Engineer

Skalbarhet

↗ om mer last kräver mer än fler maskiner

Vad det är

Skalbarhet beskriver hur ett systems kapacitet växer när det får fler resurser. Ett system skalar bra när dubbelt så många maskiner bär ungefär dubbelt så mycket last med samma svarstider, och dåligt när den extra hårdvaran mest står och väntar på något den inte kan hjälpa till med. Ordet beskriver alltså ett förhållande mellan last och kostnad, inte en egenskap som ett system antingen har eller saknar.

Det finns två riktningar att växa i. Att skala upp, vertikal skalning, betyder en större maskin: fler kärnor, mer minne, snabbare diskar. Programvaran behöver inte ändras, och vägen tar slut vid den största maskin som är värd att köpa. Att skala ut, horisontell skalning, betyder fler maskiner som delar på arbetet. Där finns inget sådant tak, men bara arbete som går att dela upp har nytta av det.

HTTP formades med utskalning i tankarna. RFC 9110 påpekar att många implementationer förlitar sig på att HTTP är tillståndslöst för att återanvända anslutningar via proxyer eller för att fördela begäranden dynamiskt över flera servrar. Ett webbskikt som inte sparar något mellan begärandena kan växa genom att ställa fler kopior av sig själv bakom en lastbalanserare, och vilken kopia som helst kan då besvara vilken begäran som helst.

Last har fler än en dimension, och ett system kan skala längs en av dem och köra fast längs en annan: begäranden per sekund, mängden lagrad data, antalet tenants och antalet processer som vill skriva till samma post samtidigt. Den sista är svårast, eftersom en enskild rad bara kan ändras av en transaktion i taget, hur många servrar som än står runt den.

Räknestycket bakom det heter Amdahls lag. När en del av arbetet måste köras i sekvens snabbar parallell hårdvara bara upp resten, och den sekventiella delen sätter gränsen: är 5 % av arbetet seriellt kan hela jobbet aldrig gå mer än 20 gånger snabbare, inte ens på tusen maskiner. I en databas är den seriella delen ofta ett lås. I en applikation är den ofta en enda delad räknare, en kö eller ett sessionslager.

Använd begreppet i ett beslut

Använd ”Vad är skalbarhet?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Övervakning och larm som säger till medan ett fel fortfarande är litet, och siffrorna som visar hur ofta det förblir så.

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. Byggde företagets första observerbarhetsdashboard — företagets första överblick över de egna systemen

Byggde företagets första observerbarhetsdashboard Hör av er →

Varför det spelar roll

Skalbarheten avgör vad tillväxt kostar. Skalar systemet bra blir en fördubbling av trafiken en rad på en faktura. Skalar det dåligt växer notan snabbare än lasten, eller så slutar mer hårdvara att hjälpa över huvud taget, och lösningen blir en omdesign som måste göras medan trafiken redan strömmar in.

Hårdvara kan avhjälpa flaskhalsar i processorer, minne eller diskar, men ineffektivt arbete och krav på sekventiell körning kan finnas kvar. En databasfråga som i onödan skannar en hel tabell kan fortfarande utföra överflödigt arbete på en större server. En större maskin tar inte bort kravet på att uppdateringar av samma rad utförs en i taget, och fler applikationsservrar kan öka antalet skrivande klienter i kön. De flesta system som ser ut att behöva fler maskiner behöver först läsas.

I PostgreSQL börjar läsningen i `pg_stat_statements`, ett tillägg som registrerar varje normaliserad sats med antal anrop och körtid. Sorterad på total tid i stället för på medelvärde visar den var servern faktiskt lägger sin tid: en SQL-fråga som tar 2 ms och körs en miljon gånger i timmen kostar 2 000 sekunder av den timmen, medan en rapport på två sekunder som körs två gånger kostar fyra.

```
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;
```

Varje steg mot fler maskiner för med sig egna driftkostnader, och därför spelar ordningen roll. En stramare sats eller ett bättre index kan förbättra prestandan utan att lägga till ännu en server. En läsreplik är konfiguration. Partitionering ändrar en tabell. Sharding, där datan delas upp på separata databasservrar, ändrar driften runt omkring: migreringar, säkerhetskopiering, återställning och rapporter måste ta hänsyn till separata shards; en join över shards kan flyttas till applikationskoden; och en SQL-fråga som inte känner till sin shardnyckel kan behöva fråga alla.

Så ser det ut i praktiken

En PostgreSQL-databas bakom en medelstor applikation höll användarkonton, tenants, geografiska data och daglig elförbrukning. Trafiken var låg, men databasen blev trögare i takt med att datavolymerna växte. Uppdraget var att göra den 10 gånger snabbare utan att göra om arkitekturen, och utrymme för framtida datatillväxt var en uttalad del av målet. Arbetet bestod av GiST-index för de geografiska data, sammansatta index på tenant och datumintervall, eltabellen partitionerad efter tid, hashpartitionering av geodatan, omskrivna databasfrågor kontrollerade med `EXPLAIN ANALYZE` samt anslutningspoolning. Svarstiderna blev tio gånger bättre, databasens resursanvändning minskade med 40 %, och systemet tog emot växande datavolymer utan att bli långsammare, med mindre behov av hårdvaruuppgraderingar.

På ett medelstort företag med tio databasservrar för PostgreSQL och SQL Server skalades infrastrukturen så att den klarade 40 % mer användartrafik utan försämrad tjänstekvalitet. Samtidigt blev svarstiderna för databasfrågor 30 % bättre genom konfigurationsjustering och indexoptimering.

Ordningen som arbetet följer beskrivs i en anteckning om att optimera innan man shardar: titta, skärp, flytta läsningarna från primärnoden, partitionera tabellerna runt den heta punkten och sharda först när en maskin är full av bra arbete. Anteckningen nämner tre fall där sharding är rimligt: primärnoden är mättad av satser som redan är slimmade, datan har vuxit ifrån sitt underhåll, eller isoleringen är själva kravet.

Var det går fel

Fler webbservrar framför en databas. Varje applikationsserver öppnar sina egna anslutningar, och när många gånger fler sessioner är aktiva än databasen har kärnor går serverns tid åt till att växla mellan dem. En anslutningspool löser det innan någon hårdvara gör det, och ännu en webbserver gör saken värre.

En het rad. När varje begäran skriver till samma post, t.ex. en lagerräknare eller ett kontosaldo, stannar genomströmningen vid ungefär de skrivningar som ett lås kan bära, delat med hur länge låset hålls, oavsett antal maskiner. Allt som går långsamt inuti låset sänker taket. Håll transaktionen kort, lägg externa anrop utanför den, och när en vara växer ur sin maskin: dela upp dess lagersaldo i mindre poster som var och en har exakt en ägare.

Tillstånd som bor på en server. Sessioner i minnet hos en enda process, uppladdade filer på dess lokala disk eller ett schemalagt jobb som utgår från att det är den enda kopian hindrar alla en andra instans från att bli likvärdig. Lägg det tillståndet i en delad lagring, så att vilken instans som helst kan besvara vilken begäran som helst.

Repliker som behandlas som kopior av primärnoden. En strömmande replik ligger lite efter, så en användare som just har skrivit något och sedan läser från repliken ser det kanske inte; läs nyss skrivna data från primärnoden. En lång rapport på en standby kan också avbrytas när den håller upp ändringarna som spelas upp. PostgreSQL dokumenterar det som frågekonflikter, tillsammans med de inställningar som väger replikens eftersläpning mot dem.

Partitioner som SQL-frågorna aldrig nämner. Partitionering delar en tabell efter en nyckel, t.ex. en månad eller en tenant, så att en SQL-fråga som anger nyckeln bara läser de partitioner den behöver. En SQL-fråga utan nyckeln läser varenda partition och vinner ingenting på uppdelningen. Partitionera på det som SQL-frågorna redan filtrerar på, och räkna med vinst bara för stora tabeller: tumregeln i PostgreSQL-manualen är en tabell som är större än databasserverns fysiska minne.

Läs mer

- RFC 9110: HTTP Semantics, section 3.3 — anslutningar, klienter och servrar, och den tillståndslösa design som låter implementationer fördela begäranden över flera servrar.

- PostgreSQL: Table Partitioning — vad partitionering är, när den lönar sig och hur partition pruning hoppar över de partitioner som en SQL-fråga inte kan behöva.
- PostgreSQL: pg_stat_statements — tillägget som håller räkningen på anrop och tider för varje normaliserad sats.
- PostgreSQL: Hot Standby — SQL-frågor som bara läser på en standby och de konflikter som kan avbryta dem.
- RFC 9111: HTTP Caching — hur HTTP-cachar sparar och återanvänder svar, så att upprepade begäranden inte behöver nå ursprungsservern.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är skalbarhet?” 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/scalability/