УДК 004.75:336.71

ОБЕСПЕЧЕНИЕ ОТКАЗОУСТОЙЧИВОСТИ МИКРОСЕРВИСНЫХ БАНКОВСКИХ СИСТЕМ

Жұмаберген Алтынбек Асқарұлы
altinbekjyma@gmail.com
магистрант 1 курса Программной инженерии
Атырауский университет имени Халела Досмұхамедова, г. Атырау, Казахстан
Научный руководитель – кандидат технических наук, профессор Курмангазиева Л.Т.

Аннотация

Микросервисные архитектуры всё чаще применяются в финансовых организациях для повышения модульности, автономности команд и ускорения процессов разработки и внедрения. Однако их использование также приводит к увеличению количества сетевых зависимостей и появлению новых типов отказов — таких как частичные сбои, рост задержек, «штормы» повторных запросов и нарушения согласованности межсервисных бизнес-транзакций. В банковской сфере эти технические риски тесно связаны с требованиями операционной устойчивости, включая способность поддерживать выполнение критически важных операций в условиях сбоев и соблюдать допустимые уровни воздействия и времени простоя даже при серьёзных, но реалистичных сценариях.

Ключевые слова: банковские системы; микросервисы; отказоустойчивость; операционная устойчивость; RTO (Recovery Time Objective — целевое время восстановления), RPO (Recovery Point Objective — допустимая точка восстановления данных); circuit breaker; at-least-once

Введение

Банковские услуги, такие как платежи, переводы, обслуживание клиентов и цифровые каналы, как правило, классифицируются как «критически важные операции» или «ключевые бизнес-сервисы», длительное нарушение работы которых может привести к ущербу для клиентов и системным последствиям. Международные надзорные рекомендации трактуют операционную устойчивость как способность поддерживать выполнение критически важных функций в условиях сбоев, а также эффективно реагировать, восстанавливаться, адаптироваться и извлекать уроки из дестабилизирующих событий. В ряде юрисдикций введены специальные нормативные требования, направленные на обеспечение устойчивости финансовых организаций. Например, в Европейском союзе принят регламент Digital Operational Resilience Act (DORA), вступающий в силу с 17 января 2025 года. Данный норматив усиливает требования к управлению ИКТ-рисками, реагированию на инциденты и обеспечению устойчивости финансовых организаций, а также их критически важных ИКТ-зависимостей. В Соединённом Королевстве надзорные органы требуют от организаций устанавливать и тестировать допустимые уровни воздействия (impact tolerances) для ключевых бизнес-сервисов, а также подтверждать способность функционировать в пределах этих показателей при серьёзных, но реалистичных сценариях. Несмотря на то, что понятие «операционная устойчивость» выходит за рамки исключительно технической надёжности, отказоустойчивость микросервисных систем является её фундаментальной составляющей: если сбой одного сервиса способен вызвать отказ всей платформы, выполнение требований по допустимым уровням воздействия и целям восстановления становится затруднительным. Инженерные практики подчёркивают, что распределённые системы должны проектироваться с учётом вероятности временных сбоев, таймаутов и различий во времени восстановления; такие системы должны обеспечивать быстрое завершение неуспешных операций (fail fast) и изоляцию отказов для предотвращения истощения ресурсов и каскадных сбоев.

В данной работе основное внимание уделяется обеспечению отказоустойчивости микросервисных банковских систем. Предлагается практико-ориентированный подход, основанный на результатах исследований, который связывает технические архитектурные паттерны с целями операционной устойчивости и требованиями надзорных органов.

Материалы и методы

Материалы исследования включали нормативные документы и официальные разъяснения по операционной устойчивости (Европейский союз — DORA; международные принципы — BCBS; Соединённое Королевство — PRA/FCA), а также инженерные руководства по устойчивости распределённых систем, публикации (книги) по паттернам микросервисной архитектуры и практики, связанные с наблюдаемостью и надёжностью.

Метод исследования включает три этапа:

1. Анализ регуляторных требований к «устойчивости критических операций» и метрик восстановления (включая RTO/RPO и максимально допустимое время простоя), применимых в финансовом секторе.

2. Выделение ключевых режимов отказов и деградации, характерных для распределённых микросервисных систем, и сопоставление им инженерных паттернов (circuit breaker, retry-стратегии, bulkhead, асинхронные взаимодействия).

3. Формирование референсного подхода и плана проверки устойчивости через метрики (SLO/error budget), телеметрию и сценарное тестирование, включая принципы chaos engineering.

Операционная устойчивость и цели восстановления в финансовом секторе. Международные органы банковского надзора определяют операционную устойчивость как способность продолжать выполнение критически важных операций в условиях сбоев и вводят понятие допустимого уровня нарушения (tolerance for disruption), отражающего приемлемую продолжительность и степень воздействия. Принципы Базельского комитета по банковскому надзору также подчёркивают важность таких компонентов, как корпоративное управление, планирование и тестирование непрерывности бизнеса, выявление взаимосвязей и зависимостей, управление рисками, связанными с третьими сторонами, а также управление инцидентами. Аналогичным образом европейское банковское регулирование рассматривает операционную устойчивость как обеспечение непрерывного выполнения критически важных операций в условиях сбоев, трактуя её как развитие и усиление механизмов управления операционными рисками, управления, аутсорсинга и обеспечения непрерывности бизнеса. С точки зрения инженерного обеспечения непрерывности и восстановления в финансовой отрасли широко применяются измеримые показатели восстановления: RTO (Recovery Time Objective — целевое время восстановления), RPO (Recovery Point Objective — допустимая точка восстановления данных), а также максимально допустимое время простоя. Например, рекомендации FFIEC по обеспечению непрерывности бизнеса прямо указывают на использование RPO и RTO в качестве стандартных метрик для определения целей восстановления после оценки воздействия сбоев. Данные показатели выступают связующим звеном между нормативной терминологией (такими как «допустимые уровни воздействия» или «tolerance for disruption») и техническими решениями, включая репликацию, механизмы аварийного переключения (failover), устойчивость хранения данных и операционные процедуры.

Почему микросервисы усиливают режимы отказов. Микросервисная архитектура предполагает разбиение приложения на набор независимо развертываемых сервисов, взаимодействующих друг с другом по сети. Это повышает модульность системы, однако распределённые вызовы приводят к возникновению частичных отказов (когда один зависимый сервис выходит из строя, а остальные продолжают функционировать), непредсказуемых задержек и проблем с обратным давлением (backpressure). В связи с этим такие паттерны, как автоматические выключатели (circuit breaker), таймауты и изоляция ресурсов (bulkhead), рассматриваются как обязательные элементы обеспечения «готовности к промышленной эксплуатации» (production readiness) при взаимодействии

сервисов. Одной из ключевых угроз в распределённых системах являются «штормы повторных запросов» (retry storms) — несогласованные или чрезмерно агрессивные попытки повторного выполнения операций, которые увеличивают нагрузку на систему и усугубляют сбои. Инженерные практики рекомендуют ограничивать количество повторных попыток, использовать экспоненциальную задержку (exponential backoff), учитывать рекомендации по повторным запросам (retry hints), а также комбинировать механизмы повторных попыток с паттернами circuit breaker и средствами мониторинга (telemetry).

Надёжный обмен сообщениями, согласованность состояния и идемпотентность. Банковские процессы часто охватывают несколько ограниченных контекстов (bounded contexts), таких как инициирование платежа, обновление бухгалтерской книги (ledger), доставка уведомлений и проведение проверок рисков. В микросервисной архитектуре единая бизнес-транзакция, затрагивающая несколько сервисов, не может опираться на единую ACID-транзакцию для всех компонентов. Вместо этого применяются такие паттерны, как saga, позволяющие организовать последовательность локальных транзакций с использованием событий или сообщений, а также компенсирующих действий. Сага, как правило, представляет собой цепочку локальных транзакций: каждая из них обновляет собственное хранилище данных и публикует сообщение или событие для запуска следующего этапа. В случае сбоя на последующих этапах выполняются компенсирующие операции. Одним из распространённых механизмов возникновения сбоев в реальных системах является несогласованность данных, вызванная так называемой проблемой «двойной записи» (dual-write): сервис сначала обновляет базу данных, а затем публикует событие; если публикация не выполняется после фиксации транзакции (или фиксация не происходит после публикации), система переходит в неконсистентное состояние. Стандартным способом устранения данной проблемы является паттерн transactional outbox: исходящее событие сохраняется в специальной таблице или коллекции (outbox) в рамках той же локальной транзакции, что и изменение бизнес-состояния, после чего события асинхронно передаются брокеру сообщений. Брокеры сообщений, как правило, обеспечивают доставку по принципу «как минимум один раз» (at-least-once) при использовании подтверждений (acknowledgements), что подразумевает возможность появления дубликатов сообщений. В связи с этим потребители должны быть идемпотентными, то есть способными безопасно обрабатывать одно и то же сообщение несколько раз — что особенно важно для финансовых транзакций и клиентских уведомлений. Официальная документация RabbitMQ указывает, что подтверждения обеспечивают доставку по принципу at-least-once, тогда как их отсутствие может привести к потере сообщений и семантике «не более одного раза» (at-most-once). Аналогично, документация Apache Kafka описывает семантику at-least-once в случаях повторной отправки сообщений без идемпотентности со стороны продюсера, а также подчёркивает использование идемпотентных продюсеров как механизма повышения гарантий доставки.

Референсная архитектура для домена обработки операций и уведомлений в банке. Референсная архитектура ориентирована на типичный сегмент цифрового банкинга: обработку операции (например, инициирование и выполнение платежа) и последующее уведомление клиента. В её основе лежит событийно-ориентированное взаимодействие (event-driven), позволяющее избежать длинных синхронных цепочек зависимостей, при этом архитектура изначально спроектирована с учётом возможности частичных сбоев.

Ключевые компоненты. API-фасад (канальный уровень) принимает входящие запросы и выполняет их базовую валидацию. Длительные или ресурсоёмкие операции, по возможности, переносятся в асинхронную обработку для снижения связности компонентов системы. Архитектурные подходы облачных систем рекомендуют использовать паттерн «асинхронный запрос–ответ» (asynchronous request–reply) в тех случаях, когда обработка на стороне бэкенда не может быть завершена в рамках одного синхронного взаимодействия.

Микросервисы обработки, такие как Payment Service, Ledger Adapter и Limits/Risk

Service, реализуют локальные транзакции и публикуют доменные события вместо выполнения всех последующих (downstream) операций в синхронном режиме, что позволяет снизить риск возникновения каскадных отказов.

Микросервис уведомлений (notification service) потребляет события и формирует запись «намерение уведомления» (intent to notify), обеспечивая тем самым гарантированную доставку уведомлений и предотвращение их дублирования для клиента. Поскольку обмен сообщениями, как правило, реализуется по принципу «как минимум один раз» (at-least-once) и допускает повторную доставку, потребитель уведомлений проектируется как идемпотентный — за счёт использования уникальных идентификаторов сообщений и механизма дедупликации.

Брокер сообщений обеспечивает буферизацию и временную декомпозицию (decoupling) компонентов системы. Доставка по принципу «как минимум один раз» (at-least-once) с использованием подтверждений со стороны потребителя означает, что в случае сбоев возможно появление дублирующих сообщений. В связи с этим требования к корректности обработки переносятся на уровень приложения и включают идемпотентное потребление, механизмы дедупликации и корректную стратегию подтверждения сообщений. Для обеспечения надёжной публикации событий используется паттерн transactional outbox. События сохраняются в той же транзакции, что и изменения бизнес-состояния, после чего асинхронно передаются брокеру сообщений, что позволяет снизить риск неконсистентности, связанной с проблемой «двойной записи» (dual-write).

Оркестрация или хореография саг (saga) применяется для управления межсервисными бизнес-процессами, такими как резервирование средств, проведение транзакций, подтверждение платежей и отправка уведомлений. Сага моделирует распределённые бизнес-транзакции как последовательность локальных операций с компенсирующими действиями, которые восстанавливают инварианты системы в случае сбоя на одном из этапов.

Обсуждение

Сопоставление результатов с нормативными требованиями показывает, что «операционная устойчивость» в контексте надзора требует не только резервирования инфраструктуры, но и способности управлять сбоями в распределённых цепочках зависимостей, которые становятся основным источником системных рисков в микросервисных архитектурах. В этом контексте инженерные паттерны (circuit breaker, retry/backoff, bulkhead) следует рассматривать не как «локальные оптимизации», а как управляемые меры. Их можно связать с бизнес-метриками (impact tolerances, RTO/RPO) и тестировать через сценарии «серьёзных, но правдоподобных» нарушений.

Важным аспектом является обеспечение корректности: даже при корректной деградации сервисов повторная доставка сообщений в системе с семантикой *at-least-once* создаёт риск дублирования финансовых операций и уведомлений. Поэтому требования к идемпотентности и механизмы контроля подтверждений должны быть неотъемлемой частью проектирования доменных обработчиков.

Наконец, компоненты наблюдаемости (observability) и политика бюджета ошибок формируют измеримую основу, позволяющую перейти от декларативного понимания устойчивости к практическому управлению ею: при исчерпании бюджета ошибок вводятся ограничения на изменения и релизы, с акцентом на устранение первопричин деградации. Такой подход соответствует практикам SRE и повышает воспроизводимость результатов тестирования устойчивости. Принципы Chaos Engineering предоставляют методологию проверки устойчивости через контролируемые эксперименты, позволяя валидировать поведение системы при сбоях на практике, а не только теоретически. Однако в банковской среде подобные эксперименты должны проводиться с чёткими ограничениями, соответствовать внутренним политикам и требованиям регуляторов по управлению рисками и минимизации ущерба.

Результаты

Основным результатом является выявление управляемой «оси выравнивания» между операционной устойчивостью на уровне бизнеса и архитектурными решениями на техническом уровне. Используемое определение операционной устойчивости опирается на формулировки международных надзорных органов (BCBS) и европейских регуляторов, подчёркивающих необходимость обеспечения выполнения критически важных операций «в условиях сбоев».

На этой основе предлагается следующая цепочка: ключевой бизнес-сервис → пользовательский сценарий → схема технических зависимостей, что соответствует требованиям по картированию и тестированию в рамках подхода impact tolerances в Великобритании, а также общему подходу к управлению ИКТ-рисками в рамках DORA.

Вторым результатом является формализация набора ключевых режимов отказов и соответствующих мер управления:

• При деградации зависимостей рекомендуется сочетание таймаутов и механизма *circuit breaker* для быстрого обнаружения сбоев и снижения нагрузки на неисправный компонент (подход *fail fast*). Данный подход подробно описан в паттерне Circuit Breaker, основная цель которого — предотвращение повторных вызовов операций, с высокой вероятностью завершающихся неудачей.

• Для временных ошибок применяются стратегии повторных попыток с паузами и управляемым увеличением интервалов (*backoff*); при этом важно контролировать количество и длительность повторов, а также использовать, например, экспоненциальный backoff для предотвращения перегрузки системы.

• Для предотвращения анти-паттерна «штормов запросов» (*retry storms*) предлагаются меры, включающие ограничение числа повторных попыток, увеличение интервалов между ними, использование *circuit breaker*, корректную обработку заголовков *retry-after*, а также применение телеметрии для выявления и остановки повторяющихся попыток.

• Для снижения риска каскадных сбоев и истощения ресурсов предлагается механизм изоляции (*bulkhead*), разделяющий систему на отдельные пулы или сегменты, что позволяет локализовать деградацию и сохранить работоспособность системы в целом.

Третьим результатом является уточнение требований к корректности при событийном взаимодействии. При использовании подтверждений со стороны потребителя брокер сообщений обеспечивает семантику *at-least-once*, что означает возможность многократной доставки сообщений и требует идемпотентной обработки, особенно в контексте финансовых транзакций и уведомлений.

Список литературы

1. European Securities and Markets Authority. Digital Operational Resilience Act (DORA): применяется с 17 января 2025 г.

2. Basel Committee on Banking Supervision. Principles for Operational Resilience (BCBS 516).

3. Bank of England, Prudential Regulation Authority. SS1/21: Operational resilience – impact tolerances for important business services.

4. Financial Conduct Authority. PS21/3: Building operational resilience: policy statement.

5. Federal Financial Institutions Examination Council (FFIEC). IT Examination Handbook: Business Continuity Management.

6. National Institute of Standards and Technology. Systems Security Engineering: Considerations for a Multidisciplinary Approach (SP 800-160, Vol. 2).

7. Fowler M. Circuit Breaker [Электронный ресурс]. – Режим доступа: https://martinfowler.com/bliki/CircuitBreaker.html (дата обращения: 03.04.2026).

8. Microsoft. Azure Architecture Center: паттерны Circuit Breaker, Retry, Bulkhead [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/azure/architecture/patterns/ (дата обращения: 03.04.2026).

9. Amazon. Timeout, retries, and backoff with jitter [Электронный ресурс] // Amazon Builders' Library. – Режим доступа: https://aws.amazon.com/builders-library/ (дата обращения: 03.04.2026).

10. Richardson C. Microservices Patterns: Saga, Transactional Outbox [Электронный ресурс]. – Режим доступа: https://microservices.io (дата обращения: 03.04.2026).

11. OpenTelemetry. Документация: трассировки, метрики и логи [Электронный ресурс]. – Режим доступа: https://opentelemetry.io (дата обращения: 03.04.2026).

12. Google. Site Reliability Engineering: SLO и error budgets [Электронный ресурс]. – Режим доступа: https://sre.google (дата обращения: 03.04.2026).