Engineer

Exponentielles Backoff

⧗ eine bei jedem Fehlschlag vervielfachte Wiederholungspause

Was es ist

Exponentielles Backoff legt fest, wie lange vor einem erneuten Versuch gewartet wird. Nach jedem fehlgeschlagenen Versuch wird die Wartezeit mit einem festen Faktor multipliziert, meist zwei, bis eine Obergrenze erreicht ist: eine Sekunde, dann zwei, vier, acht und so weiter. Die Regel liegt im Client, dient aber dem aufgerufenen Gegenüber. Server, Netzwerkpfade oder Datenbanken, die unter zu hoher Last ausfallen, erholen sich nicht schneller, wenn sofort wieder angefragt wird. Sie erholen sich, wenn ihre Aufrufer nachlassen. Eine mit jedem Fehler wachsende Wartezeit drosselt deren Anfragen umso stärker, je länger das Problem anhält.

Das älteste und am weitesten verbreitete Beispiel steckt in TCP. RFC 6298, das Standards-Track-Dokument vom Juni 2011 zur Berechnung des TCP-Zeitlimits für Neuübertragungen, setzt dieses vor der ersten Messung der Paketumlaufzeit auf eine Sekunde. Es besagt außerdem, dass ein berechneter Wert unter einer Sekunde aufgerundet werden sollte. Läuft der Timer ab und muss ein Segment erneut gesendet werden, muss der Host das Zeitlimit jeweils verdoppeln. Eine Obergrenze ist zulässig, sofern sie mindestens 60 Sekunden beträgt. Sobald eine neue Messung der Paketumlaufzeit vorliegt, wird das Zeitlimit erneut aus den Messwerten berechnet. Nach einer Folge von Verdopplungen kann es dadurch sofort wieder sinken. Der Algorithmus geht auf Van Jacobsons Aufsatz Congestion Avoidance and Control von 1988 zurück. Der Sicherheitsabschnitt des RFC nennt, was auf dem Spiel steht: Das Internet ist auf einen korrekten Neuübertragungs-Timer angewiesen, damit das Netz stabil bleibt und ein Überlastungskollaps vermieden wird.

Im Anwendungscode hat dieselbe Idee einige Einstellungen: eine anfängliche Wartezeit, einen Multiplikator, eine maximale Wartezeit als Wachstumsgrenze und eine Höchstzahl von Versuchen oder eine Grenze für die gesamte mit Wiederholungsversuchen verbrachte Zeit. Ausgeschrieben ist die Wartezeit nach

dem n-ten Fehler der kleinere Wert aus der Obergrenze und Basis × Multiplikator hoch n, wobei ab null gezählt wird. Die unten besprochene AWS-Analyse nennt das begrenztes exponentielles Backoff. Die Obergrenze ist wichtiger, als sie wirkt. Eine zehnmal verdoppelte Wartezeit von einer Sekunde ergibt 1.024 Sekunden, mehr als 17 Minuten. Für die meisten Aufrufer ist das länger, als sie überhaupt warten würden.

Die letzte Einstellung ist Jitter, ein Zufallsanteil in jeder Wartezeit. Das gRPC-Verbindungs-Backoff-Protokoll beschreibt, wie Clients nach einer gescheiterten Verbindung erneut eine Verbindung herstellen. Es beginnt bei einer Sekunde, multipliziert mit 1,6, beendet das Wachstum bei 120 Sekunden und verwendet einen Jitter von 0,2. Jede Wartezeit wird dadurch zufällig um bis zu 20 % nach oben oder unten verschoben. Außerdem erhält jeder Verbindungsversuch mindestens 20 Sekunden Zeit. Sobald der Server eine Verbindung akzeptiert hat, wird das Backoff auf seinen Anfangswert zurückgesetzt. Als diesen Zeitpunkt wertet das Protokoll das Eintreffen des SETTINGS-Frames. Das Protokoll lässt andere Implementierungen zu, etwa für ein Mobiltelefon, das möglichst selten aufwachen soll. Es verlangt aber, gleichzeitige Wiederholungsversuche zeitlich zu verteilen und eine wesentlich höhere Versuchsrate zu vermeiden.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist exponentielles Backoff?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Monitoring und Alarme melden sich, solange ein Fehler noch klein ist; Kennzahlen zeigen, wie oft das gelingt.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. Das erste Observability-Dashboard aufgebaut — der erste Einblick des Unternehmens in die eigenen Systeme

Das erste Observability-Dashboard aufgebaut Kontakt aufnehmen →

Warum es wichtig ist

Ein Wiederholungsversuch erzeugt zusätzliche Last und kommt zum ungünstigsten Zeitpunkt: genau dann, wenn das aufgerufene System ausfällt. Wird ein Dienst wegen Überlastung langsam und reagiert jeder Aufrufer mit einem sofortigen neuen Versuch, steigt die Last, während die Kapazität sinkt. Die Dokumentation von Google Cloud Storage führt Wiederholungen ohne Backoff deshalb als Antipattern auf. Sofortige oder sehr kurz aufeinanderfolgende Versuche können Kaskadenausfälle auslösen, bei denen ein Fehler weitere verursacht. Backoff macht aus dem Wiederholungsversuch eine Bremse statt eines Verstärkers: Je länger das Problem anhält, desto weniger trägt jeder Client dazu bei.

Backoff allein reicht nicht, weil gemeinsam scheiternde Clients auch gemeinsam warten. In Exponential Backoff And Jitter, von AWS im März 2015 veröffentlicht, simulierte Marc Brooker viele Clients, die mit optimistischer Nebenläufigkeitskontrolle dieselbe Zeile einer entfernten Datenbank aktualisieren. Wenn in jeder Runde jeder Client konkurrierte, wuchs die Gesamtarbeit mit dem Quadrat der Clientzahl. Begrenztes exponentielles Backoff half nur wenig. Die Aufrufe kamen weiterhin in Gruppen mit ruhigen Lücken dazwischen, sodass die Konkurrenz kaum sank. Jitter, bei dem jede Wartezeit zufällig zwischen null und dem begrenzten exponentiellen Wert gezogen wurde, beseitigte die Lücken und hinterließ eine annähernd konstante Aufrufrate. Im Fall mit 100 konkurrierenden Clients sank die Zahl der Aufrufe um mehr als die Hälfte. Unter den verglichenen Varianten war Backoff ohne Jitter nach Brookers Worten „der klare Verlierer“. Die Grenze benannte er ebenso ausdrücklich: Keiner der Ansätze ändert daran etwas, dass die Arbeit quadratisch mit der Konkurrenz wächst. Jitter verringert die Verschwendung, beseitigt aber nicht den Konflikt. In einer Aktualisierung vom Mai 2023 hält der Beitrag fest, dass die meisten AWS-SDKs inzwischen exponentielles Backoff und Jitter in ihren Standard- und adaptiven Wiederholungsmodi enthalten.

Backoff entscheidet, wann erneut versucht wird, nicht, ob das sicher ist. Die Cloud-Storage-Dokumentation stellt zwei Fragen voran. Erstens: Deutet die Antwort auf ein vorübergehendes Problem hin? Dazu zählen HTTP-Statuscodes 408, 429 und 5xx, Socket-Zeitüberschreitungen und abgebrochene TCP-Verbindungen. Zweitens: Ist die Anfrage idempotent? Eine Auflistung lässt sich beliebig wiederholen. Das Anlegen einer Pub/Sub-Benachrichtigung nicht, weil jeder erfolgreiche Aufruf eine neue Benachrichtigung erzeugt. Dazwischen liegen bedingt idempotente Operationen, deren Wiederholung durch eine Vorbedingung sicher wird. Ein Upload kann etwa nur dann gelingen, wenn die Generationsnummer des Objekts weiterhin der vom Client erwarteten entspricht.

Manchmal weiß der Server mehr als die Formel des Clients. Das in RFC 9110 definierte Headerfeld Retry-After sagt dem Client, wie lange er vor einer Folgeanfrage warten sollte, entweder als Datum oder als Anzahl von Sekunden. Bei einer Antwort 503 Service Unavailable nennt es die voraussichtliche Dauer der Nichtverfügbarkeit. Auch der Status 429 Too Many Requests aus RFC 6585, der eine Ratenbegrenzung signalisiert, darf dieses Feld enthalten. Ein Client mit einer Backoff-Schleife kann den Wert statt seiner eigenen berechneten Wartezeit verwenden, weil er von der Seite stammt, die ihre eigene Last sieht.

Wie es in der Praxis aussieht

Eine Wiederholungsschleife mit begrenztem exponentiellem Backoff und Full Jitter passt in wenige Zeilen Python:

```
import random
import time

class TransientError(Exception):
    """A timeout, a 408, 429 or 5xx response, or a dropped connection."""

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))
```

Mit diesen Werten liegen zwischen fünf Versuchen vier Wartezeiten mit Obergrenzen von 1, 2, 4 und 8 Sekunden. Die Schleife wartet deshalb insgesamt höchstens 15 Sekunden, bevor sie aufgibt, und im Mittel etwa 7,5 Sekunden, weil jede Wartezeit gleichverteilt zwischen null und ihrer Obergrenze gezogen wird. Nur bei `TransientError` wird erneut versucht. Ein Autorisierungsfehler oder eine fehlerhaft aufgebaute Anfrage löst sofort eine Ausnahme aus. Die Schleife wiederholt alles, was `op` tut. `op` muss sich daher sicher wiederholen lassen: ein Lesevorgang oder ein Schreibvorgang, dessen Wiederholung der Server erkennt.

Clientbibliotheken liefern eigene Varianten dieser Schleife. Ihre Standardwerte unterscheiden sich sogar für denselben Dienst. Der Python-Client für Cloud Storage beginnt bei einer Sekunde, verdoppelt jede Wartezeit bis auf 60 Sekunden und gibt nach einer standardmäßigen Gesamtfrist von 120 Sekunden auf. Der C++-Client beginnt ebenfalls bei einer Sekunde und verdoppelt, wartet zwischen Versuchen aber bis zu fünf Minuten und versucht es insgesamt bis zu 15 Minuten lang, mehr als siebenmal so lange. Er wiederholt auch Operationen, die nie idempotent sind, sofern das Programm nicht ausdrücklich eine strenge Idempotenzrichtlinie aktiviert. Die gcloud-Kommandozeile verwendet das von Google als begrenztes binäres exponentielles Backoff bezeichnete Verfahren,

standardmäßig mit bis zu 32 Wiederholungsversuchen. Zwei Programme mit derselben API und demselben Fehler können ihn deshalb je nach Programmiersprache nach zwei Minuten oder nach einer Viertelstunde melden.

Das Warten muss keinen Thread blockieren. Bei Hintergrundarbeit kann ein gescheiterter Job mit einem späteren Fälligkeitstermin zurück in eine Queue gestellt werden. Das Backoff wird dann zur Eigenschaft des Jobs statt eines schlafenden Prozesses. Dort ist es sinnvoll aufgehoben, wenn niemand auf die Antwort wartet.

Bei E-Mail mit Failover und Zustellprotokollen aus der seit 2025 laufenden Tätigkeit liegt Backoff auf der asynchronen Seite des Systems. E-Mails werden über drei Anbieter verschickt: SendGrid ist primär, SMTP2GO und Azure Communication Services stehen dahinter, und der Wechsel bei Ausfällen erfolgt automatisch. Der Sendeweg selbst führt über einen Client mit einem Zeitlimit von 30 Sekunden. Der asynchrone Zustellweg hat Wiederholungsversuche und Backoff, sodass eine kurze Störung keine Nachricht kostet. Die Seite fasst das Ergebnis so zusammen: Hat ein Anbieter einen schlechten Nachmittag, werden E-Mails über einen eingeschränkten Zustellweg versandt, statt ganz auszufallen. Sie nennt auch den Preis: drei Integrationen funktionsfähig zu halten statt einer.

Wo es schiefgeht

Alle Clients warten gleich lange. Ohne Jitter wiederholen gemeinsam gescheiterte Clients ihre Anfragen gemeinsam. Der Server sieht dieselbe Lastspitze nach einer, zwei, vier und acht Sekunden erneut. Brookers Simulation zeigte, dass einfaches exponentielles Backoff die Konkurrenz deshalb kaum verringerte. Ergänzen Sie Jitter: In seinem Vergleich reduzierten Full Jitter und Decorrelated Jitter sowohl die Clientarbeit als auch die Serverlast wesentlich. Beide sind eine sinnvolle Standardeinstellung.

Die Wartezeit hat keine Obergrenze oder die Versuche kein Ende. Unbegrenztes Verdoppeln überschreitet nach zehn Fehlschlägen 17 Minuten. Eine Schleife ohne Begrenzung hält eine Anfrage lange am Leben, nachdem niemand mehr auf ihre Antwort wartet. Google nennt das Ignorieren von Wiederholungsgrenzen ein Antipattern: Unbegrenzte Versuche können die Ressourcen der Anwendung erschöpfen oder weiter Anfragen an einen Dienst schicken, der sich ohne Eingriff nicht erholen wird. Setzen Sie eine maximale Wartezeit und eine zum Aufrufer passende Gesamtfrist: kurz, wenn geringe Latenz wichtig ist, länger bei Stapelverarbeitung.

Dauerhafte Fehler werden erneut versucht. Ein Autorisierungsfehler oder eine ungültige Anfrage scheitert bei jedem Versuch auf dieselbe Weise. Eine Wiederholung verzögert nur die Fehlermeldung. Ohne Begrenzung kann der Client in einer Endlosschleife landen, warnt die Cloud-Storage-Dokumentation. Ordnen Sie Fehler vor der Schleife als vorübergehend oder dauerhaft ein. Wiederholen Sie nur die vorübergehenden, etwa 408, 429, 5xx und Verbindungsabbrüche, und protokollieren Sie den Rest samt Ursache.

Eine nicht idempotente Operation wird blind wiederholt. Wurde eine Anfrage wegen Zeitüberschreitung abgebrochen, nachdem der Server sie bereits ausgeführt hatte, wiederholt der nächste Versuch die Aktion. Googles Antipattern-Liste beschreibt dies als unbeabsichtigtes Überschreiben oder Löschen von Daten. Wiederholen Sie nur idempotente Operationen, knüpfen Sie Schreibvorgänge an Bedingungen, wo die API das zulässt, und stellen Sie andernfalls sicher, dass die Gegenseite ein Duplikat erkennen kann.

Mehrere Schichten stapeln Wiederholungsversuche. Führt die Anwendung drei Versuche aus und die darunterliegende Clientbibliothek für jeden davon wiederum drei, werden aus einem Fehler neun Versuche. Das ist Googles eigenes Beispiel. Jede weitere Schicht mit eigenen Wiederholungen multipliziert die Zahl erneut. Die Dokumentation ergänzt, dass so viele Wiederholungen zur Drosselung von Anfragen und höherer Latenz führen können, ohne die Erfolgsquote zu verbessern. Wiederholen Sie in einer einzigen Schicht, üblicherweise der Bibliothek, und schalten Sie Wiederholungen in den anderen ab oder reduzieren Sie sie.

Das Backoff wird nie zurückgesetzt oder niemand bemerkt es. Ein Client, der seine letzte lange Wartezeit nach einer Erholung behält, wartet beim nächsten kleinen Fehler das Maximum ab. gRPC setzt beim Akzeptieren einer Verbindung sofort auf das anfängliche Backoff zurück; TCP berechnet seinen Timer aus neuen Messungen. Setzen Sie nach einem Erfolg zurück. Weil Wiederholungen Probleme vor Nutzern verbergen, verbergen sie diese auch vor dem Team. Zählen Sie sie im Monitoring und lösen Sie einen Alarm aus, wenn ihre Rate steigt. Eine Abhängigkeit, die drei Versuche pro Aufruf braucht, fällt bereits aus.

Weiterführende Lektüre

- RFC 6298: Berechnung des TCP-Neuübertragungs-Timers — die Standards-Track-Regeln für das TCP-Zeitlimit zur Neuübertragung, einschließlich der Verdopplung bei jedem Ablauf des Timers.

- gRPC-Verbindungs-Backoff-Protokoll — der Algorithmus zum erneuten Verbinden, seine fünf Parameter mit Werten und der Zeitpunkt zum Zurücksetzen des Backoffs.
- Exponentielles Backoff und Jitter — Marc Brookers Simulation konkurrierender Clients von 2015 im Vergleich ohne Jitter sowie mit Full, Equal und Decorrelated Jitter.
- Wiederholungsstrategie — welche Operationen Cloud Storage als sicher wiederholbar betrachtet, die Standardwerte jeder Clientbibliothek und fünf Antipatterns für Wiederholungen.
- RFC 9110: HTTP-Semantik — das Feld Retry-After, mit dem ein Server dem Client die Wartezeit vor dessen nächster Anfrage mitteilt.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Bereitstellungsweg, die Betriebsrisiken und die Bereitschaftsverantwortlichen. Wir können eine Änderung mit Prüfungen, einem Wiederherstellungsweg und klaren Zuständigkeiten abgrenzen.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist exponentielles Backoff?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/exponential-backoff/