Engineer

Kø

arbeid lagt i rekkefølge for at noe annet skal hente det

Hva det er

En kø er et oppbevaringssted mellom to programmer: den ene siden legger inn meldinger eller jobber, den andre tar dem ut når den er klar, som regel de eldste først. De fleste systemer har flere køer, én per type arbeid. Programmet som legger inn arbeid, er produsenten, programmet som tar det, er konsumenten, og det som lagrer meldingene imellom, er megleren (broker), som kan være en egen server eller en tabell i en database systemet allerede har.

Ordningen gjør at de to sidene slutter å vente på hverandre. En nettforespørsel som trenger å sende en e-post, legger inn en melding og svarer brukeren med en gang; en arbeider (worker) henter meldingen noen øyeblikk senere, og hvis hundre kommer samtidig, tømmer den dem i sitt eget tempo. En topp blir et etterslep, og en avhengighet blir en forsinkelse.

To begreper beskriver hva en kø lover om levering. Det første er kvittering (acknowledgement): en konsument sier fra til megleren når den er ferdig med en melding, og først da fjernes meldingen. I modellen AMQP 0-9-1, som RabbitMQ implementerer, går en melding videre til en annen konsument hvis konsumenten dør før den har kvittert. Det andre er varighet (durability): en varig kø med persistente meldinger overlever en omstart av megleren, og en kø som bare holdes i minnet, gjør det ikke.

Tre nabobegreper er verdt å holde fra hverandre. En meldingskø frakter data mellom tjenester. En jobbkø frakter arbeidsoppgaver til bakgrunnsarbeidere, hver med argumentene sine, en tilstand og et antall forsøk. En logg, modellen bak Apache Kafka, ser ut som en kø utenfra, men beholder hendelsene etter at de er lest, i en oppbevaringsperiode som velges per topic, slik at flere lesere kan spille av den samme historikken. Å lese fra en kø fjerner meldingen; å lese fra en logg flytter bare leserens posisjon.

En kø trenger ikke være et eget produkt i det hele tatt. PostgreSQL kan spille rollen med en tabell med jobber og en spørring som gjør krav på neste rad og hopper over rader en annen arbeider allerede har låst. Håndboken nevner kø-lignende tabeller som grunnen til at leddet `SKIP LOCKED` finnes:

```
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;
```

Bruk begrepet i en beslutning

Bruk «Hva er en kø?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Tall som kommer fram dit de trengs, til faste tider, uten at noen må kopiere et regneark ved månedsslutt.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Én infrastrukturplan for 14 avdelinger — 14 forskningsavdelinger på én felles infrastrukturplan

Én infrastrukturplan for 14 avdelinger Ta kontakt →

Hvorfor det er viktig

En kø avgjør hva en bruker venter på. Arbeid som legges i en kø, skjer etter at svaret er sendt, så en side som utløser en e-post, en PDF, en ny indeksering for søk og en omberegning, kan svare på den tiden det tar å skrive fire rader, og de fire trege oppgavene blir ferdige bak den. For en kjøper er det forskjellen på en utsjekking som svarer på noen hundre millisekunder, og en som svarer når e-postserveren er klar.

Den avgjør også hva som overlever en feil. Arbeid som startes i en bakgrunnstråd, lever i minnet til den prosessen, og en produksjonssetting som starter prosessen på nytt, tar arbeidet med seg og etterlater ikke noe spor av at

det noen gang skulle gjøres. Arbeid i en varig kø er i stedet en post: det kan prøves på nytt, inspiseres og telles.

Den skiller også to hastigheter. Produsenter beveger seg i takt med trafikken som kommer inn; konsumenter i takt med den tregeste tjenesten de kaller. En kø mellom dem tar opp forskjellen, og dybden i køen blir et av de mest ærlige tallene et system kan rapportere. En kø som har vokst i en time, sier at konsumentene taper terreng, uansett hva grafen for prosessoren antyder.

Kostnadene kommer sammen med fordelene. Arbeidet er nå ute av syne fra det sendes til det er utført. Levering skjer nesten alltid minst én gang, så en konsument må tåle å se en melding to ganger. Rekkefølgen er svakere enn den ser ut. Og en egen megler er én server til som skal driftes, sikres, overvåkes og oppgraderes.

Slik ser det ut i praksis

Det trege arbeidet på én plattform ble flyttet ut av forespørselsløpet: å sende e-post, bygge en søkeindeks på nytt, generere dokumenter, beregne rangeringer på nytt. Arbeid som overlates til en goroutine, forsvinner når prosessen starter på nytt midt i en produksjonssetting, så arbeidet ble flyttet over i en jobbkø i River som holdes i PostgreSQL. Databasen var allerede det autoritative kildesystemet, og det gjør det å legge en jobb i køen til en del av den samme transaksjonen som dataene jobben gjelder. Det plattformen fikk, er nye forsøk og en historikk noen kan åpne og lese; begrensningen siden selv oppgir, er at en kø inne i databasen ikke skalerer i det uendelige.

I en macOS-app bygd over et C-bibliotek var køen der for å holde rekkefølgen. Den blokkerende hendelsesløkken i biblioteket sender ut hendelser i en rekkefølge som betyr noe, fordi en redigering må komme etter meldingen den redigerer. Å starte én oppgave per hendelse lar kjøretiden planlegge dem i vilkårlig rekkefølge, så en redigering kan gå forbi meldingen den viser til. Designet som ble levert, har én produsenttråd, én konsument og én kø mellom dem, fulgt av ett enkelt hopp over til main actor. Rekkefølgen holder fordi det finnes nøyaktig én vei gjennom. Prisen er parallellitet: hver hendelse går gjennom den ene konsumenten.

En kø kan også gjøre skade. I én macOS-applikasjon ble en bølge av core-logghendelser beholdt i en ubegrenset hovedstrøm og kopiert til abonnenter der konsumentene ikke brukte disse hendelsene. Diagnoserapporter viste 111

GB komprimerte sider på en maskin med 36 GB; en separat stikkprøvemåling av minnebruken viste en vekst på omtrent 41 MB per sekund under en løkke med gjentatte tilkoblingsforsøk. Hendelsesbehandlingen fikk uttrykkelige grenser, med abonnementer filtrert etter hendelsestype og en uttrykkelig regel for overfylte buffere. Gjentatte diagnosehendelser kan forkastes, og berørt tilstand kan oppdateres i stedet for å beholde et ubegrenset antall hendelser. Arkitektursjekker og tester av applikasjonens ressursbudsjett verner om denne utformingen, men dokumentasjonen fastslår ikke en målt verdi for minnebruk etter rettingen eller en ytelsesmåling i produksjon.

Hvor det går galt

Trøbbel med køer viser seg sent og et annet sted, fordi arbeidet er ute av syne med vilje.

Etterslepet vokser til minnet eller disken tar slutt. En kø uten grense tar imot arbeid raskere enn det blir gjort, og beholder differansen. Gi hver kø en maksimal lengde, og bestem på forhånd hva som skjer ved den lengden: brems produsenten, avvis nytt arbeid eller kast det eldste. Følg så med på to tall, dybden og alderen på den eldste meldingen, fordi en kort kø med svært gamle meldinger er like ille som en lang.

Den samme meldingen behandles to ganger. En konsument blir ferdig med arbeidet og dør før den kvitterer; megleren leverer, helt korrekt, meldingen på nytt. Dokumentasjonen til RabbitMQ ber konsumenter regne med nye leveringer og være skrevet slik at de er idempotente. Gi hver melding en identifikator, og registrer identifikatorene som allerede er behandlet, i den samme transaksjonen som selve arbeidet. Skriveren i notatet om lynsalg uten oversalg er bygd rundt akkurat den antakelsen.

Meldinger kommer i en annen rekkefølge enn de ble sendt i. Parallelle konsumenter, forhåndshenting (prefetch) og nye forsøk som legger en feilet melding bak nyere meldinger, stokker alle om på arbeidet. Når rekkefølgen betyr noe, send alt som gjelder én konto eller én ordre, gjennom den samme partisjonen eller den samme konsumenten. Kafka gjør dette etter nøkkel, med den kostnaden at én travel nøkkel bare beveger seg så raskt som én konsument.

Én dårlig melding blokkerer eller går i løkke for alltid. En melding med et innhold som krasjer konsumenten hver gang, legges tilbake i køen, leveres på nytt og krasjer den igjen. Sett et tak på forsøkene, og flytt en melding som har

brukt opp alle, til et eget sted der en person kan se på den. RabbitMQ kaller dette dead-lettering og bruker det på avviste meldinger, utløpte meldinger og meldinger som ville gått over en lengdegrense.

Arbeid forsvinner ved omstart. En kanal, en liste i minnet eller en bakgrunnstråd er en kø som mister alt ved en omstart. I AMQP holder heller ikke en varig kø alene: meldingene må også publiseres som persistente. Avgjør for hver type arbeid om det er akseptabelt å miste det.

Databasen og køen er uenige. En tjeneste skriver en rad, publiserer så en melding og krasjer mellom de to; nå finnes det ene uten det andre. Enten holder du køen i den samme databasen, så en jobb blir synlig for arbeiderne først når transaksjonen dens committes, slik River oppfører seg, eller du skriver meldingen inn i en outbox-tabell i den transaksjonen og publiserer den derfra.

Les mer

- AMQP 0-9-1 Model Explained — exchanger, køer, bindinger og kvitteringer, slik RabbitMQ beskriver dem.
- Consumer Acknowledgements and Publisher Confirms — når en megler regner en melding som levert, og hvorfor ny levering skjer.
- Dead Letter Exchanges — betingelsene som flytter en melding til side, og hvor den havner.
- The Locking Clause of SELECT — `FOR UPDATE` og `SKIP LOCKED`, og de kø-lignende tabellene de passer til.
- River: Getting started — en jobbkø som lagrer jobbene sine i applikasjonens egen database.
- Apache Kafka: Introduction — topics, partisjoner og oppbevaring i loggmodellen.

Ordliste

Velg ut fra deres faktiske rammer

Ta med et representativt datasett, spørsmålene det skal besvare og dagens flaskehals. Vi kan avtale målinger av kvalitet, gjennomstrømning og gjenoppretting.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er en kø?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/queue/