Engineer

Warteschlange

Geordnete Arbeit, die ein anderes Programm später abholt

Was es ist

Eine Warteschlange ist ein Zwischenspeicher zwischen zwei Programmen: Eine Seite legt Nachrichten oder Aufgaben hinein, die andere nimmt sie heraus, sobald sie bereit ist, gewöhnlich die ältesten zuerst. Die meisten Systeme betreiben mehrere Warteschlangen, eine je Arbeitsart. Das hinzufügende Programm ist der Produzent, das abholende der Konsument. Was die Nachrichten dazwischen speichert, ist der Broker: ein eigener Server oder eine Tabelle in einer bereits vorhandenen Datenbank.

So müssen beide Seiten nicht mehr aufeinander warten. Eine Webanfrage, für die eine E-Mail gesendet werden muss, fügt eine Nachricht hinzu und antwortet dem Nutzer sofort. Ein Worker holt sie kurz danach ab. Kommen hundert gleichzeitig, arbeitet er sie in seinem eigenen Tempo ab. Aus einer Lastspitze wird ein Rückstand, aus einer Abhängigkeit eine Verzögerung.

Zwei Begriffe beschreiben die Zustellungszusagen einer Warteschlange. Der erste ist die Bestätigung: Ein Konsument teilt dem Broker mit, wann er eine Nachricht fertig bearbeitet hat; erst dann wird sie entfernt. Im von RabbitMQ umgesetzten AMQP-0-9-1-Modell geht eine Nachricht an einen anderen Konsumenten, wenn der bisherige vor der Bestätigung stirbt. Der zweite Begriff ist Dauerhaftigkeit: Eine dauerhafte Warteschlange mit persistenten Nachrichten übersteht einen Brokerneustart; eine nur im Arbeitsspeicher gehaltene nicht.

Drei benachbarte Begriffe sind zu unterscheiden. Eine Nachrichtenwarteschlange transportiert Daten zwischen Diensten. Eine Aufgabenwarteschlange transportiert Arbeitseinheiten für Hintergrundworker, jeweils mit Argumenten, Zustand und Versuchszähler. Ein Log, das Modell hinter Apache Kafka, sieht von außen ähnlich aus, behält Ereignisse aber nach dem

Lesen für eine je Topic festgelegte Aufbewahrungsdauer. Mehrere Leser können so dieselbe Historie erneut durchlaufen. Lesen aus einer Warteschlange entfernt die Nachricht. Lesen aus einem Log verschiebt nur die Position des Lesers.

Eine Warteschlange muss kein separates Produkt sein. PostgreSQL kann die Rolle mit einer Aufgabentabelle übernehmen: Eine Abfrage beansprucht die nächste Zeile und überspringt dabei bereits von anderen Workern gesperrte Zeilen. Das Handbuch nennt warteschlangenartige Tabellen als Grund für die Klausel `SKIP LOCKED`:

```
WITH next AS (
  SELECT id
    FROM jobs
   WHERE state = 'available'
   ORDER BY id
   LIMIT 1
     FOR UPDATE SKIP LOCKED
)
UPDATE jobs
   SET state = 'running',
       attempt = attempt + 1
  FROM next
 WHERE jobs.id = next.id
RETURNING jobs.id, jobs.args;
```

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist eine Warteschlange?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Zahlen kommen termingerecht dort an, wo sie gebraucht werden, ohne dass am Monatsende jemand eine Tabelle kopieren muss.

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. Ein Infrastrukturplan für 14 Abteilungen — 14 Forschungsabteilungen mit einem gemeinsamen Infrastrukturplan

Ein Infrastrukturplan für 14 Abteilungen Kontakt aufnehmen →

Warum es wichtig ist

Eine Warteschlange entscheidet, worauf ein Nutzer wartet. Darin abgelegte Arbeit geschieht nach dem Absenden der Antwort. Eine Seite, die eine E-Mail, ein PDF, den Neuaufbau eines Suchindexes und eine Neuberechnung auslöst, kann deshalb in der zum Schreiben von vier Zeilen benötigten Zeit antworten; die vier

langsamen Aufgaben werden danach fertig. Für einen Käufer ist das der Unterschied zwischen einem in wenigen hundert Millisekunden antwortenden Bezahlvorgang und einem, der erst antwortet, wenn der Mailserver bereit ist.

Sie entscheidet auch, was einen Ausfall übersteht. In einem Hintergrundthread begonnene Arbeit lebt im Speicher des Prozesses. Bei einem Deployment, das den Prozess neu startet, geht die Arbeit verloren, ohne eine Spur ihres Auftrags zu hinterlassen. Arbeit auf einer dauerhaften Warteschlange ist dagegen ein Datensatz: Ihre Ausführung lässt sich erneut versuchen; sie lässt sich untersuchen und zählen.

Außerdem trennt sie zwei Geschwindigkeiten. Produzenten folgen dem eingehenden Verkehr, Konsumenten dem langsamsten von ihnen aufgerufenen Dienst. Eine Warteschlange dazwischen fängt den Unterschied auf. Ihre Tiefe wird damit zu einer der aussagekräftigsten Kennzahlen eines Systems. Eine seit einer Stunde wachsende Warteschlange zeigt, dass die Konsumenten zurückfallen, unabhängig vom Prozessorgraphen.

Mit den Vorteilen kommen die Kosten. Zwischen Versand und Fertigstellung liegt die Arbeit nun außer Sicht. Zugestellt wird fast immer mindestens einmal; ein Konsument muss deshalb damit umgehen können, dieselbe Nachricht zweimal zu sehen. Die Reihenfolge ist weniger verlässlich, als sie wirkt. Ein eigener Broker ist zudem ein weiterer Server zum Betreiben, Absichern, Überwachen und Aktualisieren.

Wie es in der Praxis aussieht

Auf einer Plattform wurden E-Mail-Versand, Suchindex-Neuaufbau, Dokumenterzeugung und Neuberechnung von Ranglisten aus dem Anfragepfad herausgenommen. Einer Goroutine überlassene Arbeit verschwindet, wenn der Prozess mitten im Deployment neu startet. Deshalb wanderte die Arbeit in eine River-Aufgabenwarteschlange in PostgreSQL. Die Datenbank war bereits das maßgebliche Datensystem. Das Einreihen einer Aufgabe kann so Teil derselben Transaktion wie die betroffenen Daten sein. Die Plattform gewann die Möglichkeit erneuter Ausführungsversuche und einen einsehbaren Verlauf. Die benannte Grenze ist, dass eine Warteschlange in der Datenbank nicht unbegrenzt skaliert.

In einer auf einer C-Bibliothek aufgebauten macOS-Anwendung hielt die Warteschlange die Reihenfolge ein. Die blockierende Ereignisschleife der Bibliothek erzeugt Ereignisse in einer wichtigen Reihenfolge: Eine Bearbeitung muss nach der bearbeiteten Nachricht eintreffen. Eine Aufgabe je Ereignis zu starten erlaubt der Laufzeit, sie beliebig einzuplanen. Eine Bearbeitung kann daher ihre zugehörige Nachricht überholen. Die ausgelieferte Konstruktion hat einen Produzententhread, einen Konsumenten und eine Warteschlange dazwischen, gefolgt von einem einzigen Wechsel auf den Hauptaktor. Die Reihenfolge bleibt erhalten, weil genau ein Weg hindurchführt. Der Preis ist Parallelität: Jedes Ereignis muss durch diesen einen Konsumenten.

Eine Warteschlange kann auch schaden. In einer macOS-Anwendung blieb eine Flut von Logereignissen des Anwendungskerns in einem unbegrenzten Hauptstrom liegen und wurde an Abonnenten kopiert, deren Konsumenten diese Ereignisse nicht nutzten. Diagnoseberichte hielten 111 GB komprimierte Speicherseiten auf einem Rechner mit 36 GB fest. Eine separate Messprobe des Speicherbedarfs zeigte in einer Schleife mit erneuten Verbindungsversuchen ein Wachstum von etwa 41 MB pro Sekunde. Der Ereignispfad erhielt ausdrückliche Grenzen, nach Ereignistyp gefilterte Abonnements und eine ausdrückliche Überlaufregel. Wiederholte Diagnoseereignisse dürfen verworfen und der betroffene Zustand aktualisiert werden, statt unbegrenzt Ereignisse zu behalten. Architekturprüfungen und Tests der Ressourcenbudgets der Anwendung schützen diese Konstruktion. Die Aufzeichnungen belegen aber weder einen gemessenen Speicherwert nach der Korrektur noch einen Produktionsbenchmark.

Wo es schiefgeht

Probleme mit Warteschlangen treten spät und anderswo auf, weil die Arbeit absichtlich außer Sicht liegt.

Der Rückstand wächst, bis Speicher oder Datenträger voll sind. Eine unbegrenzte Warteschlange nimmt Arbeit schneller an, als sie erledigt wird, und behält die Differenz. Geben Sie jeder Warteschlange eine Höchstlänge und legen Sie vorher fest, was dort geschieht: den Produzenten bremsen, neue Arbeit ablehnen oder die älteste verwerfen. Beobachten Sie anschließend zwei Zahlen, die Tiefe und das Alter der ältesten Nachricht. Eine kurze Schlange sehr alter Nachrichten ist ebenso problematisch wie eine lange.

Dieselbe Nachricht wird zweimal verarbeitet. Ein Konsument beendet die Arbeit und stirbt vor der Bestätigung. Der Broker stellt die Nachricht richtigerweise erneut zu. Die RabbitMQ-Dokumentation fordert dazu auf, mit erneuter Zustellung zu rechnen und Konsumenten mit Blick auf Idempotenz zu schreiben. Geben Sie jeder Nachricht eine Kennung und speichern Sie die bereits bearbeiteten Kennungen in derselben Transaktion wie die Arbeit selbst. Eine Notiz über begrenzte Bestände ohne Überverkauf baut ihren Schreibmechanismus auf genau dieser Annahme auf.

Nachrichten kommen in einer anderen Reihenfolge als beim Versand an. Parallele Konsumenten, Vorabruf und Wiederholungen, die eine gescheiterte Nachricht hinter neuere stellen, ordnen Arbeit um. Ist die Reihenfolge wichtig, leiten Sie alles zu einem Konto oder einer Bestellung durch dieselbe Partition oder denselben Konsumenten. Kafka tut dies anhand des Schlüssels, mit dem Preis, dass ein stark beanspruchter Schlüssel nur so schnell vorankommt wie ein Konsument.

Eine fehlerhafte Nachricht blockiert oder läuft endlos im Kreis. Bringt ihre Nutzlast den Konsumenten jedes Mal zum Absturz, wird sie erneut eingereiht, erneut zugestellt und bringt ihn wieder zum Absturz. Begrenzen Sie die Versuche. Verschieben Sie eine Nachricht nach Ausschöpfen aller Versuche an einen gesonderten Ort, an dem ein Mensch sie ansehen kann. RabbitMQ nennt das Dead-Lettering und wendet es auf abgelehnte und abgelaufene Nachrichten sowie auf solche an, die eine Längengrenze überschreiten würden.

Arbeit verschwindet beim Neustart. Ein Kanal, eine Liste im Speicher oder ein Hintergrundthread ist eine Warteschlange, die nichts über einen Neustart hinweg behält. Auch bei AMQP genügt eine dauerhafte Warteschlange allein nicht: Die Nachrichten müssen ebenfalls als persistent veröffentlicht werden. Entscheiden Sie je Arbeitsart, ob Verlust hinnehmbar ist.

Datenbank und Warteschlange widersprechen sich. Ein Dienst schreibt eine Zeile, veröffentlicht danach eine Nachricht und stürzt dazwischen ab. Nun existiert eines ohne das andere. Halten Sie entweder die Warteschlange in derselben Datenbank, sodass Worker eine Aufgabe erst nach dem Commit ihrer Transaktion sehen, wie bei River, oder schreiben Sie die Nachricht in dieser Transaktion in eine Outbox-Tabelle und veröffentlichen Sie sie von dort.

Weiterführende Lektüre

- Das AMQP-0-9-1-Modell erklärt — Exchanges, Warteschlangen, Bindungen und Bestätigungen, dargestellt von RabbitMQ.
- Konsumentenbestätigungen und Publisherbestätigungen — wann der Broker eine Nachricht als zugestellt betrachtet und warum sie erneut zugestellt wird.
- Dead Letter Exchanges — unter welchen Bedingungen eine Nachricht beiseitegelegt wird und wohin sie gelangt.
- Die Sperrklausel von SELECT — `FOR UPDATE` und `SKIP LOCKED` sowie die dafür geeigneten warteschlangenartigen Tabellen.
- Erste Schritte mit River — eine Aufgabenwarteschlange, die ihre Aufgaben in der eigenen Datenbank der Anwendung speichert.
- Einführung in Apache Kafka — Topics, Partitionen und Aufbewahrung im Logmodell.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Bringen Sie einen repräsentativen Datensatz, die Fragen, die er beantworten muss, und den aktuellen Engpass mit. Wir können vereinbaren, wie Qualität, Durchsatz und Wiederherstellung gemessen werden.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist eine Warteschlange?“ 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/queue/