Engineer

Scalabilité

↗ Jusqu'où ajouter des machines suffit à gérer plus de charge

Ce que c'est

La scalabilité décrit la façon dont la capacité d'un système augmente lorsqu'on lui ajoute des ressources. Un système passe bien à l'échelle lorsque deux fois plus de machines supportent environ deux fois plus de charge avec les mêmes temps de réponse. Il passe mal à l'échelle lorsque le matériel supplémentaire attend surtout quelque chose qu'il ne peut pas aider à accomplir. Le mot décrit une relation entre charge et coût, pas une caractéristique qu'un système possède ou non.

Il existe deux directions de croissance. La mise à l'échelle verticale consiste à utiliser une machine plus puissante : davantage de cœurs et de mémoire, des disques plus rapides. Elle ne demande aucune modification du logiciel et s'arrête à la plus grosse machine qu'il reste rentable d'acheter. La mise à l'échelle horizontale consiste à répartir le travail entre davantage de machines. Elle n'a pas ce plafond, mais ne profite qu'aux tâches divisibles.

HTTP a été conçu en tenant compte de cette croissance horizontale. La RFC 9110 note que de nombreuses implémentations s'appuient sur l'absence d'état de HTTP pour réutiliser des connexions passant par des proxys ou répartir dynamiquement les requêtes entre plusieurs serveurs. Une couche web qui ne conserve rien entre les requêtes peut croître par ajout de copies derrière un répartiteur de charge. Chacune peut répondre à n'importe quelle requête.

La charge a plusieurs dimensions. Un système peut progresser sur l'une et bloquer sur une autre : requêtes par seconde, volume de données stockées, nombre d'organisations clientes et nombre de clients qui tentent d'écrire simultanément dans le même enregistrement. Cette dernière dimension est la plus difficile : une seule transaction peut modifier une ligne à la fois, quel que soit le nombre de serveurs autour.

La loi d'Amdahl en donne l'arithmétique. Lorsqu'une partie du travail doit s'exécuter en séquence, le matériel parallèle n'accélère que le reste. La partie séquentielle fixe donc la limite : avec 5 % du travail en série, la tâche complète ne peut jamais être accélérée de plus de 20 fois, même sur mille machines. Dans une base de données, cette partie est souvent un verrou. Dans une application, c'est souvent un élément partagé : un compteur, une file d'attente ou un stockage de sessions.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce que la scalabilité ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Une supervision et des alertes qui signalent les incidents tant qu'ils restent mineurs, avec les chiffres qui montrent à quelle fréquence ils le restent.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Premier tableau de bord d'observabilité — première vue de l'entreprise sur ses propres systèmes

Premier tableau de bord d'observabilité Contactez-nous →

Pourquoi c'est important

La scalabilité détermine le coût de la croissance. Lorsqu'un système passe bien à l'échelle, doubler le trafic représente une ligne sur une facture. Lorsqu'il le fait mal, la facture augmente plus vite que la charge, ou ajouter du matériel n'aide plus du tout. La correction devient alors une refonte menée pendant que le trafic continue d'arriver.

Le matériel peut soulager les goulets d'étranglement des processeurs, de la mémoire ou des disques, mais le travail inefficace et les contraintes d'exécution séquentielle peuvent persister. Une requête qui parcourt inutilement une table entière peut encore effectuer du travail superflu sur un serveur plus puissant. Une machine plus puissante ne supprime pas l'exécution séquentielle des mises à jour d'une même ligne partagée, et des serveurs applicatifs supplémentaires peuvent ajouter à la file d'attente des clients qui effectuent des écritures. La plupart des systèmes semblant nécessiter davantage de machines ont d'abord besoin qu'on les examine.

Dans PostgreSQL, l'examen commence par `pg_stat_statements`, une extension qui enregistre chaque instruction normalisée avec son nombre d'appels et son temps d'exécution. Un tri par temps total plutôt que moyen montre où le serveur

dépense réellement ses ressources : une requête de 2 ms exécutée un million de fois par heure coûte 2 000 secondes de cette heure, contre quatre pour un rapport de deux secondes lancé deux fois.

```
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;
```

Chaque étape vers davantage de machines entraîne ses propres coûts d'exploitation, d'où l'importance de l'ordre. Une instruction plus efficace ou un meilleur index peut améliorer les performances sans ajouter de serveur. Un réplicat de lecture relève de la configuration. Le partitionnement modifie une table. Le sharding, qui répartit les données entre serveurs de bases de données distincts, modifie aussi l'exploitation : les migrations, sauvegardes, restaurations et rapports doivent tenir compte des shards distincts ; une jointure entre shards peut être déplacée dans le code applicatif ; une requête ignorant sa clé de sharding peut devoir les interroger tous.

À quoi cela ressemble en pratique

Une base PostgreSQL derrière une application de taille moyenne contenait les comptes utilisateurs, les organisations clientes, les données géographiques et la consommation électrique quotidienne. Le trafic était faible, mais la base ralentissait à mesure que les données augmentaient. L'objectif était de la rendre dix fois plus rapide sans refondre l'architecture, en prévoyant la croissance future des données. Le travail a porté sur des index GiST pour les données spatiales, des index composites sur l'organisation cliente et la plage de dates, le partitionnement temporel de la table de consommation électrique, le partitionnement par hachage des données géographiques, la réécriture des requêtes vérifiées avec `EXPLAIN ANALYZE` et un pool de connexions. Les temps de réponse ont été divisés par dix, l'utilisation des ressources de la base a baissé de 40 % et le système a absorbé l'augmentation des données sans ralentir, avec moins de besoins de mise à niveau du matériel.

Dans une entreprise de taille moyenne exploitant dix serveurs de bases PostgreSQL et SQL Server, l'infrastructure a été dimensionnée pour absorber une hausse de 40 % du trafic utilisateur sans dégradation du service.

Parallèlement, les temps de réponse des requêtes ont été améliorés de 30 % grâce au réglage de la configuration et à l'optimisation des index.

L'ordre du travail est présenté dans une note sur l'optimisation avant le sharding : observer, optimiser, décharger le serveur principal des lectures, partitionner les tables autour du point chaud, puis ne répartir entre serveurs que lorsqu'une machine est saturée de travail utile. La note décrit trois cas où le sharding est raisonnable : le serveur principal est saturé par des instructions déjà efficaces, les données ont dépassé les capacités de maintenance ou l'isolation est elle-même une exigence.

Où les problèmes surviennent

Davantage de serveurs web devant une seule base. Chaque serveur applicatif ouvre ses propres connexions. Lorsque le nombre de sessions actives dépasse de nombreuses fois le nombre de cœurs de la base, le serveur passe son temps à basculer entre elles. Un pool de connexions corrige cela avant tout ajout de matériel, et un serveur web supplémentaire aggrave le problème.

Une ligne très sollicitée. Lorsque chaque requête écrit dans le même enregistrement, comme un compteur de stock ou un solde, le débit est limité approximativement au nombre d'écritures qu'un verrou peut traiter, divisé par sa durée de détention, quel que soit le nombre de machines. Toute opération lente sous ce verrou abaisse le plafond. Gardez les transactions courtes et les appels externes en dehors. Lorsqu'un article dépasse les capacités de sa machine, répartissez son stock en petits ensembles ayant chacun un propriétaire unique.

Un état conservé sur un seul serveur. Des sessions dans la mémoire d'un processus, des fichiers téléversés sur son disque local ou une tâche planifiée qui suppose qu'aucune autre copie ne s'exécute empêchent une seconde instance d'être équivalente à la première. Conservez cet état dans un stockage partagé pour que chaque instance puisse répondre à n'importe quelle requête.

Des réplicats traités comme des copies du serveur principal. Un réplicat alimenté en continu a un léger retard. Un utilisateur qui vient d'écrire puis lit sur ce réplicat peut ne pas voir son écriture : lisez les données récemment écrites sur le serveur principal. Un long rapport sur un serveur de secours peut aussi être annulé lorsqu'il retarde la réapplication des changements. PostgreSQL documente ces conflits de requêtes et les paramètres qui permettent un compromis entre eux et le retard du réplicat.

Des partitions que les requêtes ne désignent jamais. Le partitionnement divise une table selon une clé, comme un mois ou une organisation cliente. Une requête indiquant la clé ne lit que les partitions nécessaires. Une requête qui l'omet lit toutes les partitions et ne gagne rien à cette division. Partitionnez selon les critères déjà utilisés par les requêtes et n'attendez un bénéfice que pour les grandes tables : le manuel PostgreSQL donne comme règle approximative une table dépassant la mémoire physique du serveur de base de données.

Pour aller plus loin

- RFC 9110 : sémantique HTTP, section 3.3 — connexions, clients, serveurs et conception sans état permettant de répartir les requêtes entre plusieurs serveurs.
- PostgreSQL : partitionnement des tables — définition, conditions de rentabilité et élagage des partitions dont une requête ne peut pas avoir besoin.
- PostgreSQL : pg_stat_statements — extension conservant le nombre d'appels et les durées de chaque instruction normalisée.
- PostgreSQL : serveur de secours en lecture — requêtes en lecture seule et conflits pouvant les annuler.
- RFC 9111 : cache HTTP — stockage et réutilisation des réponses pour que les requêtes répétées n'atteignent pas nécessairement le serveur d'origine.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le parcours de déploiement, les risques opérationnels et les personnes d'astreinte. Nous pouvons définir un changement avec des vérifications, une procédure de reprise et une responsabilité claire.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce que la scalabilité ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/scalability/