Engineer

Eksponentiel backoff

⧗ en pause mellem forsøg, der ganges op efter hver fejl

Hvad det er

Eksponentiel backoff er en regel for, hvor længe man venter, før man prøver igen. Efter hvert mislykket forsøg ganges ventetiden med en fast faktor, oftest to, indtil den når et loft: ét sekund, så to, fire, otte og så videre. Reglen bor i klienten, men den findes for den tjenestes skyld, som klienten kalder. En server, en netværksvej eller en database, der fejler under for høj belastning, kommer sig ikke hurtigere af at blive spurgt igen med det samme. Den kommer sig, når de, der kalder den, letter trykket, og en ventetid, der vokser med hver fejl, får dem til at lette det mere, jo længere problemerne varer.

Det ældste og mest udbredte eksempel findes i TCP. RFC 6298, Standards Track-dokumentet fra juni 2011, der fastlægger, hvordan TCP beregner sin timeout for genudsendelse, sætter timeouten til ét sekund, før den første rundturstid er målt, og siger, at en beregnet værdi under ét sekund bør rundes op. Hver gang timeren udløber, og et segment skal sendes igen, skal værten fordoble timeouten. En øvre grænse er tilladt, så længe den er mindst 60 sekunder. Når en ny måling af rundturstiden kommer ind, beregnes timeouten igen ud fra målingerne, og det kan bringe den helt ned igen efter en række fordoblinger. Algoritmen går tilbage til Van Jacobsons artikel Congestion Avoidance and Control fra 1988, og RFC'ens afsnit om sikkerhed siger, hvad der står på spil: internettet er afhængigt af en korrekt timer for genudsendelse for at holde netværket stabilt og undgå congestion collapse, et sammenbrud under overbelastning.

I applikationskode kommer den samme idé med en håndfuld indstillinger: en første ventetid, en multiplikator, en maksimal ventetid, hvor væksten stopper, og en grænse for antallet af forsøg eller for den samlede tid, der bruges på nye forsøg. Skrevet ud er ventetiden efter den n'te fejl den mindste af loftet og basis × multiplikator i n'te potens, talt fra nul. Den analyse fra AWS, der omtales nedenfor, kalder det capped exponential backoff, altså eksponentiel backoff med

loft. Loftet betyder mere, end man skulle tro. Fordobler man en ventetid på ét sekund ti gange, får man 1.024 sekunder, mere end 17 minutter, og det er længere, end de fleste vil vente på noget som helst.

Den sidste indstilling er jitter, et tilfældigt element i hver ventetid. Protokollen for connection backoff i gRPC, der beskriver, hvordan gRPC-klienter forbinder igen efter en mislykket forbindelse, starter ved ét sekund, ganger med 1,6, holder op med at vokse ved 120 sekunder og bruger en jitter på 0,2, så hver ventetid flyttes tilfældigt op eller ned med op til 20 %. Den giver også hvert forbindelsesforsøg mindst 20 sekunder til at blive gennemført, og den nulstiller backoff til startværdien, når serveren har accepteret en forbindelse, hvilket den regner for at ske, når SETTINGS-rammen ankommer. Protokollen tillader andre implementeringer, for eksempel på en mobiltelefon, der skal vågne så sjældent som muligt, men kræver, at backoffs, der er startet på samme tidspunkt, spredes, og at ingen implementering forsøger at forbinde væsentligt oftere.

Brug begrebet i en beslutning

Brug »Hvad er eksponentiel backoff?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Overvågning og alarmer, der siger til, mens fejlen stadig er lille, og tallene, der viser, hvor tit den bliver ved med at være det.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har bygget det første observerbarhedsdashboard — virksomhedens første overblik over sine egne systemer

Har bygget det første observerbarhedsdashboard Kontakt os →

Hvorfor det betyder noget

Et nyt forsøg er ekstra belastning, og det kommer på det værst tænkelige tidspunkt: netop når det, der bliver kaldt, er ved at fejle. Bliver en tjeneste langsommere, fordi den er overbelastet, og prøver alle, der kalder den, igen med det samme, stiger belastningen, mens kapaciteten falder. Dokumentationen til Google Cloud Storage regner derfor nye forsøg uden backoff for et antimønster, fordi øjeblikkelige eller meget korte genforsøg kan føre til kaskadefejl, hvor én fejl udløser andre. Backoff gør det nye forsøg fra en forstærker til en bremse: jo længere problemerne varer, jo mindre bidrager hver klient til dem.

Backoff alene er ikke nok, for klienter, der fejler samtidig, venter også samtidig. I Exponential Backoff And Jitter, udgivet af AWS i marts 2015, simulerede Marc Brooker mange klienter, der brugte optimistisk samtidighedskontrol til at opdatere den samme række i en fjern database. Når alle klienter konkurrerede i

hver runde, voksede det samlede arbejde med kvadratet på antallet af klienter. Eksponentiel backoff med loft hjalp kun lidt. Kaldene kom stadig i klynger med stille huller imellem, så konkurrencen faldt næsten ikke. Da han tilføjede jitter ved at trække hver ventetid tilfældigt mellem nul og den eksponentielle værdi med loft, forsvandt hullerne, kaldene kom i en nogenlunde jævn strøm, og med 100 konkurrerende klienter faldt antallet af kald til under det halve. Blandt de varianter, han sammenlignede, var backoff uden jitter med Brookers ord »den klare taber«. Han var også tydelig om grænsen: ingen af metoderne ændrer ved, at arbejdet vokser med kvadratet på konkurrencen. Jitter mindsker spildet, men fjerner ikke konflikten. I en opdatering fra maj 2023 skriver indlægget, at de fleste AWS SDK'er nu har eksponentiel backoff og jitter indbygget i deres retry-tilstande standard og adaptive.

Backoff afgør, hvornår man prøver igen, ikke om det er sikkert at gøre det. Dokumentationen til Cloud Storage stiller to spørgsmål først. Det ene er, om svaret tyder på et forbigående problem: HTTP-statuskoderne 408, 429 og 5xx, timeouts på sockets og afbrudte TCP-forbindelser. Det andet er, om forespørgslen er idempotent. En list-operation kan gentages frit. Det kan oprettelsen af en Pub/Sub-notifikation ikke, for hvert vellykket kald opretter en ny notifikation. Imellem de to ligger betinget idempotente operationer, der bliver sikre at gentage, når de har en forudsætning med, for eksempel en upload, der kun lykkes, hvis objektets generationsnummer stadig er det, klienten forventer.

Nogle gange ved serveren bedre end klientens formel. Headerfeltet Retry-After, der er defineret i RFC 9110, fortæller klienten, hvor længe den bør vente, før den sender en ny forespørgsel, enten som en dato eller som et antal sekunder, og sammen med et 503 Service Unavailable-svar siger det, hvor længe tjenesten regner med at være utilgængelig. Statuskoden 429 Too Many Requests fra RFC 6585, der signalerer ratebegrænsning, må have det samme felt med. En klient med en backoff-løkke kan bruge den værdi i stedet for sin egen beregnede ventetid, for den kommer fra den side, der kan se sin egen belastning.

Sådan ser det ud i praksis

En løkke med nye forsøg, eksponentiel backoff med loft og fuld jitter fylder kun nogle få linjer i Python:

```
import random
import time

class TransientError(Exception):
    """En timeout, et 408-, 429- eller 5xx-svar eller en tabt forbindelse."""

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 værdier er fem forsøg adskilt af fire ventetider, hvis lofter er 1, 2, 4 og 8 sekunder. Løkken sover derfor højst 15 sekunder, før den giver op, og i gennemsnit omkring 7,5 sekunder, fordi hver ventetid trækkes jævnt mellem nul og dens loft. Kun `TransientError` får et nyt forsøg; en afvist autorisation eller en forkert udformet forespørgsel kaster en undtagelse med det samme. Og løkken gentager, hvad `op` end gør, så `op` skal være sikker at gentage: en læsning eller en skrivning, som serveren kan genkende som en gentagelse.

Klientbiblioteker leverer deres egne udgaver af denne løkke, og deres standardværdier er forskellige, selv for den samme tjeneste. Til Cloud Storage starter Python-klienten ved ét sekund, fordobler hver ventetid op til 60 sekunder og giver op efter en standardfrist på 120 sekunder. C++-klienten starter også ved ét sekund og fordobler, men venter op til fem minutter mellem forsøgene og bliver ved med at prøve i op til 15 minutter, mere end syv gange så længe. Den prøver også igen med operationer, der aldrig er idempotente, medmindre programmet vælger en streng idempotenspolitik. Kommandolinjeværktøjet gcloud bruger det, Google kalder truncated binary exponential backoff, med op til 32 nye forsøg som standard. To programmer, der kalder det samme API med den samme fejl, kan derfor melde den efter to minutter eller efter et kvarter, alt efter hvilket sprog de er skrevet i.

Ventetiden behøver ikke at blokere en tråd. I baggrundsarbejde kan et job, der fejler, sættes tilbage i en kø med et senere tidspunkt for udførelse, og så bliver backoff en egenskab ved jobbet og ikke ved en proces, der sover. Det er det naturlige sted for den, når ingen sidder og venter på resultatet.

I e-mail med failover og leveringslog, fra 2025, hører backoff til på systemets asynkrone side. Mailen går ud gennem tre udbydere, med SendGrid som den primære og SMTP2GO og Azure Communication Services bag den, og der skiftes automatisk mellem dem. Selve afsendelsen går gennem en klient med en timeout på 30 sekunder, mens den asynkrone levering prøver igen med backoff, så en kort forstyrrelse ikke får en besked til at gå tabt. Siden opsummerer resultatet: når en udbyder har en dårlig eftermiddag, bliver leveringen forringet i stedet for at gå ned. Den nævner også prisen, nemlig tre integrationer at holde i gang i stedet for én.

Hvor det går galt

Alle klienter venter lige længe. Uden jitter prøver klienter, der fejlede sammen, igen sammen, og serveren ser den samme top igen efter ét, to, fire og otte sekunder. Brookers simulering viste, at almindelig eksponentiel backoff af den grund næsten ikke mindskede konkurrencen. Tilføj jitter: i hans sammenligning skar både Full Jitter og Decorrelated Jitter markant ned på klienternes arbejde og serverens belastning, og begge er et fornuftigt standardvalg.

Ventetiden har intet loft, eller forsøgene har ingen ende. En fordobling uden loft passerer 17 minutter efter ti fejl, og en løkke uden grænse holder en forespørgsel i live længe efter, at den, der ventede på svaret, er gået videre. Google regner det at ignorere grænser for nye forsøg for et antimønster: at prøve igen i det uendelige kan opbruge programmets ressourcer eller blive ved med at sende forespørgsler til en tjeneste, der ikke kommer sig uden indgriben. Sæt en maksimal ventetid og en samlet frist, der er afstemt efter den, der kalder: kort, hvor latenstiden betyder noget, og længere for batchjob.

Permanente fejl får nye forsøg. En afvist autorisation eller en ugyldig forespørgsel fejler på samme måde ved hvert forsøg. Et nyt forsøg udskyder kun fejlen, og uden en grænse kan klienten ende i en uendelig løkke, som dokumentationen til Cloud Storage advarer om. Sortér fejlene i forbigående og permanente før løkken, prøv kun de forbigående igen, for eksempel 408, 429, 5xx og afbrudte forbindelser, og log resten med deres årsag.

En ikke-idempotent operation gentages i blinde. Fik en forespørgsel timeout, efter at serveren allerede havde udført den, gentager et nyt forsøg handlingen, hvad Googles liste over antimønstre beskriver som utilsigtede overskrivninger

eller sletninger af data. Prøv kun idempotente operationer igen, gør skrivninger betingede, hvor API'et tillader det, og sørg ellers for, at den modtagende side kan genkende en dublet.

Nye forsøg stables i flere lag. Foretager programmet tre forsøg, og foretager klientbiblioteket under det tre forsøg for hvert af de forsøg, bliver én fejl til ni forsøg, hvilket er Googles eget eksempel, og hvert yderligere lag, der prøver igen på egen hånd, ganger antallet op igen. Dokumentationen tilføjer, at så mange nye forsøg kan få forespørgslerne droslet og give højere latenstid uden at forbedre andelen af vellykkede kald. Prøv igen i ét lag, som regel biblioteket, og slå nye forsøg fra eller ned i de andre.

Backoff nulstilles aldrig, eller ingen ser det. En klient, der beholder sin sidste, lange ventetid efter en genopretning, sidder maksimummet ud ved den næste lille fejl; gRPC nulstiller til den første backoff, så snart en forbindelse er accepteret, og TCP beregner sin timer igen ud fra nye målinger. Nulstil efter en succes. Og fordi nye forsøg skjuler problemerne for brugerne, skjuler de dem også for teamet. Tæl dem i overvågningen, og udløs en alarm, når andelen af nye forsøg stiger, for en afhængighed, der skal bruge tre forsøg pr. kald, fejler allerede.

Læs mere

- RFC 6298: Computing TCP's Retransmission Timer — reglerne fra Standards Track for TCP's timeout for genudsendelse, herunder fordoblingen, hver gang timeren udløber.
- gRPC Connection Backoff Protocol — algoritmen for genforbindelse, dens fem parametre med deres værdier, og hvornår backoff nulstilles.
- Exponential Backoff And Jitter — Marc Brookers simulering fra 2015 af konkurrerende klienter, der sammenligner backoff uden jitter med Full, Equal og Decorrelated Jitter.
- Retry strategy — hvad Cloud Storage anser for sikkert at prøve igen, standardværdierne i hvert klientbibliotek og fem antimønstre for nye forsøg.
- RFC 9110: HTTP Semantics — headerfeltet Retry-After, som en server bruger til at fortælle en klient, hvor længe den skal vente før sin næste forespørgsel.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring udrulningsforløbet, driftsrisiciene og dem, der har vagten. Vi kan afgrænse en ændring med kontroller, en vej tilbage og tydeligt ansvar.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er eksponentiel backoff?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

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

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/exponential-backoff/