікація через черги повідомлень справді дозволяє розв'язати компоненти системи і обробляти великі обсяги подій без блокування. При цьому варто розуміти і зворотну сторону: EDA ускладнює дебагінг (відсутній чіткий control flow), вимагає уваги до ordering гарантій і ідемпотентності обробників. В умовах транзакційних систем це не дрібниці — це архітектурні рішення, від яких залежить коректність бізнес-логіки. Реактивні підходи та actor-based моделі розглянуті коректно. Їхня перевага — збереження responsiveness під непередбачуваним трафіком — є актуальною для мобільних і IoT-сценаріїв, де клієнти можуть з'являтися і зникати непередбачувано. Edge computing поданий як засіб зниження затримки через наближення обчислень до джерела даних. Це справедливо — і особливо актуально для сценаріїв, де центральний дата-центр фізично далеко від кінцевих пристроїв. Автор коректно обмежує сферу застосування цього підходу розподіленими середовищами. Загалом технічний матеріал викладено грамотно і структуровано, хоча деяким аспектам бракує глибини.

Коментарі та пропозиції. Зауваження стосуються конкретних прогалин, які, на мій погляд, обмежують практичну цінність роботи. Кількісні орієнтири відсутні там, де вони найбільш потрібні. Стаття оперує такими поняттями, як «зниження затримки» або «покращення чутливості системи»,

але жодного разу не наводить конкретних цифр. Для читача-практика принципово важливо розуміти порядок величин: наскільки edge computing може скоротити round-trip latency в порівнянні з централізованою обробкою. Вираз «великий обсяг подій» у контексті EDA — не доцільний. Навіть приблизні цифрові орієнтири суттєво підвищили б практичну цінність матеріалу.

Моніторинг та спостережуваність без конкретики. Автор слушно акцентує увагу на складності розподілених систем, але не згадує інструментарій, що став de facto стандартом: Jaeger або Zipkin для distributed tracing, Prometheus разом з Grafana для агрегації метрик, спеціалізований моніторинг черг у Kafka або RabbitMQ. Без цього розділ залишається на рівні загальних рекомендацій, а не практичного керівництва.

Консистентність даних — тема ширша, ніж позначено**.** При обговоренні мікросервісів і EDA бажано окремо зупинитися на моделях eventual consistency та стратегіях ідемпотентної обробки повідомлень. У системах, що обробляють транзакційні дані (платежі, резервування, оновлення інвентаризації), ці аспекти визначають коректність бізнес-логіки. Їх не варто залишати поза увагою.

Варто згадати тестування real-time систем. Хороша архітектура — це не тільки те, що добре виглядає на діаграмі, але й те, що витримує production. Chaos engineering (ін'єкція збоїв для перевірки стійкості), load testing під реальними патернами трафіку, контрактне тестування між сервісами — є невід'ємною частиною роботи з distributed real-time системами. Їх відсутність у статті помітна.

Висновок. Стаття закриває реальну потребу: якісного, доступного огляду архітектурних підходів до real-time систем. Матеріал логічно

структурований, термінологія коректна, а розмежування між hard і soft real-time — одна з найчіткіших і найкорисніших частин роботи.

Стаття може бути рекомендована до публікації за умови доопрацювання деяких аспектів.

Рецензент:

Люшенко Андрій Миколайович

провідний інженер-дослідник,

архітектор мобільних систем,

експерт у проектуванні масштабованих мобільних та розподілених систем,

магістр прикладної математики

(Одеса, Україна)