Engineer

Kö

arbete i ordningsföljd som något annat ska plocka upp

Vad det är

En kö är ett mellanlager mellan två program: den ena sidan lägger in meddelanden eller jobb, den andra tar ut dem när den är redo, oftast det äldsta först. De flesta system kör flera köer, en per sorts arbete. Programmet som lägger till arbete är producenten, programmet som tar det är konsumenten, och det som lagrar meddelandena däremellan är brokern, som kan vara en egen server eller en tabell i en databas som systemet redan har.

Upplägget gör att de två sidorna slutar vänta på varandra. En webbförfrågan som behöver skicka ett mejl lägger in ett meddelande och svarar användaren direkt. En worker plockar upp meddelandet en stund senare, och om hundra kommer på en gång betar den av dem i sin egen takt. En topp blir en eftersläpning, och ett beroende blir en fördröjning.

Två idéer beskriver vad en kö lovar om leveransen. Den första är kvittering: en konsument talar om för brokern när den är klar med ett meddelande, och först då tas meddelandet bort. I modellen för AMQP 0-9-1, som RabbitMQ implementerar, går ett meddelande vars konsument dör innan den hinner kvittera vidare till en annan konsument. Den andra är beständighet: en beständig kö med beständiga meddelanden överlever att brokern startas om, medan en kö som bara hålls i minnet går förlorad.

Tre närliggande begrepp är värda att hålla isär. En meddelandekö bär data mellan tjänster. En jobbkö bär arbetsenheter åt workers i bakgrunden, var och en med sina argument, ett tillstånd och ett antal försök. En logg, modellen bakom Apache Kafka, ser ut som en kö utifrån men behåller händelserna efter att de har lästs, under en lagringstid som väljs per ämne, så att flera läsare kan spela upp samma historik igen. Att läsa från en kö tar bort meddelandet, medan läsning från en logg bara flyttar läsarens position framåt.

En kö behöver inte alls vara en separat produkt. PostgreSQL kan spela rollen med en tabell med jobb och en databasfråga som gör anspråk på nästa rad och kliver förbi rader som en annan worker redan har låst. Handboken anger köliknande tabeller som själva skälet till att satsdelen `SKIP LOCKED` finns:

```
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;
```

Använd begreppet i ett beslut

Använd ”Vad är en kö?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Siffror som kommer fram dit de behövs, på fasta tider, utan att någon kopierar ett kalkylblad i slutet av månaden.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. En infrastrukturplan för 14 avdelningar — 14 forskningsavdelningar under en och samma infrastrukturplan

En infrastrukturplan för 14 avdelningar Hör av er →

Varför det spelar roll

En kö avgör vad en användare väntar på. Arbete som läggs i kön sker efter att svaret har gått iväg, så en sida som utlöser ett mejl, en PDF, en omindexering för sökning och en omräkning kan svara på den tid det tar att skriva fyra rader, medan de fyra långsamma uppgifterna blir klara bakom den. För en köpare är det skillnaden mellan en kassa som svarar på några hundra millisekunder och en som svarar när e-postservern råkar vara redo.

Kön avgör också vad som överlever ett fel. Arbete som startas i en bakgrundstråd lever i den processens minne, och en driftsättning som startar om processen tar med sig arbetet utan att lämna något spår av att det någonsin skulle utföras.

Arbete i en beständig kö är i stället en post: det går att försöka igen, inspektera och räkna.

Den skiljer dessutom på två hastigheter. Producenterna rör sig i takt med den inkommande trafiken, konsumenterna i takt med den långsammaste tjänst de anropar. En kö mellan dem tar upp skillnaden, och dess djup blir ett av de mest sanningsenliga tal som ett system kan rapportera. En kö som har vuxit i en timme säger att konsumenterna tappar mark, vad grafen över processorn än antyder.

Kostnaderna kommer tillsammans med fördelarna. Arbetet ligger nu utom synhåll mellan att det skickas och att det blir gjort. Leveransen är nästan alltid minst en gång, så en konsument måste tåla att se ett meddelande två gånger. Ordningen är svagare än den ser ut. En dedikerad broker är dessutom ännu en server att driva, säkra, bevaka och uppgradera.

Så ser det ut i praktiken

Det som gör en kö värd sin plats skiljer sig från fall till fall: här är det att jobb överlever en omstart, att händelser behåller sin ordning och att eftersläpningen har ett tak.

På en plattform flyttades det långsamma arbetet bort från anropsvägen: att skicka e-post, bygga om ett sökindex, generera dokument och räkna om placeringar. Arbete som lämnas till en goroutine försvinner när processen startas om mitt under en driftsättning, så arbetet flyttades till en jobbkö i River som ligger i PostgreSQL. Databasen var redan den auktoritativa källan, vilket gör att ett jobb läggs i kön i samma transaktion som den data det gäller. Plattformen fick nya försök och en historik som någon kan öppna och läsa. Den uttalade gränsen är att en kö inne i databasen inte skalar i all evighet.

I en app för macOS som byggdes ovanpå ett bibliotek i C fanns kön där för att hålla ordning. Bibliotekets blockerande händelseloop skickar händelser i en ordning som spelar roll, eftersom en redigering måste komma efter det meddelande den redigerar. Att starta en task per händelse låter körmiljön schemalägga dem i godtycklig ordning, så att en redigering kan gå om meddelandet den gäller. Designen som levererades har en producenttråd, en konsument och en kö mellan dem, följd av ett enda hopp över till main actor. Ordningen håller eftersom det finns exakt en väg igenom. Priset är parallellismen: varje händelse passerar genom den enda konsumenten.

En kö kan också göra skada. I en macOS-applikation behölls en skur av core-logghändelser i en obegränsad huvudström och kopierades till prenumeranter vars konsumenter inte använde de händelserna. Diagnostikrapporter registrerade 111 GB komprimerade sidor på en maskin med 36 GB; ett separat stickprov av minnesförbrukningen registrerade en ökning med ungefär 41 MB per sekund under en återförsöksslinga för anslutningen. Händelsevägen fick uttryckliga gränser, med prenumerationer filtrerade efter händelsetyp och en uttrycklig policy för buffertöverflöde. Upprepade diagnostiska händelser kan kastas bort och berört tillstånd läsas in på nytt i stället för att behålla ett obegränsat antal händelser. Arkitekturkontroller och tester av applikationens budgetar skyddar den designen, men beskrivningen fastställer inte någon uppmätt minnesförbrukning efter korrigeringen eller något prestandatest i produktion.

Var det går fel

Problem med en kö märks sent och någon annanstans, eftersom arbetet ligger utom synhåll per konstruktion.

Eftersläpningen växer tills minnet eller disken tar slut. En kö utan gräns tar emot arbete snabbare än det blir gjort och behåller skillnaden. Ge varje kö en största längd och bestäm i förväg vad som händer vid den längden: bromsa producenten, vägra nytt arbete eller släpp det äldsta. Bevaka sedan två tal, djupet och åldern på det äldsta meddelandet, eftersom en kort kö med mycket gamla meddelanden är lika illa som en lång.

Samma meddelande behandlas två gånger. En konsument gör klart arbetet och dör innan den kvitterar, och brokern levererar, helt korrekt, meddelandet igen. Dokumentationen till RabbitMQ säger åt konsumenter att räkna med nya leveranser och att skrivas med idempotens i åtanke. Ge varje meddelande en identifierare och registrera de identifierare som redan har hanterats, i samma transaktion som själva arbetet. Skrivaren i anteckningen om en blixtrea på en enda het rad är byggd kring precis det antagandet.

Meddelanden kommer i en annan ordning än de skickades i. Parallella konsumenter, förhämtning och nya försök som lägger ett misslyckat meddelande bakom nyare meddelanden kastar alla om arbetet. När ordningen spelar roll, skicka allt som rör ett konto eller en order genom samma partition eller samma konsument. Kafka gör det per nyckel, till priset att en upptagen nyckel bara rör sig lika snabbt som en enda konsument.

Ett dåligt meddelande blockerar eller går i cirklar för alltid. Ett meddelande vars innehåll kraschar konsumenten varje gång läggs tillbaka i kön, levereras igen och kraschar den igen. Sätt ett tak för antalet försök och flytta ett meddelande som har förbrukat dem till en separat plats där en människa kan titta på det. RabbitMQ kallar det dead-lettering och tillämpar det på meddelanden som har avvisats, som har gått ut och som skulle få kön att överskrida en gräns för längden.

Arbete försvinner vid en omstart. En kanal, en lista i minnet eller en bakgrundstråd är en kö som inte behåller något över en omstart. I AMQP räcker inte heller en beständig kö ensam: meddelandena måste också publiceras som beständiga. Bestäm för varje sorts arbete om det är acceptabelt att förlora det.

Databasen och kön är oense. En tjänst skriver en rad, publicerar sedan ett meddelande och kraschar mellan de två, och nu finns det ena utan det andra. Håll antingen kön i samma databas, så att ett jobb blir synligt för workers först när dess transaktion har committats, vilket är så River fungerar, eller skriv meddelandet till en utkorgstabell i den transaktionen och publicera det därifrån.

Läs mer

- AMQP 0-9-1 Model Explained — exchanges, köer, bindningar och kvitteringar, så som RabbitMQ beskriver dem.
- Consumer Acknowledgements and Publisher Confirms — när en broker betraktar ett meddelande som levererat och varför det levereras igen.
- Dead Letter Exchanges — villkoren som flyttar undan ett meddelande och vart det tar vägen.
- The Locking Clause of SELECT — `FOR UPDATE` och `SKIP LOCKED` och de köliknande tabeller de passar för.
- River: Getting started — en jobbkö som lagrar sina jobb i applikationens egen databas.
- Apache Kafka: Introduction — ämnen, partitioner och lagringstid i loggmodellen.



Ordlista

Välj utifrån era faktiska förutsättningar

Ta med ett representativt dataset, frågorna det ska besvara och dagens flaskhals. Vi kan avtala mått på kvalitet, genomströmning och återställning.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är en kö?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/queue/