Engineer

Експоненційна затримка

⏳ пауза між спробами, що множиться після кожної невдачі

Що це таке

Експоненційна затримка (exponential backoff) — це правило, яке визначає, скільки чекати перед наступною спробою. Після кожної невдалої спроби пауза множиться на сталий коефіцієнт, найчастіше на два, доки не досягне стелі: одна секунда, потім дві, чотири, вісім і так далі. Правило живе в клієнті, але існує заради того, до чого клієнт звертається. Сервер, мережевий шлях чи база даних, що відмовляють під надмірним навантаженням, не відновляться швидше, якщо їх одразу ж запитати знову. Вони відновлюються, коли ті, хто до них звертається, послаблюють тиск, а пауза, що зростає з кожною невдачею, змушує їх послаблювати його тим сильніше, чим довше триває проблема.

Найстаріший і найпоширеніший приклад — у TCP. RFC 6298, документ категорії Standards Track від червня 2011 року, що визначає, як TCP обчислює тайм-аут повторної передачі, встановлює цей тайм-аут в одну секунду, доки не виміряно перший час проходження даних туди й назад (round-trip time), і каже, що обчислене значення, менше за одну секунду, слід округлити вгору. Щоразу, коли таймер спрацьовує і сегмент доводиться надсилати знову, хост мусить подвоїти тайм-аут. Верхню межу дозволено, якщо вона становить щонайменше 60 секунд. Коли надходить свіже вимірювання часу проходження даних туди й назад, тайм-аут знову обчислюється з вимірювань, і це може одразу повернути його вниз після серії подвоєнь. Алгоритм походить зі статті Ван Джейкобсона Congestion Avoidance and Control 1988 року, а розділ RFC про безпеку пояснює, що поставлено на карту: інтернет покладається на правильний таймер повторної передачі, щоб мережа залишалася стабільною і не сталося перевантажувального колапсу (congestion collapse).

У прикладному коді та сама ідея має кілька налаштувань: початкову паузу, множник, максимальну паузу, на якій зростання зупиняється, і ліміт на кількість спроб або на загальний час, витрачений на повтори. Якщо записати це формулою, пауза після n-ї невдачі дорівнює меншому з двох значень: стелі та базової паузи, помноженої на множник у степені n, де n рахують від нуля. Аналіз AWS, про який ідеться нижче, називає це capped exponential backoff, тобто експоненційною затримкою зі стелею. Стеля важить більше, ніж здається. Якщо подвоїти секундну паузу десять разів, вийде 1024 секунди, понад 17 хвилин, а це довше, ніж більшість готова чекати будь-чого.

Останнє налаштування — джитер (jitter), випадкова складова кожної паузи. Протокол connection backoff у gRPC, що описує, як клієнти gRPC перепідключаються після невдалого з'єднання, починає з однієї секунди, множить на 1,6, припиняє зростання на 120 секундах і застосовує джитер 0,2, тож кожна пауза випадково зсувається вгору чи вниз щонайбільше на 20%. Він також дає кожній спробі з'єднання щонайменше 20 секунд на завершення і скидає backoff до початкового значення, щойно сервер прийняв з'єднання, а моментом прийняття вважає надходження фрейму SETTINGS. Протокол допускає інші реалізації, наприклад на мобільному телефоні, який має прокидатися якомога рідше, але вимагає, щоб backoff, запущені одночасно, розходилися в часі і щоб жодна реалізація не намагалася з'єднуватися суттєво частіше.

Застосуйте поняття до рішення

Застосуйте «Що таке експоненційна затримка?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Моніторинг і сповіщення, які попереджають, поки збій ще малий, і цифри, що показують, як часто так і залишається.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Запустила перший дашборд спостережуваності — перший огляд власних систем компанії

Запустила перший дашборд спостережуваності Зв'язатися з нами →

Чому це важливо

Повторна спроба — це додаткове навантаження, і воно приходить у найгірший момент: саме тоді, коли те, до чого звертаються, дає збій. Якщо сервіс сповільнюється через перевантаження, а кожен, хто до нього звертається, одразу пробує знову, навантаження зростає, поки потужність падає. Тому документація Google Cloud Storage відносить повтори без backoff до антипатернів: негайні або дуже короткі повтори можуть

спричинити каскадні збої, коли одна відмова тягне за собою інші. Backoff перетворює повторну спробу з підсилювача на гальмо: що довше триває проблема, то менше кожен клієнт до неї додає.

Сам по собі backoff не рятує, бо клієнти, які зазнають невдачі разом, разом і чекають. У статті Exponential Backoff And Jitter, яку AWS опублікувала в березні 2015 року, Марк Брукер змоделював багато клієнтів, що за допомогою оптимістичного керування конкурентним доступом (optimistic concurrency control) оновлюють один і той самий рядок у віддаленій базі даних. Коли всі клієнти змагалися в кожному раунді, загальний обсяг роботи зростав пропорційно квадрату кількості клієнтів. Експоненційна затримка зі стелею допомагала лише трохи. Виклики й далі надходили скупченнями з тихими проміжками між ними, тож конкуренція майже не зменшувалася. Додавання джитеру, коли кожну паузу вибирають випадково між нулем і експоненційним значенням зі стелею, прибрало проміжки, дало приблизно сталий потік викликів, а у варіанті зі 100 конкурентними клієнтами скоротило кількість викликів більш ніж удвічі. Серед порівняних варіантів backoff без джитеру був, за словами Брукера, «явним аутсайдером». Він також прямо назвав межу: жоден із підходів не змінює того, що робота зростає з квадратом конкуренції. Джитер зменшує марні витрати, але не прибирає самого конфлікту. В оновленні від травня 2023 року в дописі зазначено, що більшість AWS SDK тепер мають експоненційну затримку і джитер у стандартному та адаптивному режимах повторних спроб.

Backoff вирішує, коли пробувати знову, а не чи безпечно це робити. Документація Cloud Storage спершу ставить два питання. Перше — чи вказує відповідь на тимчасову проблему: коди стану HTTP 408, 429 і 5xx, тайм-аути сокетів і розриви TCP-з’єднань. Друге — чи є запит ідемпотентним. Операцію отримання списку можна повторювати вільно. Створення сповіщення Pub/Sub — ні, бо кожен успішний виклик створює нове сповіщення. Між ними лежать умовно ідемпотентні операції, які стає безпечно повторювати, коли вони несуть передумову, наприклад завантаження, що вдається лише тоді, коли номер покоління (generation) об’єкта досі такий, на який розраховує клієнт.

Іноді сервер знає краще за формулу клієнта. Поле заголовка Retry-After, визначене в RFC 9110, повідомляє клієнтові, скільки йому варто зачекати перед наступним запитом, або як дату, або як кількість секунд, а разом із відповіддю 503 Service Unavailable вказує, скільки часу сервіс, за власною оцінкою, буде недоступний. Код стану 429 Too Many Requests із RFC 6585, що

сигналізує про обмеження частоти запитів, може містити те саме поле. Клієнт із циклом backoff може взяти це значення замість власної обчисленої паузи, адже воно надходить від сторони, яка бачить своє навантаження.

Як це виглядає на практиці

Цикл повторних спроб з експоненційною затримкою зі стелею та повним джитером (full jitter) займає кілька рядків на Python:

```
import random
import time

class TransientError(Exception):
    """Тайм-аут, відповідь 408, 429 чи 5xx або розірване з'єднання."""

def call_with_backoff(op, attempts=5, base=1.0, cap=30.0):
    for attempt in range(attempts):
        try:
            return op()
        except TransientError:
            if attempt == attempts - 1:
                raise
            ceiling = min(cap, base * 2**attempt)
            time.sleep(random.uniform(0, ceiling))
```

З цими значеннями п’ять спроб розділяють чотири паузи зі стелями 1, 2, 4 і 8 секунд. Отже, цикл спить щонайбільше 15 секунд, перш ніж здатися, а в середньому — близько 7,5 секунди, бо кожну паузу вибирають рівномірно між нулем і її стелею. Повтор отримує лише `TransientError`; помилка авторизації чи неправильно сформований запит одразу піднімають виняток. І цикл повторює все, що робить `op`, тож `op` має бути безпечною для повторення: читанням або записом, який сервер може розпізнати як повтор.

Клієнтські бібліотеки мають власні версії цього циклу, і типові значення в них різняться навіть для одного й того самого сервісу. Для Cloud Storage клієнт на Python починає з однієї секунди, подвоює кожну паузу до 60 секунд і здається після типового дедлайну 120 секунд. Клієнт на C++ теж починає з однієї секунди й подвоює, але чекає між спробами до п’яти хвилин і продовжує повтори до 15 хвилин, тобто більш ніж усемеро довше. До того ж він повторює й операції, які ніколи не бувають ідемпотентними, якщо програма не ввімкне сувору політику ідемпотентності. Командний рядок gcloud використовує те, що Google називає truncated binary exponential

backoff, і за замовчуванням робить до 32 повторних спроб. Тож дві програми, які викликають той самий API з тією самою помилкою, можуть повідомити про неї через дві хвилини або через чверть години — залежно від мови, якою їх написано.

Очікування не мусить блокувати потік виконання. У фоновій роботі завдання, що зазнало невдачі, можна повернути в чергу з пізнішим часом виконання, і тоді backoff стає властивістю завдання, а не процесу, який спить. Це природне місце для нього щоразу, коли на відповідь ніхто не чекає.

У пошті з failover і журналом доставки, проєкті 2025 року, backoff працює на асинхронному боці системи. Листи йдуть через трьох провайдерів: SendGrid як основний, а SMTP2GO та Azure Communication Services за ним, і перемикання між ними автоматичне. Сам шлях надсилання проходить через клієнт із тайм-аутом 30 секунд, а асинхронний шлях доставки повторює спроби з backoff, щоб коротка заминка не коштувала листа. Сторінка підсумовує результат: коли в одного провайдера поганий день, пошта переходить у деградований режим, а не падає зовсім. Вона ж називає і ціну: три інтеграції, які треба підтримувати, замість однієї.

Де виникають проблеми

Усі клієнти чекають однаково. Без джитеру клієнти, що зазнали невдачі разом, разом і повторюють спробу, і сервер бачить той самий сплеск знову через одну, дві, чотири й вісім секунд. Моделювання Брукера показало, що саме через це проста експоненційна затримка майже не зменшувала конкуренцію. Додайте джитер: у його порівнянні і Full Jitter, і Decorrelated Jitter суттєво скоротили роботу клієнтів і навантаження на сервер, і будь-який із них — розумний вибір за замовчуванням.

Пауза не має стелі, або повторам немає кінця. Необмежене подвоєння перевищує 17 хвилин після десяти невдач, а цикл без ліміту тримає запит живим ще довго після того, як на відповідь уже ніхто не чекає. Google відносить ігнорування лімітів повторів до антипатернів: нескінченні повтори можуть вичерпати ресурси застосунку або без кінця надсилати запити до сервісу, який без втручання не відновиться. Встановіть максимальну паузу й загальний дедлайн відповідно до того, хто чекає: короткі там, де важлива затримка, і довші для пакетної обробки.

Повторюють постійні помилки. Помилка авторизації чи некоректний запит щоразу закінчуються однаково. Повтор лише відкладає помилку, а без ліміту клієнт може опинитися в нескінченному циклі, про що попереджає документація Cloud Storage. Розділіть помилки на тимчасові й постійні ще до циклу, повторюйте лише тимчасові, як-от 408, 429, 5xx і розірвані з'єднання, а решту записуйте в журнал разом із причиною.

Неідемпотентну операцію повторюють наосліп. Якщо запит вичерпав тайм-аут уже після того, як сервер його виконав, повтор виконає дію вдруге, а це, за списком антипатернів Google, означає ненавмисні перезаписи чи видалення даних. Повторюйте лише ідемпотентні операції, робіть записи умовними там, де API це дозволяє, а в інших випадках подбайте, щоб приймальна сторона могла розпізнати дублікат.

Повтори закладено на кількох рівнях. Якщо застосунок робить три спроби, а клієнтська бібліотека під ним робить три спроби на кожну з цих спроб, одна невдача перетворюється на дев'ять спроб; це власний приклад Google, і кожен наступний рівень, що повторює самостійно, знову множить цю кількість. Документація додає, що стільки повторів може призвести до придушення запитів (throttling) і більшої затримки, не покращивши частку успішних запитів. Повторюйте на одному рівні, зазвичай у бібліотеці, а на інших вимкніть повтори або зменште їхню кількість.

Backoff ніколи не скидається, або його ніхто не бачить. Клієнт, що після відновлення зберігає свою останню довгу паузу, за наступного дрібного збою відсиджує максимум; gRPC повертається до початкового backoff, щойно з'єднання прийнято, а TCP переобчислює таймер зі свіжих вимірювань. Скидайте паузу після успіху. І оскільки повтори приховують проблеми від користувачів, вони приховують їх і від команди. Рахуйте їх у системі моніторингу і вмикайте тривогу, коли частка повторів зростає, бо залежність, якій потрібні три спроби на виклик, уже відмовляє.

Що почитати далі

- RFC 6298: Computing TCP's Retransmission Timer — правила Standards Track для тайм-ауту повторної передачі в TCP, зокрема подвоєння щоразу, коли спрацьовує таймер.
- gRPC Connection Backoff Protocol — алгоритм перепідключення, його п'ять параметрів зі значеннями і момент, коли backoff скидається.

- Exponential Backoff And Jitter — моделювання Марка Брукера 2015 року з конкурентними клієнтами, де backoff без джитеру порівнюється з Full, Equal і Decorrelated Jitter.
- Retry strategy — що Cloud Storage вважає безпечним для повтору, типові значення кожної клієнтської бібліотеки і п’ять антипатернів повторних спроб.
- RFC 9110: HTTP Semantics — поле Retry-After, яким сервер повідомляє клієнтові, скільки чекати перед наступним запитом.

Словник

Обирайте за фактичними умовами

Опишіть шлях розгортання, експлуатаційні ризики та чергову команду. Ми можемо визначити зміну з перевірками, шляхом відновлення та чіткою відповідальністю.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке експоненційна затримка?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/exponential-backoff/