Engineer

Skalerbarhet

📈 om mer last krever mer enn flere maskiner

Hva det er

Skalerbarhet sier hvor mye mer et system klarer når du gir det flere ressurser. Et system skalerer godt når dobbelt så mange maskiner bærer omtrent dobbelt så mye last med de samme svartidene. Det skalerer dårlig når den nye maskinvaren for det meste står og venter på noe den ikke kan hjelpe med. Ordet beskriver altså et forhold mellom last og kostnad, ikke en egenskap et system enten har eller mangler.

Et system kan vokse i to retninger. Å skalere opp, eller vertikal skalering, betyr en større maskin: flere kjerner, mer minne, raskere disker. Programvaren trenger ingen endring, men veksten stopper ved den største maskinen det lønner seg å kjøpe. Å skalere ut, eller horisontal skalering, betyr flere maskiner som deler på arbeidet. Der finnes ikke det samme taket, men bare arbeid som lar seg dele opp, får noe igjen for det.

HTTP ble formet med tanke på å skalere ut. RFC 9110 påpeker at mange implementasjoner er avhengige av at HTTP er tilstandsløs, for å kunne gjenbruke tilkoblinger gjennom en proxy eller fordele forespørsler dynamisk over flere servere. Et webnivå som ikke husker noe mellom forespørslene, kan vokse ved å legge til kopier av seg selv bak en lastbalanserer, og hvilken som helst kopi kan svare på hvilken som helst forespørsel.

Last har mer enn én dimensjon, og et system kan skalere langs én og stå fast langs en annen: forespørsler per sekund, mengden lagrede data, antall leietakere og antall skrivere som slåss om den samme posten. Den siste er den vanskeligste, fordi én rad bare kan endres av én transaksjon om gangen, uansett hvor mange servere som står rundt den.

Regnestykket bak dette heter Amdahls lov. Når en del av arbeidet må kjøres i rekkefølge, gjør parallell maskinvare bare resten raskere, og den sekvensielle delen setter grensen: er 5 % av arbeidet serielt, kan hele jobben aldri gå mer enn 20 ganger raskere, selv på tusen maskiner. I en database er den serielle delen ofte en lås. I en applikasjon er den ofte én delt teller, én kø eller ett felles lager for sesjoner.

Bruk begrepet i en beslutning

Bruk «Hva er skalerbarhet?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Overvåking og varsler som sier fra mens en feil fortsatt er liten, og tallene som viser hvor ofte den holder seg liten.

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. Bygde selskapets første observerbarhetsdashbord — selskapets første innsyn i egne systemer

Bygde selskapets første observerbarhetsdashbord Ta kontakt →

Hvorfor det er viktig

Skalerbarheten avgjør hva vekst koster. Skalerer systemet godt, blir en dobling av trafikken en ny linje på fakturaen. Skalerer det dårlig, vokser regningen raskere enn lasten, eller ny maskinvare slutter å hjelpe i det hele tatt, og løsningen blir en omlegging av arkitekturen som må gjøres mens trafikken strømmer inn.

Maskinvare kan avhjelpe flaskehalser i prosessorer, minne eller disker, men ineffektivt arbeid og behov for sekvensiell utføring kan bestå. En spørring som unødvendig leser hele tabellen, kan fortsatt utføre overflødig arbeid på en større server. En større maskin fjerner ikke kravet om at oppdateringer av samme rad utføres én om gangen, og flere applikasjonsservere kan øke antallet skrivende klienter i køen. De fleste systemer som ser ut til å trenge flere maskiner, trenger først å bli lest.

I PostgreSQL begynner lesingen i `pg_stat_statements`, en utvidelse som registrerer hver normaliserte setning med antall kall og kjøretid. Sortert etter total tid i stedet for gjennomsnitt viser den hvor serveren faktisk bruker kreftene sine: en spørring som tar 2 ms og kjøres en million ganger i timen, koster 2 000 sekunder av den timen, mens en rapport på to sekunder som kjøres to ganger, 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 steg mot flere maskiner har sine egne driftskostnader, og derfor betyr rekkefølgen noe. En strammere setning eller en bedre indeks kan forbedre ytelsen uten å legge til enda en server. En lesereplika er konfigurasjon. Partisjonering endrer én tabell. Sharding, å dele dataene mellom separate databaseservere, endrer driften rundt dem: migreringer, sikkerhetskopiering, gjenoppretting og rapporter må ta hensyn til separate shards; en join på tvers av shards kan flyttes til applikasjonskoden; og en spørring som ikke kjenner shardnøkkelen sin, kan måtte spørre alle.

Slik ser det ut i praksis

En PostgreSQL-database bak en mellomstor applikasjon holdt brukerkontoer, leietakere, geografiske data og daglige målinger av strømforbruk. Trafikken var lav, men databasen ble tregere etter hvert som datamengdene vokste. Oppdraget var å gjøre den 10 ganger raskere uten å bygge om arkitekturen, og plass til framtidig datavekst var en del av målet. Arbeidet besto av GiST-indekser for de romlige dataene, sammensatte indekser på leietaker og datointervall, tidsbasert partisjonering av strømtabellen, hash-partisjonering av de geografiske dataene, omskrevne spørringer sjekket med `EXPLAIN ANALYZE`, og tilkoblingspooling. Svartidene ble 10 ganger bedre, databasens ressursbruk falt med 40 %, og systemet tok imot voksende datamengder uten å bli tregere, med mindre behov for maskinvareoppgraderinger.

Hos et mellomstort teknologiselskap med ti databaseservere på PostgreSQL og SQL Server ble infrastrukturen skalert for 40 % økning i brukertrafikken uten forringelse av tjenesten. Samtidig ble svartidene for spørringer 30 % bedre, gjennom justering av konfigurasjonen og optimalisering av indekser.

Rekkefølgen arbeidet følger, står i et notat om å optimalisere før du sharder: se, stram inn, ta lesingene bort fra primærnoden, partisjoner tabellene rundt det hete punktet, og shard først når én maskin er full av godt arbeid. Notatet nevner tre tilfeller der sharding er rimelig: primærnoden er mettet av setninger som allerede er slanke, dataene har vokst fra vedlikeholdet sitt, eller isolasjon er selve kravet.

Hvor det går galt

Flere webservere foran én database. Hver applikasjonsserver åpner sine egne tilkoblinger, og når mange ganger flere økter er aktive enn databasen har kjerner, går serverens tid med til å bytte mellom dem. Tilkoblingspooling løser det før ny maskinvare gjør det, og enda en webserver gjør det verre.

Én varm rad. Når hver forespørsel skriver til den samme posten, f.eks. en lagerteller eller en saldo, begrenses gjennomstrømningen til omtrent antallet skrivinger én lås kan håndtere, delt på hvor lenge låsen holdes, uansett antall maskiner. Alt som er tregt inne i låsen, senker det taket. Hold transaksjonen kort, legg eksterne kall utenfor den, og når én vare vokser fra maskinen sin, del lageret opp i mindre biter som hver har nøyaktig én eier.

Tilstand som bor på én server. Sesjoner i minnet til én prosess, opplastinger skrevet til lokal disk eller en planlagt jobb som tar for gitt at den er den eneste kopien, hindrer alle sammen at en ny instans blir likeverdig med den første. Legg den tilstanden i et delt lager, slik at hvilken som helst instans kan svare på hvilken som helst forespørsel.

Replikaer behandlet som kopier av primærnoden. En strømmende replika henger litt etter, så den som nettopp har skrevet noe og så leser fra replikaen, kan få se den gamle verdien. Les ferske skrivinger fra primærnoden. En lang rapport på en standby kan også bli avbrutt når den holder igjen endringene som spilles av. PostgreSQL dokumenterer dette som spørringskonflikter, sammen med innstillingene som veier replikaforsinkelse opp mot dem.

Partisjoner som spørringene aldri nevner. Partisjonering deler én tabell etter en nøkkel, f.eks. måned eller leietaker, slik at en spørring som oppgir nøkkelen, bare leser partisjonene den trenger. En spørring som utelater nøkkelen, leser alle partisjonene og får ingenting igjen for oppdelingen. Partisjoner etter det spørringene allerede filtrerer på, og regn med at det lønner seg først for store tabeller: tommelfingerregelen i PostgreSQL-manualen er en tabell som er større enn databaseserverens fysiske minne.

Les mer

- RFC 9110: HTTP Semantics, section 3.3 — tilkoblinger, klienter og servere, og den tilstandsløse utformingen som lar implementasjoner fordele forespørsler over flere servere.

- PostgreSQL: Table Partitioning — hva partisjonering er, når den lønner seg, og hvordan partition pruning hopper over partisjonene en spørring ikke kan trenge.
- PostgreSQL: pg_stat_statements — utvidelsen som tar vare på antall kall og kjøretid for hver normaliserte setning.
- PostgreSQL: Hot Standby — lesespørringer på en standby, og konfliktene som kan avbryte dem.
- RFC 9111: HTTP Caching — hvordan HTTP-cacher lagrer og gjenbruker svar, slik at gjentatte forespørsler slipper å gå helt til opprinnelsesserveren.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er skalerbarhet?» 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/scalability/