Engineer

Kø

arbejde lagt i rækkefølge, så noget andet kan tage det

Hvad det er

En kø er et mellemlager mellem to programmer: den ene side lægger beskeder eller jobs ind, den anden tager dem ud, når den er klar, som regel de ældste først. Næsten ethvert system har flere køer, én pr. slags arbejde, og i køen ligger det arbejde, der endnu venter på at blive taget. Programmet, der lægger arbejde ind, kaldes producenten, programmet, der tager det, kaldes forbrugeren, og det, der opbevarer beskederne imellem, kaldes en broker. Brokeren kan være en selvstændig server eller en tabel i en database, systemet allerede har.

Pointen er, at de to sider holder op med at vente på hinanden. En webrequest, der skal have sendt en mail, lægger en besked i køen og svarer brugeren med det samme; en worker henter beskeden et øjeblik senere, og kommer der hundrede på én gang, arbejder den sig igennem dem i sit eget tempo. Et spidsbelastningsøjeblik bliver til et efterslæb, og en afhængighed bliver til en forsinkelse.

To begreber beskriver, hvad en kø lover om levering. Det første er kvittering: forbrugeren melder tilbage til brokeren, når den er færdig med en besked, og først da fjernes beskeden. I AMQP 0-9-1-modellen, som RabbitMQ bygger på, går en besked videre til en anden forbruger, hvis den første dør uden at kvittere. Det andet er holdbarhed: en holdbar kø med persistente beskeder overlever, at brokeren genstarter, mens en kø, der kun findes i hukommelsen, glemmer alt.

Tre beslægtede begreber er værd at holde adskilt. En beskedkø bærer data mellem tjenester. En jobkø bærer arbejdsopgaver til baggrunds-workers, hver med sine argumenter, en tilstand og et antal forsøg. En log, den model Apache Kafka bygger på, ligner en kø udefra, men beholder hændelserne, efter at de er

læst, i en opbevaringsperiode, der vælges pr. topic, så flere læsere kan gennemgå den samme historik. At læse fra en kø fjerner beskeden; at læse fra en log flytter kun læserens position.

En kø behøver ikke være et selvstændigt produkt. PostgreSQL kan løfte opgaven med en tabel af jobs og en SQL-forespørgsel, der tager den næste række og springer over rækker, en anden worker allerede har låst. Manualen nævner netop kølignende tabeller som grunden til, at `SKIP LOCKED` findes:

```
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;
```

Brug begrebet i en beslutning

Brug »Hvad er en kø?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Tal, der lander, hvor der er brug for dem, efter en fast plan, uden at nogen kopierer et regneark ved månedens slutning.

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. Én infrastrukturplan til 14 afdelinger — 14 forskningsafdelinger på én infrastrukturplan

Én infrastrukturplan til 14 afdelinger Kontakt os →

Hvorfor det betyder noget

Køen afgør, hvad brugeren venter på. Arbejde, der lægges i kø, sker efter at svaret er sendt, så en side, der udløser en mail, en PDF, en genopbygning af et søgeindeks og en genberegning, kan svare på den tid, det tager at skrive fire rækker. De fire langsomme opgaver bliver færdige bagefter. For en køber er det forskellen på et checkout, der svarer på få hundrede millisekunder, og et, der svarer, når mailserveren har tid.

Køen afgør også, hvad der overlever en fejl. Arbejde, der er startet i en baggrundstråd, findes kun i processens hukommelse, og en udrulning, der genstarter processen, tager arbejdet med sig uden at efterlade spor. Arbejde i en holdbar kø er derimod en optegnelse: det kan prøves igen, undersøges og tælles.

Og køen adskiller to hastigheder. Producenter bevæger sig i takt med den indkommende trafik, forbrugere i takt med den langsomste tjeneste, de kalder. Køen imellem dem opsuger forskellen, og dens dybde bliver et af de ærligste tal, et system kan melde. En kø, der er vokset i en time, fortæller, at forbrugerne sakker bagud, uanset hvad processorgrafen viser.

Fordelene har en pris. Arbejdet ligger nu uden for synsfeltet mellem at blive sendt og at blive gjort. Levering sker næsten altid mindst én gang, så en forbruger skal kunne tåle at se den samme besked to gange. Rækkefølgen er svagere, end den ser ud. Og en selvstændig broker er endnu en server, der skal drives, sikres, overvåges og opgraderes.

Sådan ser det ud i praksis

Mail, et søgeindeks, der skal genopbygges, dokumenter og genberegnede placeringer er alt sammen arbejde, en bruger ikke bør vente på, og én platform flyttede det væk fra request-stien. Arbejde, der gives til en goroutine, forsvinder, når processen genstarter midt i en deploy, så arbejdet kom over i en River-jobkø i PostgreSQL. Databasen var i forvejen det autoritative register, så at lægge et job i kø bliver en del af den samme transaktion som de data, jobbet handler om. Platformen fik nye forsøg og en historik, man kan slå op i; den erklærede grænse er, at en kø i databasen ikke skalerer i det uendelige.

I et macOS-program bygget oven på et C-bibliotek var køen der for at holde rækkefølgen. Bibliotekets blokerende hændelsesløkke sender hændelser i en rækkefølge, der betyder noget, for en redigering skal ankomme efter den besked, den redigerer. At starte én opgave pr. hændelse lader køretiden planlægge dem i vilkårlig rækkefølge, så en redigering kan overhale sin egen besked. Den løsning, der blev valgt, har én producenttråd, én forbruger og én kø imellem dem og derefter et enkelt hop over på hovedaktøren. Rækkefølgen holder, fordi der kun er én vej. Prisen er parallelitet: alle hændelser går gennem den ene forbruger.

En kø kan også gøre skade. I en macOS-applikation blev en byge af core-loghændelser beholdt i en ubegrænset hovedstrøm og kopieret til abonnenter, hvis forbrugere ikke brugte de hændelser. Diagnoserapporter registrerede 111 GB komprimerede sider på en maskine med 36 GB; en separat måling af hukommelsesforbruget registrerede en vækst på omkring 41 MB i sekundet under en løkke med gentagne forbindelsesforsøg. Hændelsesvejen fik udtrykkelige grænser, med abonnementer filtreret efter hændelsestype og en udtrykkelig politik for bufferoverløb. Gentagne diagnostiske hændelser kan kasseres, og den berørte tilstand genindlæses i stedet for at beholde ubegrænset mange hændelser. Arkitekturkontroller og tests af applikationens budgetter understøtter det design, men beskrivelsen fastslår ikke et målt hukommelsesforbrug efter rettelsen eller et benchmark fra produktion.

Hvor det går galt

Problemer med køer viser sig sent og et andet sted, fordi arbejdet med vilje ligger uden for synsfeltet.

Efterslæbet vokser, til hukommelse eller disk slipper op. En kø uden grænse tager imod arbejde hurtigere, end det bliver gjort, og gemmer forskellen. Giv hver kø en største længde, og beslut på forhånd, hvad der sker ved den: brems producenten, afvis nyt arbejde, eller smid det ældste væk. Hold derefter øje med to tal, dybden og alderen på den ældste besked, for en kort kø af meget gamle beskeder er lige så slem som en lang.

Den samme besked behandles to gange. En forbruger gør arbejdet færdigt og dør, før den kvitterer; brokeren leverer, helt korrekt, beskeden igen. RabbitMQ's dokumentation beder forbrugere om at regne med genlevering og om at være skrevet med idempotens for øje. Giv hver besked et id, og registrér de id'er, der allerede er behandlet, i samme transaktion som selve arbejdet. En note på dette site om at sælge et begrænset lager uden at oversælge bygger sin skriver på præcis den antagelse.

Beskederne ankommer i en anden rækkefølge, end de blev sendt i. Parallelle forbrugere, forudhentning og nye forsøg, der stiller en fejlet besked bag nyere, bytter alle om på arbejdet. Når rækkefølgen betyder noget, så send alt om én konto eller én ordre gennem den samme partition eller den samme forbruger. Kafka gør det efter nøgle, til den pris, at én travl nøgle kun går så hurtigt som én forbruger.

Én dårlig besked blokerer eller kører i ring. En besked, hvis indhold får forbrugeren til at gå ned hver gang, lægges tilbage, leveres igen og får den til at gå ned igen. Sæt loft over antallet af forsøg, og flyt en besked, der har brugt dem alle, over et separat sted, hvor et menneske kan se på den. RabbitMQ kalder det dead-lettering og bruger det på afviste beskeder, udløbne beskeder og beskeder, der ville sprænge en længdegrænse.

Arbejdet forsvinder ved genstart. En kanal, en liste i hukommelsen eller en baggrundstråd er en kø, der intet beholder hen over en genstart. I AMQP er en holdbar kø heller ikke nok alene: beskederne skal også sendes som persistente. Afgør for hver slags arbejde, om det er acceptabelt at miste det.

Databasen og køen er uenige. En tjeneste skriver en række, sender derefter en besked og går ned mellem de to; nu findes det ene uden det andet. Hold enten køen i den samme database, så et job først bliver synligt for workers, når dets transaktion committer, sådan som River gør, eller skriv beskeden i en outbox-tabel i samme transaktion og send den derfra.

Læs mere

- AMQP 0-9-1 Model Explained — exchanges, køer, bindinger og kvitteringer, forklaret af RabbitMQ.
- Consumer Acknowledgements and Publisher Confirms — hvornår en broker anser en besked for leveret, og hvorfor den leveres igen.
- Dead Letter Exchanges — de betingelser, der flytter en besked til side, og hvor den havner.
- The Locking Clause of SELECT — `FOR UPDATE` og `SKIP LOCKED` og de kølignende tabeller, de passer til.
- River: Getting started — en jobkø, der gemmer sine jobs i programmets egen database.
- Apache Kafka: Introduction — topics, partitioner og opbevaring i log-modellen.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring et repræsentativt datasæt, de spørgsmål det skal besvare, og den aktuelle flaskehals. Vi kan aftale målinger af kvalitet, gennemløb og gendannelse.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er en kø?« 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/queue/