Engineer

# Eksponentiell backoff

⏳ en pause mellom forsøk som ganges opp etter hver feil

## Hva det er

Eksponentiell backoff er en regel for hvor lenge man skal vente før man prøver igjen. Etter hvert mislykket forsøk ganges ventetiden med en fast faktor, som oftest to, til den når et tak: ett sekund, så to, fire, åtte og så videre. Regelen ligger i klienten, men er til for tjenesten klienten kaller. En server, en nettverksvei eller en database som svikter under for mye last, kommer seg ikke raskere av å bli spurt igjen med en gang. Den kommer seg når de som kaller den, letter på trykket, og en ventetid som vokser med hver feil, får dem til å lette mer jo lenger problemet varer.

Det eldste og mest utbredte eksempelet finnes i TCP. RFC 6298, dokumentet i Standards Track fra juni 2011 som spesifiserer hvordan TCP beregner tidsavbruddet for retransmisjon (retransmission timeout), setter dette tidsavbruddet til ett sekund før den første rundturstiden (round-trip time) er målt, og sier at en beregnet verdi under ett sekund bør rundes opp. Hver gang tidtakeren løper ut og et segment må sendes på nytt, må verten doble tidsavbruddet. En øvre grense er tillatt, så lenge den er minst 60 sekunder. Når en fersk måling av rundturstiden kommer inn, beregnes tidsavbruddet fra målingene igjen, og det kan bringe det rett ned igjen etter en rekke doblinger. Algoritmen går tilbake til Van Jacobsons artikkel Congestion Avoidance and Control fra 1988, og avsnittet om sikkerhet i RFC-en sier hva som står på spill: internett er avhengig av en korrekt tidtaker for retransmisjon for å holde nettet stabilt og unngå kollaps på grunn av overbelastning (congestion collapse).

I applikasjonskode kommer den samme ideen med en håndfull innstillinger: en innledende ventetid, en multiplikator, en maksimal ventetid der veksten stopper, og en grense for antall forsøk eller for den samlede tiden som brukes på nye forsøk. Skrevet som formel er ventetiden etter den n-te feilen den minste av to verdier: taket, og grunnventetiden ganget med multiplikatoren opphøyd i n, der n telles fra null. AWS-analysen som omtales nedenfor, kaller dette capped

exponential backoff, altså eksponentiell backoff med tak. Taket betyr mer enn det ser ut til. Dobler man en ventetid på ett sekund ti ganger, får man 1 024 sekunder, mer enn 17 minutter, og det er lenger enn de fleste vil vente på noe som helst.

Den siste innstillingen er jitter, et tilfeldig element i hver ventetid. Protokollen for connection backoff i gRPC, som beskriver hvordan gRPC-klienter kobler seg til på nytt etter en mislykket tilkobling, starter på ett sekund, ganger med 1,6, slutter å vokse ved 120 sekunder og bruker en jitter på 0,2, slik at hver ventetid flyttes tilfeldig opp eller ned med inntil 20 %. Den gir også hvert tilkoblingsforsøk minst 20 sekunder på å fullføre, og den nullstiller backoff til startverdien så snart serveren har godtatt en tilkobling, noe den regner som øyeblikket da SETTINGS-rammen kommer fram. Protokollen tillater andre implementasjoner, for eksempel på en mobiltelefon som bør våkne så sjelden som mulig, men krever at backoff-perioder som startet samtidig, sprer seg, og at ingen implementasjon prøver å koble til vesentlig oftere.

Bruk begrepet i en beslutning

Bruk «Hva er eksponentiell backoff?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Overvåking og varsler som sier fra mens en feil fortsatt er liten, og tallene som viser hvor ofte den holder seg liten.

**Se tjenestene våre** Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Bygde selskapets første observerbarhetsdashbord — selskapets første innsyn i egne systemer

**Bygde selskapets første observerbarhetsdashbord** Ta kontakt →

## Hvorfor det er viktig

Et nytt forsøk er ekstra last, og det kommer på verst tenkelige tidspunkt: akkurat når det som kalles, svikter. Hvis en tjeneste går tregere fordi den er overbelastet, og alle som kaller den, svarer med å prøve igjen med en gang, stiger lasten mens kapasiteten faller. Derfor regner dokumentasjonen for Google Cloud Storage nye forsøk uten backoff som et antimønster: umiddelbare eller svært korte nye forsøk kan føre til kaskadefeil, der én feil utløser andre. Med backoff blir det nye forsøket en brems i stedet for en forsterker: jo lenger problemet varer, desto mindre legger hver klient til det.

Backoff alene er ikke nok, for klienter som feiler sammen, venter også sammen. I Exponential Backoff And Jitter, publisert av AWS i mars 2015, simulerte Marc Brooker mange klienter som brukte optimistisk samtidighetskontroll (optimistic

concurrency control) til å oppdatere den samme raden i en ekstern database. Når hver klient konkurrerte i hver runde, vokste det samlede arbeidet med kvadratet av antall klienter. Eksponentiell backoff med tak hjalp bare litt. Kallene kom fortsatt i klynger, med stille hull imellom, så konkurransen falt knapt. Å legge til jitter, ved å trekke hver ventetid tilfeldig mellom null og den eksponentielle verdien med tak, fjernet hullene, ga en tilnærmet jevn strøm av kall og kuttet antall kall med mer enn halvparten i tilfellet med 100 konkurrerende klienter. Av variantene han sammenlignet, var backoff uten jitter med Brookers ord «den klare taperen». Han var også tydelig på grensen: ingen av tilnærmingene endrer det faktum at arbeidet vokser med kvadratet av konkurransen. Jitter reduserer sløsingen; den fjerner ikke konflikten. I en oppdatering fra mai 2023 opplyser innlegget at de fleste AWS SDK-er nå har eksponentiell backoff og jitter i standardmodusen og den adaptive modusen for nye forsøk.

Backoff avgjør når man skal prøve igjen, ikke om det er trygt å gjøre det. Dokumentasjonen for Cloud Storage stiller to spørsmål først. Det ene er om svaret peker på et forbigående problem: HTTP-statuskodene 408, 429 og 5xx, tidsavbrudd på socketen og brutte TCP-forbindelser. Det andre er om forespørselen er idempotent. En listeoperasjon kan gjentas fritt. Å opprette en Pub/Sub-varsling kan ikke det, fordi hvert vellykket kall oppretter en ny varsling. Mellom de to ligger betinget idempotente operasjoner, som blir trygge å prøve på nytt når de har en forutsetning, for eksempel en opplasting som bare lykkes hvis objektets generasjonsnummer fortsatt er det klienten forventer.

Noen ganger vet serveren bedre enn klientens formel. Headerfeltet Retry-After, definert i RFC 9110, forteller klienten hvor lenge den bør vente før en oppfølgende forespørsel, enten som en dato eller som et antall sekunder, og sammen med svaret 503 Service Unavailable sier det hvor lenge tjenesten regner med å være utilgjengelig. Statuskoden 429 Too Many Requests fra RFC 6585, som signaliserer hastighetsbegrensning, kan ha det samme feltet. En klient med en backoff-løkke kan bruke den verdien i stedet for sin egen beregnede ventetid, siden den kommer fra den siden som kan se sin egen last.

## Slik ser det ut i praksis

En løkke for nye forsøk med eksponentiell backoff med tak og full jitter får plass i noen få linjer Python:

```
import random
import time

class TransientError(Exception):
    """Tidsavbrudd, 408-, 429- eller 5xx-svar eller brutt tilkobling."""

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))
```

Med disse verdiene skilles fem forsøk av fire ventetider med tak på 1, 2, 4 og 8 sekunder. Løkken sover derfor i høyst 15 sekunder før den gir opp, og i gjennomsnitt i omtrent 7,5 sekunder, siden hver ventetid trekkes jevnt mellom null og taket sitt. Bare `TransientError` prøves på nytt; en autorisasjonsfeil eller en feilformet forespørsel kaster unntaket med en gang. Og løkken gjentar alt `op` gjør, så `op` må være trygg å gjenta: en lesing, eller en skriving som serveren kan kjenne igjen som en gjentakelse.

Klientbiblioteker har sine egne versjoner av denne løkken, og standardverdiene deres er forskjellige, selv for den samme tjenesten. For Cloud Storage starter Python-klienten på ett sekund, dobler hver ventetid opp til 60 sekunder og gir opp etter en standardfrist på 120 sekunder. C++-klienten starter også på ett sekund og dobler, men venter opptil fem minutter mellom forsøkene og fortsetter med nye forsøk i opptil 15 minutter, mer enn sju ganger så lenge. Den prøver også på nytt operasjoner som aldri er idempotente, med mindre programmet velger en streng idempotenspolicy. Kommandolinjeverktøyet gcloud bruker det Google kaller truncated binary exponential backoff, med opptil 32 nye forsøk som standard. To programmer som kaller det samme API-et med den samme feilen, kan altså rapportere den etter to minutter eller etter et kvarter, alt etter hvilket språk de er skrevet i.

Ventingen trenger ikke å blokkere en tråd. I bakgrunnsarbeid kan en jobb som feiler, legges tilbake i en kø med et senere tidspunkt for kjøring, og da blir backoff en egenskap ved jobben og ikke ved en prosess som sover. Det er det naturlige stedet for den når ingen venter på svaret.

I e-post med failover og leveringslogging, fra 2025, sitter backoff på den asynkrone siden av systemet. Meldingene sendes gjennom tre leverandører, med SendGrid som primær og SMTP2GO og Azure Communication Services i reserve, og failover mellom dem skjer automatisk. Selve sendingen går gjennom en klient med et tidsavbrudd på 30 sekunder, og den asynkrone leveringsveien prøver på nytt med backoff, slik at en kort forstyrrelse ikke koster en melding. Porteføljesiden oppsummerer resultatet: når én leverandør har en dårlig ettermiddag, går e-posten over i degradert drift i stedet for å stanse helt. Den nevner også prisen, som er tre integrasjoner å vedlikeholde i stedet for én.

## Hvor det går galt

**Alle klientene venter like lenge.** Uten jitter prøver klienter som feilet sammen, igjen sammen, og serveren ser den samme toppen igjen etter ett, to, fire og åtte sekunder. Brookers simulering viste at vanlig eksponentiell backoff av denne grunn knapt reduserte konkurransen. Legg til jitter: i hans sammenligning kuttet både Full Jitter og Decorrelated Jitter klientarbeidet og serverlasten betydelig, og begge er et fornuftig standardvalg.

**Ventetiden har ikke noe tak, eller de nye forsøkene tar aldri slutt.** En dobling uten tak passerer 17 minutter etter ti feil, og en løkke uten grense holder en forespørsel i live lenge etter at den som ventet på svaret, har gitt opp. Google regner det å ignorere grenser for nye forsøk som et antimønster: å prøve på nytt i det uendelige kan tømme applikasjonens ressurser eller fortsette å sende forespørsler til en tjeneste som ikke kommer seg uten inngrep. Sett en maksimal ventetid og en samlet frist som er tilpasset den som kaller: korte der latens betyr noe, og lengre for batcharbeid.

**Permanente feil prøves på nytt.** En autorisasjonsfeil eller en ugyldig forespørsel feiler på samme måte hver gang. Et nytt forsøk utsetter bare feilen, og uten en grense kan klienten havne i en uendelig løkke, som dokumentasjonen for Cloud Storage advarer om. Sorter feilene i forbigående og permanente før løkken, prøv bare de forbigående på nytt, som 408, 429, 5xx og brutte tilkoblinger, og logg resten med årsaken.

**En operasjon som ikke er idempotent, prøves på nytt i blinde.** Hvis en forespørsel fikk tidsavbrudd etter at serveren allerede hadde utført den, gjentar et nytt forsøk handlingen, noe Googles liste over antimønstre beskriver som

utilsiktede overskrivinger eller slettinger av data. Prøv bare idempotente operasjoner på nytt, gjør skrivinger betingede der API-et tillater det, og sørg ellers for at mottakersiden kan oppdage et duplikat.

**Nye forsøk er lagt inn i flere lag.** Hvis applikasjonen gjør tre forsøk og klientbiblioteket under den gjør tre forsøk for hvert av disse forsøkene, blir én feil til ni forsøk, som er Googles eget eksempel, og hvert ytterligere lag som prøver på nytt på egen hånd, ganger opp antallet igjen. Dokumentasjonen legger til at så mange nye forsøk kan føre til struping av forespørsler (throttling) og høyere latens uten at andelen vellykkede forespørsler blir bedre. Prøv på nytt i ett lag, som regel i biblioteket, og slå av eller skru ned nye forsøk i de andre.

**Backoff nullstilles aldri, eller ingen ser den.** En klient som beholder sin siste, lange ventetid etter at alt har kommet seg, sitter ut hele maksimumet ved neste lille feil; gRPC går tilbake til den innledende backoff-verdien så snart en tilkobling er godtatt, og TCP beregner tidtakeren sin på nytt fra ferske målinger. Nullstill etter et vellykket kall. Og fordi nye forsøk skjuler problemer for brukerne, skjuler de dem også for teamet. Tell dem i overvåkingen og send et driftsvarsel når andelen nye forsøk stiger, for en avhengighet som trenger tre forsøk per kall, svikter allerede.

## Les mer

- RFC 6298: Computing TCP's Retransmission Timer — reglene i Standards Track for tidsavbruddet for retransmisjon i TCP, inkludert doblingen hver gang tidtakeren løper ut.
- gRPC Connection Backoff Protocol — algoritmen for ny tilkobling, de fem parameterne med verdiene deres, og når backoff nullstilles.
- Exponential Backoff And Jitter — Marc Brookers simulering fra 2015 av konkurrerende klienter, som sammenligner backoff uten jitter med Full, Equal og Decorrelated Jitter.
- Retry strategy — hva Cloud Storage regner som trygt å prøve på nytt, standardverdiene i hvert klientbibliotek og fem antimønstre for nye forsøk.
- RFC 9110: HTTP Semantics — feltet Retry-After, som en server bruker til å fortelle en klient hvor lenge den skal vente før neste forespørsel.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

**Se hva det koster** Ta kontakt →

# Gjør begrepene praktiske

Spør hvordan «Hva er eksponentiell backoff?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

**Se tjenestene våre**

welcome@engineer.company +45 50 10 61 84 København, Danmark

Ta med deg denne siden

Visittkort

**Laget med kjærlighet av D.**

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

**Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.**

**Skrevet ut fra https://engineer.company/nb/glossary/exponential-backoff/**