Engineer

指数バックオフ

⧗ 失敗のたびに一定の倍率で延ばす再試行までの待ち時間

どのようなものか

指数バックオフは、再試行までどのくらい待つかを決める規則です。失敗のたびに、待ち時間を、通常は二という固定の倍率で増やし、上限で止めます。一秒、二秒、四秒、八秒、といった具合です。規則はクライアントにありますが、守るのは、呼び出す相手です。過負荷で失敗するサーバー、ネットワーク経路、データベースは、すぐもう一度呼んでも、早く復旧しません。呼び出す側が控えたときに回復します。失敗ごとに待ち時間を延ばせば、問題が長引くほど、さらに控えるようになります。

最も古く、最も広く使われる例はTCPです。2011年6月の標準化過程の文書、RFC 6298は、TCPの再送タイムアウトの計算を定めます。最初の往復時間を測る前は一秒にし、計算結果が一秒未満なら、一秒へ切り上げるべきとします。タイマーが切れ、セグメントを再送するたびに、ホストはタイムアウトを倍にしなければなりません。上限を設けてもよいものの、少なくとも60秒が必要です。新しい往復時間を測ると、測定値から再計算します。何度も倍にした後でも、すぐ短くできることがあります。アルゴリズムは、Van Jacobsonの1988年の論文、Congestion Avoidance and Controlへさかのぼります。RFCのセキュリティ節は、何を守る必要があるかを述べます。インターネットは、ネットワークの安定を保ち、輻輳崩壊を防ぐため、正しい再送タイマーに依存します。

アプリのコードでは、同じ考え方に、いくつかの設定を加えます。初期の待ち時間、倍率、増加を止める最大待ち時間、試行回数か再試行の総時間の上限です。式にすれば、ゼロから数えてn回目の失敗後の待ち時間は、上限と、基準値 × 倍率のn乗の、小さい方です。後述のAWSの分析は、上限付き指数バックオフと呼びます。上限は、見た目以上に重要です。一秒を十回倍にすると1,024秒で、17分を超えます。大半の呼び出し側は、何を呼び出していても、そこまでは待ちません。

最後の設定は、各待ち時間にランダム性を持たせる、ジッターです。gRPCの接続バックオフプロトコルは、接続失敗後の再接続を説明します。一秒から始め、1.6倍し、120秒で増加を止め、0.2のジッターを適用します。各待ち時間が、ランダムに最大20%増減します。各接続試行には、完了まで少なくとも20秒を与えます。サーバーが接続を受け入れたら、初期値へ戻します。このプロトコルが、受け入れた時点と見なすのは、SETTINGSフレームが届いたときです。できるだけスリープからの復帰回数を抑えたい携帯電話など、別の実装も認めます。ただし、同時の再試行を分散させ、試行頻度を大幅に高めないことを要求します。

概念を判断に生かす

「指数バックオフとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 障害が小さいうちに知らせる監視とアラートを整え、安定した状態をどれだけ維持できているかを数値で示します。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 社内初のオブザーバビリティダッシュボードを構築 — 社内システムの状態を初めて可視化

社内初のオブザーバビリティダッシュボードを構築 お問い合わせ →

なぜ重要なのか

再試行は、追加の負荷です。しかも、呼び出す相手が失敗している、最も悪い瞬間に届きます。過負荷でサービスが遅くなり、すべての呼び出し側が直ちに再試行すると、処理能力が下がる一方で負荷が増えます。そのため、Google Cloud Storageの文書は、バックオフなしの再試行をアンチパターンに挙げます。即座、または極端に短い間隔の再試行は、一つの失敗がほかを呼ぶ、連鎖障害につながるためです。バックオフは、再試行を、増幅器からブレーキへ変えます。問題が長引くほど、各クライアントが加える負荷は減ります。

バックオフだけでは足りません。同時に失敗するクライアントは、同時に待つためです。AWSが2015年3月に公開したExponential Backoff And Jitterで、Marc Brookerは、リモートデータベースの同じ行を、楽観的な同時実行制御で更新する、多数のクライアントをシミュレーションしました。各ラウンドで全員が競うと、仕事の総量は、クライアント数の二乗で増えました。上限付き指数バックオフの効果は、わずかでした。呼び出しがまとまって届き、その間には呼び出しのない空白ができるため、競合はほぼ減りませんでした。ゼロから上限付き指数値まで、各待ち時間をランダムに選ぶジッターを加えると、空白がなくなり、呼び出しは、ほぼ一定の頻度になりました。100クライアントが競う例では、呼び出し数を半分未満へ減らしました。比較した方式の中で、ジッターなしのバックオフは、Brookerの言葉では「明らかな敗者」でした。限界も明記しています。ど

の方法も、仕事が競合の二乗で増える事実は変えません。ジッターは無駄を減らしても、競合をなくしません。2023年5月の追記では、大半のAWS SDKが、標準と適応型の再試行モードに、指数バックオフとジッターを備えると述べます。

バックオフが決めるのは、いつ再試行するかで、安全かどうかではありません。Cloud Storageの文書は、先に二つを問います。一つは、応答が一時的な問題を示すかです。HTTP 408、429、5xxのステータスコード、ソケットのタイムアウト、TCP切断などです。もう一つは、リクエストが冪等かです。一覧を取得する操作は、自由に繰り返せます。Pub/Sub通知の作成は、できません。成功するたびに、新しい通知を作るためです。中間には、条件付きで冪等な操作があります。オブジェクトの世代番号が、まだクライアントの期待するものと一致する場合だけ成功するアップロードなど、前提条件を付ければ、安全に再試行できます。

クライアントの式より、サーバーの方がよく知っていることもあります。RFC 9110のRetry-Afterヘッダーフィールドは、次のリクエストまで待つべき時間を、日時か秒数で伝えます。503 Service Unavailableでは、サービスが予想する利用不能時間です。RFC 6585の429 Too Many Requestsは、流量制限を示し、同じフィールドを含めることがあります。バックオフのループを持つクライアントは、自分で計算した待ち時間の代わりに、その値を使えます。自分の負荷が見える側からの情報だからです。

実際の活用例

上限付き指数バックオフとFull Jitterを使う再試行ループは、Pythonの数行に収まります。

```
import random
import time

class TransientError(Exception):
    """タイムアウト、408・429・5xxの応答、または接続の切断。"""

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))
```

この値では、五回の試行の間に、上限が1、2、4、8秒の、四回の待機があります。ゼロから各上限まで一様に選ぶため、諦めるまでに眠る時間の合計は、最大15秒、平均約7.5秒です。再試行するのは `TransientError` だけです。認可の失敗や不正な形式のリクエストは、その場で例外になります。ループは、 `op` の行うすべてを繰り返します。そのため、 `op` は安全に繰り返せる必要があります。読み取りか、サーバーが重複と認識できる書き込みです。

クライアントライブラリーにも、このループの実装があり、同じサービスでも既定値は違います。Cloud StorageのPythonクライアントは、一秒から始め、最大60秒まで倍にし、既定の総期限120秒で諦めます。C++クライアントも、一秒から倍にしますが、試行の間は最大五分待ち、最大15分間、再試行します。七倍を超える長さです。厳密な冪等性ポリシーを明示的に選ばなければ、決して冪等にならない操作まで再試行します。gcloudのコマンドラインは、Googleが打ち切り付き二進指数バックオフと呼ぶ方式を使い、既定で最大32回の再試行を行います。同じAPIで同じ失敗に遭遇する二つのプログラムが、書いた言語によって、二分で報告することも、十五分で報告することもあります。

待機で、スレッドを止める必要はありません。バックグラウンド処理なら、失敗したジョブを、後の実行予定時刻でキューへ戻せます。バックオフは、スリープ中のプロセスではなく、ジョブの属性になります。誰も答えを待たないときには、それが自然な置き場所です。

2025年からの職務で手がけたフェイルオーバーと配信ログを備えるメールでは、バックオフを、非同期側に置きます。メールは、主系のSendGridと、後続のSMTP2GO、Azure Communication Servicesの三事業者を使って送信し、切り替えは自動です。送信経路自体は、30秒のタイムアウトを持つクライアントを通ります。非同期の配信経路には、再試行とバックオフがあり、短い不調でメッセージを失わないようにします。事例は、結果をこうまとめます。一つの事業者の調子が悪いとき、メールは停止するのではなく、機能を落とした経路を通ります。代償は、一つではなく、三つの連携を動かし続けることです。

問題が起きるところ

全クライアントが、同じ時間だけ待ちます。 ジッターなしでは、同時に失敗したクライアントが、同時に再試行します。サーバーには、一秒、二秒、四秒、八秒後に、同じ負荷の山が来ます。Brookerのシミュレーションでも、そのため、単純な指数バックオフは、競合をほとんど減らしませんでした。ジッターを加えます。比較では、Full JitterもDecorrelated Jitterも、クライアントの仕事量とサーバー負荷を大きく減らしました。どちらも、妥当な既定の選択です。

待ち時間に上限がないか、再試行が終わりません。 上限なく倍にすると、十回の失敗後には17分を超えます。回数や時間を制限しないループは、誰も返答を待たなくなってからも、リクエストを生かし続けます。Googleは、再試行の制限を無視することを、アンチパターンに挙げます。無期限の再試行で、アプリの資源を使い尽くしたり、介入なしには回復しないサービスへ、送り続けたりするためです。呼び出す側に合う最大待ち時間と総期限を決めます。レイテンシーが重要なら短く、バッチ処理なら長くします。

恒久的なエラーを、再試行します。 認可の失敗や不正なリクエストは、毎回同じように失敗します。再試行は、エラーを遅らせるだけです。Cloud Storageの文書が警告するように、制限がなければ、クライアントが無限ループへ入ることもあります。ループの前に、一時的なエラーと恒久的なエラーを分け、408、429、5xx、切断など、一時的なものだけを再試行します。残りは、原因と一緒にログへ記録します。

冪等でない操作を、無条件に再試行します。 サーバーがすでに処理した後でタイムアウトすると、再試行が同じ操作を繰り返します。Googleのアンチパターンの一覧は、意図しない上書きやデータ削除と説明します。冪等な操作だけを再試行し、APIが認めるなら書き込みへ条件を付け、それ以外では、受け取る側が重複を検出できるようにします。

複数の層で、再試行を重ねます。 アプリが三回試し、その一回ごとに、下のクライアントライブラリーも三回試すと、一つの失敗が、九回の試行になります。Google自身の例で、さらに自分で再試行する層が増えるたびに、回数を掛け合わせます。文書は、これほどの再試行が、成功率を上げることなく、リクエストの流量制限やレイテンシーの増加を招くこともあると付け加えます。再試行は、通常はライブラリーの、一層だけで行い、ほかの層では無効にするか減らします。

バックオフを戻さないか、誰もその状況を見ていません。 回復後も最後の長い待ち時間を保つクライアントは、次の小さい失敗でも、最大時間を待ち切ります。gRPCは、接続を受け入れたら直ちに初期値へ戻し、TCPは、新しい測定からタイマーを再計算します。成功したら、元に戻します。再試行が問題を利用者から隠すと、チームからも隠れます。監視で回数を数え、頻度が上がったらアラートを出します。一回の呼び出しに三回の試行が必要な依存先は、すでに失敗しています。

参考資料

- RFC 6298：TCPの再送タイマーの計算 — タイマーが切れるたびの倍増を含む、TCP再送タイムアウトについての、標準化過程の規則です。
- gRPCの接続バックオフプロトコル — 再接続のアルゴリズム、五つのパラメーターと値、バックオフを戻す時点です。
- 指数バックオフとジッター — 競合するクライアントを扱うMarc Brookerの2015年のシミュレーションです。ジッターなしと、Full、Equal、Decorrelated Jitterを比較します。
- 再試行の方針 — Cloud Storageが安全に再試行できるとする操作、各クライアントライブラリーの既定値、五つの再試行のアンチパターンです。
- RFC 9110：HTTPの意味論 — サーバーが、次のリクエストまで待つ時間をクライアントへ伝える、Retry-Afterフィールドです。

用語集

実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「指数バックオフとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

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

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/exponential-backoff/