Engineer

Skalerbarhed

📈 om mere belastning kræver andet end flere maskiner

Hvad det er

Skalerbarhed er den måde, et systems kapacitet vokser på, når der tilføjes ressourcer. Et system skalerer godt, når dobbelt så mange maskiner bærer omtrent dobbelt så meget belastning med de samme svartider, og dårligt, når den ekstra hardware mest venter på noget, den ikke kan hjælpe med. Ordet beskriver et forhold mellem belastning og omkostning, ikke en egenskab, et system enten har eller mangler.

Der er to retninger at vokse i. At skalere op, vertikal skalering, betyder en større maskine: flere kerner, mere hukommelse, hurtigere diske. Det kræver ingen ændring af softwaren og stopper ved den største maskine, det kan betale sig at købe. At skalere ud, horisontal skalering, betyder flere maskiner, der deler arbejdet. Her er der ikke samme loft, men kun arbejde, der kan deles op, får gavn af det.

HTTP er formet med udskalering for øje. RFC 9110 bemærker, at mange implementeringer bygger på HTTP's tilstandsløse design for at genbruge forbindelser gennem proxyer eller fordele forespørgsler dynamisk på flere servere. Et weblag, der ikke gemmer noget mellem forespørgslerne, kan vokse ved at sætte flere kopier af sig selv bag en load balancer, og enhver kopi kan besvare enhver forespørgsel.

Belastning har mere end én dimension, og et system kan skalere i én og gå i stå i en anden: forespørgsler i sekundet, mængden af lagrede data, antallet af lejere og antallet af skrivere, der kæmper om den samme post. Den sidste er den sværeste, fordi en enkelt række kun kan ændres af én transaktion ad gangen, uanset hvor mange servere der står omkring den.

Regnestykket bag det er Amdahls lov. Når en del af arbejdet skal køre i rækkefølge, gør parallel hardware kun resten hurtigere, så den sekventielle del sætter grænsen: er 5 % af arbejdet serielt, kan hele jobbet aldrig køre mere end 20 gange hurtigere, heller ikke på tusind maskiner. I en database er den serielle del ofte en lås. I en applikation er den ofte én delt tæller, én kø eller ét sessionslager.

Brug begrebet i en beslutning

Brug »Hvad er skalerbarhed?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Overvågning og alarmer, der siger til, mens fejlen stadig er lille, og tallene, der viser, hvor tit den bliver ved med at være det.

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. Har bygget det første observerbarhedsdashboard — virksomhedens første overblik over sine egne systemer

Har bygget det første observerbarhedsdashboard Kontakt os →

Hvorfor det betyder noget

Skalerbarheden afgør, hvad vækst koster. Når et system skalerer godt, er en fordobling af trafikken en linje på en faktura. Når det skalerer dårligt, vokser regningen hurtigere end belastningen, eller mere hardware holder helt op med at hjælpe, og løsningen bliver et redesign, der skal udføres, mens trafikken strømmer ind.

Hardware kan afhjælpe flaskehalse i processorer, hukommelse eller diske, men ineffektivt arbejde og krav om sekventiel udførelse kan bestå. En forespørgsel, der unødigt scanner en hel tabel, kan stadig udføre overflødigt arbejde på en større server. En større maskine fjerner ikke kravet om, at opdateringer af den samme række skal udføres én ad gangen, og flere applikationsservere kan øge antallet af skrivende klienter i køen. De fleste systemer, der ser ud til at have brug for flere maskiner, skal læses først.

I PostgreSQL er stedet at begynde at læse `pg_stat_statements`, en udvidelse, der registrerer hver normaliseret sætning med antal kald og udførelsestid. Sorteret efter samlet tid frem for gennemsnit viser den, hvor serveren faktisk bruger sig selv: en forespørgsel, der tager 2 ms og kører en million gange i timen, koster 2.000 sekunder af den time, mens en rapport på to sekunder, der kører to gange, koster fire.

```
SELECT calls, round(total_exec_time) AS total_ms,
       left(query, 60) AS query
  FROM pg_stat_statements
 ORDER BY total_exec_time DESC
 LIMIT 10;
```

Hvert skridt mod flere maskiner har sine egne driftsomkostninger, og derfor betyder rækkefølgen noget. En strammere sætning eller et bedre indeks kan forbedre ydeevnen uden at tilføje endnu en server. En læsereplika er konfiguration. Partitionering ændrer én tabel. Sharding, hvor data deles ud på separate databaseservere, ændrer driften omkring dem: migreringer, backup, gendannelser og rapporter skal tage højde for separate shards; en joinoperation på tværs af shards kan flyttes til applikationskoden; og en forespørgsel, der ikke kender sin shardnøgle, kan være nødt til at spørge dem alle.

Sådan ser det ud i praksis

En PostgreSQL-database bag en mellemstor applikation rummede brugerkonti, lejere, geografiske data og dagligt elforbrug. Den kørte under lav belastning, men blev langsommere, efterhånden som datamængderne voksede. Opgaven var at gøre den ti gange hurtigere uden at omlægge arkitekturen, med plads til fremtidig datavækst som en del af målet. Arbejdet var GiST-indeks til de geografiske data, sammensatte indeks på lejer og datointerval, eltabellen partitioneret efter tid, hash-partitionering af de geografiske data, omskrevne forespørgsler tjekket med `EXPLAIN ANALYZE` og connection pooling. Svartiderne blev ti gange bedre, databasens ressourceforbrug faldt med 40 %, og systemet kunne tage imod voksende datamængder uden ydeevnetab og med mindre behov for hardwareopgraderinger.

I en mellemstor virksomhed med ti PostgreSQL- og SQL Server-databaseservere blev infrastrukturen skaleret til 40 % mere brugertrafik uden servicetab. Samtidig blev svartiderne for forespørgsler 30 % bedre gennem konfigurationstuning og indeksoptimering.

Rækkefølgen, arbejdet følger, er beskrevet i en note om at optimere før sharding: kig, stram op, tag læsningerne væk fra den primære, partitionér tabellerne omkring det varme punkt, og shard først, når én maskine er fuld af godt arbejde. Noten nævner tre tilfælde, hvor sharding er rimeligt: den primære er mættet af sætninger, der allerede er slanke, data er vokset fra deres vedligeholdelse, eller isolation er selve kravet.

Hvor det går galt

Flere webservere foran én database. Hver applikationsserver åbner sine egne forbindelser, og når mange gange flere sessioner er aktive, end databasen har kerner, bruger serveren sin tid på at skifte mellem dem. En forbindelsespulje løser det, før nogen hardware gør, og endnu en webserver gør det værre.

Én varm række. Når hver forespørgsel skriver til den samme post, for eksempel en lagertæller eller en saldo, er gennemløbet begrænset til omtrent de skrivninger, én lås kan bære, divideret med den tid, låsen holdes, uanset antallet af maskiner. Alt langsomt inde i låsen sænker loftet. Hold transaktionen kort, hold eksterne kald uden for den, og når én vare vokser fra sin maskine, så del dens lager op i mindre stykker, der hver har præcis én ejer.

Tilstand, der bor på én server. Sessioner i én proces' hukommelse, uploads skrevet til dens lokale disk eller et planlagt job, der går ud fra, at det er den eneste kopi, forhindrer alle, at en anden instans kan være ligeværdig. Læg den tilstand i et fælles lager, så enhver instans kan besvare enhver forespørgsel.

Replikaer behandlet som kopier af den primære. En streaming-replika ligger lidt bagefter, så en bruger, der lige har skrevet noget og derefter læser fra replikaen, ser det måske ikke; læs nylige skrivninger fra den primære. En lang rapport på en standby kan også blive annulleret, når den holder de ændringer tilbage, der afspilles. PostgreSQL dokumenterer det som forespørgselskonflikter, sammen med de indstillinger, der afvejer replikaforsinkelse mod dem.

Partitioner, forespørgslerne aldrig nævner. Partitionering deler én tabel efter en nøgle, for eksempel en måned eller en lejer, så en forespørgsel, der nævner nøglen, kun læser de partitioner, den har brug for. En forespørgsel uden nøglen læser hver eneste partition og vinder intet ved opdelingen. Partitionér efter det, forespørgslerne allerede filtrerer på, og forvent kun en gevinst for store tabeller: PostgreSQL-manualens tommelfingerregel er en tabel, der er større end databaseserverens fysiske hukommelse.

Læs mere

- RFC 9110: HTTP Semantics, section 3.3 — forbindelser, klienter og servere, og det tilstandsløse design, der lader implementeringer fordele forespørgsler på flere servere.

- PostgreSQL: Table Partitioning — hvad partitionering er, hvornår det betaler sig, og hvordan partition pruning springer de partitioner over, en forespørgsel ikke kan få brug for.
- PostgreSQL: pg_stat_statements — udvidelsen, der holder styr på antal kald og tider for hver normaliseret sætning.
- PostgreSQL: Hot Standby — læseforespørgsler på en standby og de konflikter, der kan annullere dem.
- RFC 9111: HTTP Caching — hvordan HTTP-caches gemmer og genbruger svar, så gentagne forespørgsler ikke behøver at nå originserveren.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring udrulningsforløbet, driftsrisiciene og dem, der har vagten. Vi kan afgrænse en ændring med kontroller, en vej tilbage og tydeligt ansvar.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er skalerbarhed?« 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/scalability/