Технічні науки

УДК 004.658.3

Денік Юрій Олександрович
студент
Національного технічного університету України
«Київський політехнічний інститут імені Ігоря Сікорського»
Деник Юрий Александрович
студент
Национального технического университета Украины
«Киевский политехнический институт имени Игоря Сикорского»
Denik Y.
student
National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute"

ПРИНЦИПИ РОЗРОБКИ ВИСОКОНАВАНТАЖЕНИХ ВЕБ-СИСТЕМ
ПРИНЦИПЫ РАЗРАБОТКИ ВЫСОКОНАГРУЖЕННЫХ ВЕБ-СИСТЕМ
THE PRINCIPLES OF HIGH LOAD WEB-SYSTEMS DEVELOPMENT

Анотація: Виконано огляд проблеми розробки високонавантажених веб-систем та запропоновані відповідні архітектурні рішення.

Ключові слова: веб-система, високе навантаження, шардінг, реплікація, сервер, база даних, розробка, архітектура.

Аннотация: Выполнен осмотр проблемы разработки высоконагруженных веб-систем и предложены соответствующие архитектурные решения.

Ключевые слова: веб-система, высокая нагрузка, шардинг, репликация, сервер, база данных, разработка, архитектура.

Summary: The overview of problem in high load web-systems development is fulfilled and appropriate architectural solutions are offered.

Key words: web-system, high load, sharding, replication, server, database, development, architecture.

Постановка задачі

Поняття високої завантаженості – достатньо відносне, оскільки всі веб-сайти специфічні і в залежності від своєї архітектури, структури можуть по різному реагувати на однакову кількість запитів. Відповідно навантаження також буде різним. Тому високе навантаження, визначається як навантаження, з яким дане апаратне забезпечення, по певним причинам, не може справитись. Коли робота системи доходить до такої точки, є сенс починати масштабування і оптимізацію інфраструктури додатку.

Щоб зрозуміти, що існує проблема перенавантаження, її потрібно діагностувати. Саме тому, при будь-якому навантаженні потрібна надійна система моніторингу, яка допоможе визначити момент, коли потрібно розпочинати масштабування. Основними симптомами даної проблеми є повільне завантаження сторінок, випадкові помилки, обірвані з'єднання з веб-сервером, часткове завантаження вмісту додатку і деякі інші. В такому випадку, в першу чергу варто визначити причину неполадок. Слід перевірити чи правильно налаштований веб-сервер, оскільки це допоможе значно розвантажити залізо. Оптимізація клієнтської частини також дозволить зекономити значну кількість ресурсів і пришвидшити роботу системи для користувача. Часто проблеми зустрічаються в базах даних.

Але, чи можна передбачити і запобігти даних проблем до того моменту, коли вони виникнуть? Звичайно, так. Якщо ви працюєте над сайтом, який поступово розвивається і розширює свою аудиторію, то рано

чи пізно ви будете змушеними переходити в режим високого навантаження, і продумати це варто заранння. Перш за все, переконайтесь, що всі прості речі вже виконані і вдосконалені. З перших днів роботи, потрібно вмикати моніторинг, оскільки тут діє правило – чим більше ви знаєте, тим більш готовими ви будете до розв'язання задачі. Але, ні в якому разі не варто займатись попередньою оптимізацією. Інколи, можна вгадати, але частіше за все, ви ризикуєте витратити час на оптимізацію того, що швидше всього зазнає значних змін з часом. Краще зосередитись на гнучкості системи, що дозволить швидко вносити необхідні зміни. Проте найважливішу роль в цьому процесі відіграє планування усієї архітектури системи.

Основні принципи розробки

Розробляючи успішний, що означає і масивний, веб-додаток, необхідно розуміти принципи побудови великих систем.

- Динаміка

Ніколи невідомо, що може статись із системою завтра. Можливо, кількість користувачів збільшиться в декілька разів, а, можливо, різко почне набувати популярності якийсь другорядний функціонал. Разом зі збільшенням системи, буде зменшуватись ефективність довгострокового планування.

Успішність роботи над великим проектом не полягає у детальному плануванні всіх аспектів. Основні зусилля повинні бути направлені на забезпечення гнучкості системи, адже ця властивість дозволить швидко вносити зміни в майбутньому і, тому, воно вважається одним з найважливіших.

- Поступовий ріст

Не варто намагатись спрогнозувати величину аудиторії у далекій перспективі. Те саме стосується і архітектури додатку. Основа успішної

розробки – поступові рішення, як по програмній частині, так і по апаратній.

- Прогресивні зміни

Під час роботи над масивними проектами ви рухаєтесь, по завданням і рішенням, не поступово, а хаотично. Постійно відбуватиметься переключення з одних рішень, на інші, потрібно буде перероблювати і вдосконалювати їх. Проте не потрібно концентруватись на абсолютно усіх завданнях. Є сенс витрачати час тільки на, приблизно, 95% функцій, адже решта, зазвичай, є частковими випадками, які призводять до вкрай сильного ускладнення системи [1].

Тому слід розставляти пріоритети правильно – вирішуйте ті проблеми, які виникають у абсолютної більшості користувачів.

Архітектурні рішення

Архітектурні рішення - це фундамент створення будь-яких додатків. Включаючи і веб-додатки з високим навантаженням. Важливо розуміти, що правильно спроектована архітектура веб-сервісу забезпечує успішність його роботи, зокрема, і здатність справлятись з навантаженнями.

Масштабування будь-якого веб-додатку – це поступовий процес, який включає в себе наступні етапи:

1. Аналіз навантаження.
2. Визначення ділянок системи, які мають найбільший ризик перенавантаження.
3. Винесення таких ділянок на окремі вузли та їх оптимізація.
4. Повернення до пункту 1.

1. Проста архітектура веб-додатку.

Новий веб-сервіс зазвичай запускається на одному сервері, на якому працюють і веб-сервер, і база даних, і сам додаток. Це економить час та фінанси на запуск, і, відповідно, такий підхід є прийнятним для старту.

2. Відокремлення бази даних

Частіше всього, першим вузлом, який зазнає навантаження, є база даних, оскільки кожен запит від користувача – це від 10 до 100 запитів до бази даних. Винесення бази даних на окремий сервер дозволить збільшити її продуктивність і знизити її негативний вплив на решту компонентів. Масштабування баз даних – одне з найскладніших завдань під час росту проекту. Для цього існує багато прийомів та патернів: денормалізація, реплікація, шардінг та багато інших.

3. Відокремлення серверу.

Далі на черзі йде веб-сервер. Його відокремлення на окремий вузол дозволить залишити більше ресурсів для додатку. Якщо використовується завантаження файлів, для цього також виділяється окремий вузол для файлового сховища.

4. Декілька бекендів.

Коли навантаження росте, веб-сервіс поступово починає сповільнювати швидкість роботи. В певний момент причина буде лежати в самій реалізації додатку. Тоді варто встановити декілька бек-ендів та збалансувати навантаження між ними. Як тільки почнеться використання декількох бекендів, запити від одного користувача будуть направлятись до різних серверів. Виникне потреба використання єдиного сховища для сесій.

5. Черги задач

Черги завдань дозволяють виконувати тяжкі операції асинхронно, не сповільнюючи основного додатку. Сервер черги приймає завдання від

додатку. Worker серверу оброблює задачі. Їх кількість слід збільшувати, коли середня кількість задач в черзі буде поступово рости [2].

6. Файлові сховища

Завантаження і обробка файлів зазвичай відбувається на бекенді. Коли бекендів декілька, це викликає незручності:

- Прийдеться пам'ятати, на який бекенд був завантажений файл.
- Завантаження і обробка файлів може сильно знизити продуктивність бекенду.
- Прийдеться використовувати сервери з великими дисками, в чому, зазвичай, нема необхідності.

Правильним рішенням буде використання окремих серверів для завантаження, зберігання та обробки файлів [3].

Шардінг і реплікація

Масштабування баз даних – одна з найскладніших задач під час росту проекту. Абсолютна більшість усіх зусиль йде якраз на роботу, пов'язану з ростом об'єму даних і операцій з ними. Один сервер бази даних в певний момент перестає справлятись з навантаженням і саме в цей момент варто використовувати описані далі техніки масштабування.

В основі масштабування даних лежить той самий принцип, що і в основі масштабування веб-додатків. Це розділення даних на групи та перенесення їх на окремі сервери. Існує дві основних стратегії – реплікація та шардінг.

Реплікація дозволяє створити повну копію бази даних. Таким чином, замість одного серверу – їх буде декілька.

Частіше всього використовують схему master-slave.

Master – це основний сервер бази даних, куди поступають усі дані. Всі зміни в даних (додавання, оновлення, видалення) повинні відбуватися на цьому сервері.

Slave – це допоміжний сервер бази даних, який копіює всі дані з master-серверу. Звідси потрібно читати дані. Таких серверів може бути декілька.

Реплікація дозволяє використовувати два або більше однакових серверів замість одного. Операцій читання даних, часто, набагато більше, ніж операцій зміни даних. Тому, реплікація дозволяє розвантажити основний сервер за рахунок перенесення операцій читання на slave. В додатку буде два з'єднання з базою даних. Одне – для master, друге для slave-сервера.

Варто відмітити, що реплікація, сама по собі, не дуже зручний механізм для масштабування. Головною причиною цьому служить порушення синхронності даних і затримки в копіюванні з master-сервера на slave. Проте, це чудовий засіб для забезпечення відмовостійкості. Якщо майстер ламається, ви завжди маєте змогу переключитись на slave-сервер і навпаки. Частіше всього, реплікація використовується разом із шардінгом саме з міркувань надійності [4].

Шардінг – це наступна техніка масштабування роботи з даними. Його суть полягає в діленні бази даних на окремі частини так, щоб кожну з них можна було винести на окремий сервер. Цей процес залежить від структури бази даних і виконується прямо у веб-додатку, на відміну від реплікації.

Існує два види шардінгу – вертикальний та горизонтальний.

Вертикальний шардінг – це виділення таблиці чи групи таблиць бази даних на окремий сервер. Наприклад, є такі таблиці: users (дані користувачів), photos (фотографії користувачів), albums (альбоми користувачів). Таблицю users залишаємо на одному сервері, а таблиці photos і albums переміщаємо на інший. В такому випадку, у додатку необхідно використовувати відповідне з'єднання для роботи з кожною

таблицею. На відміну від реплікації, тут використовуються різні з'єднання для будь-яких операцій, але з певними таблицями.

Горизонтальний шардінг – це розподілення однієї таблиці на різні сервери. Це необхідно використовувати для великих таблиць, які не поміщаються на одному сервері. Розділення таблиці на шматки відбувається за наступним принципом:

- На декількох серверах створюється одна і та ж таблиця (тільки структура, без даних).
- У додатку вибирається умова, по якому буде визначатись потрібне з'єднання (наприклад, парні на один сервер, непрані – на інший).
- Перед кожним зверненням до таблиці відбувається вибір потрібного з'єднання.

Припустимо, що наш веб-сервіс працює з великою таблицею, яка зберігає фотографії користувачів. Ми підготували два сервери (зазвичай вони називаються шардами) для цієї таблиці. Для непарних користувачів будемо працювати з першим сервером, а для парних – з другим. Таким чином, на кожному із серверів буде тільки частина усіх даних про фотографії користувачів.

Не варто застосовувати техніку шардінгу до всіх таблиць. Правильний підхід – це поетапний процес розділення зростаючих таблиць. Слід задумуватись про горизонтальний шардінг, коли кількість записів в одній таблиці переходить за межі від декількох десятків, до сотень мільйонів.

Шардінг і реплікація часто використовуються разом одночасно. В нашому прикладі, можна використовувати по два сервери на кожен шард таблиці. Наприклад, photos_master_1 – master першої половини таблиці, photos_slave_1 – slave першої половини таблиці, photos_master_2 – master другої половини таблиці, photos_slave_2 – slave другої половини таблиці.

Така схема часто використовується не для масштабування, а для забезпечення відмовостійкості. Таким чином, при виході з ладу одного з серверів шарду, завжди буде запасний.

Висновок

Шардінг та реплікація – це популярні і потужні техніки масштабування систем роботи з даними. Ці підходи є достатньо універсальними і можуть застосовуватись не тільки в MySQL, але й в багатьох інших технологіях. Реплікація використовується більшою мірою для резервування баз даних і меншою – для масштабування. Master-slave реплікація зручна для розподілу запитів читання по декільком серверам. Часто, реплікація використовується разом із шардінгом при вирішенні питань масштабування. Шардінг являється одним із найпотужніших засобів масштабування бази даних. Горизонтальний шардінг дає змогу розділити таблиці на окремі сервери, що відкриває значні можливості при масштабування системи. Шардінг потрібно впроваджувати поступово і починати тільки із найбільших таблиць. Вертикальний шардінг використовується для розподілу навантаження між групами таблиць.

Процес масштабування даних – це архітектурне рішення, і воно не пов'язане з конкретною технологією. Тому причини проблем зазвичай криються в архітектурі, а не в конкретно вибраній базі даних. З навантаженнями справляються не технології, а архітектура. Тому, якщо ви плануєте розробити додаток, який повинен витримувати високе навантаження, варто наперед проаналізувати усі нюанси, пов'язані з архітектурою системи.

Література:

1. Особенности высоконагруженных сайтов [Електронний ресурс] - Режим доступу http://kharchuk.ru/%D0%A1%D1%82%D0%B0%D1%82%D1%8C%D0%B8/7-PHP%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5/160-highload-sites
2. HighLoad++ для начинающих [Електронний ресурс] - Режим доступу http://highload.guide/blog/highload-for-beginners.html
3. Високонавантаженi системи: рiшення основних проблем [Електронний ресурс] - Режим доступу http://it-ua.info/news/2014/03/26/visokonavantazhenih-sistemi-rshennya-osnovnih-problem.html
4. Оптимизация репликации в Mysql [Електронний ресурс] - Режим доступу https://ruhighload.com/post/%D0%9E%D0%BF%D1%82%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F+%D1%80%D0%B5%D0%BF%D0%BB%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D0%B8+%D0%B2+Mysql