

Программирование: как измерить и снизить хвостовые задержки в микросервисах

27.10.2026 · PDF-версия статьи с сайта cms.dolnais.ru

Почему быстрые сервисы дают медленный ответ

В микросервисной архитектуре запрос часто требует данных из нескольких источников. Когда сервисы вызываются параллельно, кажется, что итоговое время равно максимальному времени ответа среди них. На практике всё сложнее: чем больше зависимостей, тем выше вероятность, что хотя бы одна ответит медленно. Это явление называют **fan-out** — ветвление запроса на множество параллельных вызовов. В 2026 году с ростом числа интеграций и ИИ-агентов, которые сами делают множественные запросы, проблема становится особенно актуальной.

Рассмотрим типичный пример — карточку товара в интернет-магазине. Чтобы её отрисовать, фронтенд обращается к сервису каталога, сервису цен, сервису рекомендаций, сервису отзывов и сервису доставки. Каждый из них может отвечать за 10–50 мс, но если хотя бы один задержится на 500 мс, весь ответ замедлится. При большом количестве запросов такие «хвосты» распределения становятся заметны в метрике **p99** — времени, за которое укладываются 99% запросов. Именно p99 часто фигурирует в SLA.

Эксперимент: как fan-out раздувает p99

В недавнем исследовании (октябрь 2026) инженеры воспроизвели эту проблему на примере карточки товара. Они собрали стенд, где каждый из пяти сервисов отвечал в среднем 20 мс с редкими выбросами до 200 мс. При последовательных вызовах итоговое время было предсказуемым, но при параллельном fan-out p99 вырос в несколько раз. Причина — вероятностный эффект: если каждый сервис имеет 1% запросов длительностью более 200 мс, то при 5 параллельных вызовах вероятность хотя бы одного такого запроса составляет уже около 5%. А для 10 сервисов — почти 10%.

Чем больше зависимостей, тем выше вероятность дождаться хотя бы одного медленного ответа. Это приводит к росту хвостовых задержек и нарушению SLA, даже если средние значения в норме.

Авторы эксперимента также отметили, что кэширование и индексы могут улучшить ситуацию, но не всегда решают проблему полностью. Например, кэш помогает только для повторяющихся запросов, а индекс ускоряет выборку из БД, но не устраняет сетевые задержки.

Стратегии борьбы с хвостовыми задержками

Чтобы уложиться в SLA, нужно управлять ожиданием. Вот несколько подходов, которые можно применять в 2026 году.

1. Ограничение времени ожидания необязательных данных

Не все данные одинаково важны для ответа. Например, для карточки товара цена и наличие критичны, а рекомендации и отзывы — нет. Можно установить для необязательных вызовов таймаут, меньший, чем общий

бюджет запроса, и при его истечении вернуть частичный ответ без этих данных. Это снизит p99, хотя и потребует продумать, как отображать отсутствие информации.

2. Асинхронная загрузка и стриминг

Вместо того чтобы ждать все данные сразу, можно отдавать ответ по частям. Сначала — основную информацию, затем — дополнительную по мере готовности. Такой подход реализуется через HTTP/2 Server Push, WebSocket или GraphQL с возможностью частичного ответа. Он позволяет улучшить воспринимаемую производительность, хотя общее время ответа может остаться высоким.

3. Деградация функциональности

Если сервис недоступен или отвечает слишком долго, можно вернуть значение по умолчанию или закешированные данные. Например, если сервис рекомендаций не ответил за 100 мс, показать популярные товары из кэша. Это требует наличия резервных данных и логики fallback.

4. Изоляция и пулы соединений

Ограничение количества одновременных вызовов к каждому сервису (bulkhead) и настройка пулов соединений помогают избежать перегрузки. Если один сервис начинает тормозить, это не должно влиять на другие. В 2026 году популярны инструменты типа Envoy, Linkerd и Istio, которые предоставляют такие возможности из коробки.

5. Мониторинг и профилирование

Чтобы понять, где именно возникают хвосты, нужно измерять задержки на каждом этапе. Используйте распределённый трейсинг (OpenTelemetry, Jaeger) и собирайте гистограммы задержек. Анализ p99 и p99.9 поможет выявить проблемные зависимости. Также полезно проводить нагрузочное тестирование с реалистичным профилем (например, с помощью k6 или Locust) и смотреть, как ведёт себя система при разных уровнях параллелизма.

Пример из практики: сокращатель ссылок

В другом эксперименте (также октябрь 2026) инженер оценил нагрузку на сокращатель ссылок по главе из книги «System Design Interview» Алекса Сюя. Ожидалось 500 запросов в секунду с p95 около 20 мс. Однако на стенде в Docker с нагрузкой через k6 удалось получить лишь 24 RPS при p95 22 секунды. Причина — не только в fan-out, но и в неоптимальных запросах к БД. Автор показал, что кэш исправляет только один профиль нагрузки (например, чтение), а индекс — оба (чтение и запись). Этот пример подчёркивает, что хвостовые задержки могут возникать не только из-за количества зависимостей, но и из-за архитектурных решений.

Что делать в 2026 году

С ростом сложности систем и распространением ИИ-агентов, которые генерируют множество параллельных запросов, проблема хвостовых задержек становится всё более острой. Вот несколько рекомендаций:

- **Определите бюджет задержки** для каждого типа запроса и распределите его между зависимостями.
- **Классифицируйте зависимости** на обязательные и необязательные, настройте таймауты и fallback.
- **Используйте асинхронные паттерны**, где это возможно, чтобы не блокировать ответ.
- **Внедряйте observability** — без измерений вы не узнаете, где узкие места.
- **Тестируйте под нагрузкой**, близкой к реальной, включая сценарии с отказами сервисов.

Помните, что p99 — это не просто цифра, а показатель того, насколько комфортно будет меньшинству пользователей. Игнорирование хвостов может привести к потере клиентов и репутации. В 2026 году инструменты и практики позволяют держать задержки под контролем, если подходить к проектированию осознанно.

Источник: <https://task.dolnais.ru/programmirovanie-kak-izmerit-i-snizit-hvostovye-zaderzhki-v-mikroservisah>