Engineer

Масштабованість

↗ чи потребує більше навантаження чогось, крім машин

Що це таке

Масштабованість — це те, як зростає потужність системи, коли їй додають ресурси. Система масштабується добре, коли вдвічі більше машин несуть приблизно вдвічі більше навантаження з тим самим часом відповіді, і погано, коли додане залізо здебільшого чекає на щось, чому воно зарадити не може. Це слово описує співвідношення між навантаженням і вартістю, а не властивість, яка в системи або є, або її немає.

Рости можна у двох напрямках. Вертикальне масштабування (scaling up) — це більша машина: більше ядер, більше пам'яті, швидші диски. Воно не потребує змін у програмі й зупиняється на найбільшій машині, яку ще варто купувати. Горизонтальне масштабування (scaling out) — це більше машин, що ділять роботу між собою. Такої стелі воно не має, але виграє від нього лише та робота, яку можна поділити.

HTTP створювали з думкою про горизонтальне масштабування. RFC 9110 зауважує, що багато реалізацій покладаються на те, що HTTP не зберігає стану між запитами: так вони повторно використовують з'єднання через проксі або динамічно розподіляють запити між кількома серверами. Вебрівень, який нічого не тримає між запитами, може рости, додаючи власні копії за балансувальником навантаження, і будь-яка копія відповість на будь-який запит.

Навантаження має кілька вимірів, і система може масштабуватися за одним і застрягати за іншим: запити за секунду, обсяг збережених даних, кількість орендарів і кількість записувачів, які змагаються за той самий запис. Останній вимір найважчий, бо один рядок водночас може змінювати лише одна транзакція, скільки б серверів довкола нього не стояло.

Арифметику за цим описує закон Амдала. Коли частина роботи мусить виконуватися послідовно, паралельне залізо прискорює лише решту, тож послідовна частина й задає межу: якщо послідовні 5% роботи, уся задача ніколи не виконається більш ніж у 20 разів швидше, навіть на тисячі машин. У базі даних послідовною частиною часто є блокування. У застосунку — один спільний лічильник, черга чи сховище сесій.

Застосуйте поняття до рішення

Застосуйте «Що таке масштабованість?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Моніторинг і сповіщення, які попереджають, поки збій ще малий, і цифри, що показують, як часто так і залишається.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Запустила перший дашборд спостережуваності — перший огляд власних систем компанії

Запустила перший дашборд спостережуваності Зв'язатися з нами →

Чому це важливо

Масштабованість визначає, скільки коштує зростання. Коли система масштабується добре, подвоєння трафіку — це ще один рядок у рахунку. Коли погано, рахунок росте швидше за навантаження або додане залізо взагалі перестає допомагати, і виправленням стає переробка, яку доводиться робити просто під напливом трафіку.

Апаратне забезпечення може усунути вузькі місця в роботі процесорів, пам’яті чи дисків, але неефективна робота й потреба в послідовному виконанні операцій можуть залишитися. Запит, що без потреби сканує всю таблицю, і на більшому сервері може виконувати зайву роботу. Більша машина не усуває потреби виконувати оновлення одного й того самого рядка по черзі, а додаткові сервери застосунку можуть додати до черги більше клієнтів, які виконують запис. Більшість систем, яким нібито потрібно більше машин, спершу треба прочитати.

У PostgreSQL читати варто почати з `pg_stat_statements` — розширення, яке записує кожен нормалізований запит разом із кількістю викликів і часом виконання. Якщо сортувати за сумарним часом, а не за середнім, видно, на що сервер насправді витрачається: запит, що триває 2 мс і виконується мільйон разів на годину, забирає 2 000 секунд цієї години, а двосекундний звіт, запущений двічі, — чотири.

```
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;
```

Кожен крок до більшої кількості машин має власну ціну в експлуатації, тому порядок важливий. Оптимізований запит чи кращий індекс можуть підвищити продуктивність без додавання ще одного сервера. Репліка для читання — справа конфігурації. Партиціонування змінює одну таблицю. Шардування, тобто розподіл даних між окремими серверами баз даних, змінює пов'язані з цим експлуатаційні процеси: міграції, резервне копіювання, відновлення та звіти мають враховувати окремі шарди; join між шардами може перейти в код застосунку; а запит, що не знає свого ключа шарда, може потребувати звернення до всіх шардів.

Як це виглядає на практиці

База PostgreSQL за застосунком середнього розміру зберігала облікові записи користувачів, дані орендарів, географічні дані та щоденне споживання електроенергії. Вона працювала за низького навантаження, але з ростом обсягів даних сповільнилася. Завданням було зробити її вдесятеро швидшою без переробки архітектури, і запас для майбутнього зростання даних входив у ціль. Для цього додали індекси GiST для геопросторових даних і композитні індекси за орендарем і діапазоном дат, розділили таблицю електроенергії на партиції за часом, застосували хеш-партиціонування до географічних даних, переписали запити, перевіряючи їх через `EXPLAIN ANALYZE` , і налаштували пулінг з'єднань. Час відповіді покращився вдесятеро, використання ресурсів бази знизилося на 40%, і система витримувала зростаючі обсяги даних без погіршення продуктивності та з меншою потребою в апаратних оновленнях.

У компанії середнього розміру з десятьма серверами баз даних PostgreSQL і SQL Server інфраструктуру масштабували під зростання трафіку на 40% без погіршення обслуговування. Паралельно час відповіді запитів покращився на 30% завдяки налаштуванню конфігурації та оптимізації індексів.

Порядок, якого дотримується ця робота, описано в нотатці про оптимізацію перед шардуванням: дослідити, оптимізувати, зняти читання з основної бази, розділити таблиці навколо гарячої точки на партиції і шардувати лише

тоді, коли одна машина повністю завантажена корисною роботою. Нотатка називає три випадки, коли шардування доречне: основна база завантажена до межі вже ощадними запитами, обсяг даних перевищив можливості їх обслуговування або вимогою є сама ізоляція.

Де виникають проблеми

Більше вебсерверів перед однією базою. Кожен сервер застосунку відкриває власні з'єднання, і коли активних сесій у багато разів більше, ніж ядер у бази, сервер витрачає час на перемикання між ними. Пул з'єднань виправить це раніше за будь-яке залізо, а ще один вебсервер лише погіршить ситуацію.

Один гарячий рядок. Коли кожен запит пише в той самий запис, наприклад у лічильник залишку чи баланс, пропускна здатність обмежена приблизно кількістю записів, яку несе одне блокування, поділеною на час його утримання, — на будь-якій кількості машин. Усе повільне всередині блокування знижує цю стелю. Тримайте транзакцію короткою, виносьте зовнішні виклики за її межі, а коли один товар переростає свою машину, діліть його запас на менші частини, кожна з яких має рівно одного власника.

Стан, що живе на одному сервері. Сесії в пам'яті одного процесу, завантажені файли на його локальному диску чи запланована задача, яка вважає себе єдиною копією, — усе це заважає другому екземпляру стати рівноправним. Тримайте такий стан у спільному сховищі, щоб будь-який екземпляр міг відповісти на будь-який запит.

Репліки, які вважають копіями основної бази. Потокова репліка трохи відстає, тож користувач, який щойно щось записав і читає з репліки, може цього не побачити; читайте свіжі записи з основної бази. Довгий звіт на standby-сервері також можуть скасувати, коли він затримує зміни, які репліка відтворює. Документація PostgreSQL називає це конфліктами запитів і описує налаштування, що обмінюють відставання репліки на менше таких конфліктів.

Партиції, яких запити не називають. Партиціонування ділить одну таблицю за ключем, наприклад за місяцем чи орендарем, тож запит, що називає ключ, читає лише потрібні партиції. Запит без ключа читає всі партиції й не виграє від поділу. Партиціонуйте за тим, за чим запити вже фільтрують,

і чекайте виграшу лише для великих таблиць: емпіричне правило з документації PostgreSQL — таблиця, більша за фізичну пам’ять сервера бази даних.

Що почитати далі

- RFC 9110: HTTP Semantics, section 3.3 — з’єднання, клієнти й сервери та те, як відсутність стану між запитами дає реалізаціям змогу розподіляти запити між кількома серверами.
- PostgreSQL: Table Partitioning — що таке партиціонування, коли воно окупається і як відсікання партицій (partition pruning) пропускає ті, що запиту знадобитися не можуть.
- PostgreSQL: pg_stat_statements — розширення, що веде облік викликів і часу для кожного нормалізованого запиту.
- PostgreSQL: Hot Standby — запити лише на читання на standby-сервері та конфлікти, які можуть їх скасувати.
- RFC 9111: HTTP Caching — як HTTP-кеші зберігають і повторно використовують відповіді, щоб повторні запити могли не доходити до вихідного сервера.

Словник

Обирайте за фактичними умовами

Опишіть шлях розгортання, експлуатаційні ризики та чергову команду. Ми можемо визначити зміну з перевірками, шляхом відновлення та чіткою відповідальністю.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке масштабованість?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/scalability/