# Вирішення проблеми розширення СУБД в масштабованих системах

**Білан Д.О.,Гніліцька В.В.**
**НТУУ "КПІ", dmytrobilan@aol.com**

***This paper presents and investigates one of the approaches of solving the problem of expanding the databases. This problem is the most critical for large systems that require permanent scaling. The organization of key-value database was chosen as an optimal solution. The most popular products on the market were analyzed. MongoDB was elected.***

***This solution provides the most effective performance at a minimum cost***

## Вступ

Для великих масштабованих систем зі збільшенням завантаженості постійно виникає необхідність їх розширення. Більшість з цих систем використовують СУБД. Найбільш критичним місцем при розширенні системи є збільшення пропускної здатності бази даних. Існує декілька підходів до вирішення проблеми збільшення продуктивності СУБД. Один із них - додати ще один або декілька серверів, з яких додаток може лише зчитувати дані, а записувати їх тільки на перший сервер, котрий у фоновому режимі переносить нові дані на інші сервери. Така архітектура називається майстер-робітник. Інший підхід – майстер-майстер. Він полягає в тому, що існує декілька серверів, на кожному з яких розміщена вся інформація, а додаток випадковим чином (або за певним алгоритмом) вибирає, з яким сервером працювати. Зміни на одному сервері відразу передаються на інші. В даній роботі ми пропонуємо інший підхід – використання key-value (ключ-значення) баз даних.

## Key-value база даних

База даних key-value – це максимально спрощена база даних, тобто сховище, де всі дані зведені до звичайної пари: ключ (індекс) і безпосередньо дані. Таким чином, вся база даних - це список ключів та зіставлених з ними даних. Інтерфейс доступу до такої бази також максимально простий. Зазвичай це найпростіші команди типу: get (отримати дані по ключу), set (записати дані з ключем), delete (видалити ключ і його дані), update (оновити вже існуючі дані). Основною перевагою є те, що у випадку правильного проектування такої бази даних, складність вказаних вище операцій (тобто час обчислення результату) буде заздалегідь відома і не залежить від обсягу даних або кількості серверів. Операції зазвичай атомарні (в SQL базах даних це називається транзакціями). Тобто задаючи команду, можна бути впевненим, що вона або успішно відпрацює, або відразу поверне помилку, і при цьому інші користувачі не завадять виконанню операції, навіть якщо будуть намагатися виконати аналогічну операцію. Це найпростіший тип баз даних ключ-значення.

## Існуючі рішення

Подібних проектів існує багато. Відрізняються вони, як правило, типами даних, можливих для зберігання. Наприклад, окрім рядків можна зберігати числа або двійкові об'єкти (BLOB-и). Також вони можуть відрізнятись кількістю операцій. Описані вище операції є найпростішими; зазвичай підтримується також інкременент та декремент (лічильник в пам'яті), деякі бази даних можуть зберігати масиви та списки. На низькому рівні такі бази будуються на основі хеш-таблиць та їх різновиду - розподілених хеш-таблиць (DHT). Це звичайні об'ємні таблиці, які можуть автоматично розподілятися на будь-яку кількість

робочих станцій і підтримують пошук та отримання інформації, де знаходяться дані. Сильна сторона таких рішень - масштабованість і швидкість.

В рамках роботи були розглянуті наступні реалізації: Memcached (дані зберігаються тільки в пам'яті, що дозволяє здійснювати швидкий доступ, але при цьому необхідно окремо зберігати данні на диск), MongoDB (документо-орієнтована СУБД, що позиціонує себе як проміжна ланка між найпростішими key-value СУБД та реляційними СУБД), Tokyo Cabinet5 (підтримує як зберігання даних в пам'яті, так і на диску; для віддаленого доступу використовується Tyrant6 Tokio). Порівняння СУБД представлено в таблиці 1 нижче.

Таблиця 1 – Порівняння СУБД

| Назва | Memcached | MongoDB | Tokyo |
|---|---|---|---|
| Опер. put/мс | 4,13 | 6,28 | 3,32 |
| Опер. get/мс | 4,72 | 2,95 | 4,45 |
| Розподілен. | + | + | - |
| Реплікація | - | + | - |
| Модель | binary | object | binary |

Оптимальним рішенням було обрано СУБД MongoDB, так як вона проста в налаштуванні, досить надійна і найбільш продуктивна для операції зчитування, яку ми розглядаємо як найбільш критичну. MongoDB підтримує кілька баз даних на одному сервері, а також різні колекції (аналог таблиць в реляційних СУБД) всередині однієї бази даних. Серед основних недоліків можна відмітити відсутність будь-якої авторизації та аутентифікації. MongoDB підтримує зберігання документів в JSON-подібному форматі, має досить гнучку мову для формування запитів, може створювати індекси для різних збережених атрибутів, ефективно забезпечує зберігання великих бінарних об'єктів, підтримує журналювання операцій зі зміною та додаванням даних в базу даних, може працювати відповідно до парадигми Map/Reduce, підтримує реплікацію та побудову відмовостійких конфігурацій. У MongoDB існують вбудовані засоби із забезпечення шардінгу (розподілу набору даних по серверах на основі певного ключа), комбінуючи який реплікацією даних можна побудувати горизонтально масштабований кластер зберігання. В цьому кластері відсутня єдина точка відмови (збій будь-якого вузла не позначається на роботі БД), підтримується автоматичне відновлення після збою і перенесення навантаження з вузла, який вийшов з ладу. Розширення кластера або перетворення одного сервера в кластер проводиться без зупинки роботи БД простим додаванням нових машин.

## Висновки

Використання реляційних СУБД у високонавантажених проектах або сайтах, де необхідно обслуговувати клієнтів без затримки, не є раціональним рішенням. Якщо раніше основні проблеми можна було вирішити кешуванням данних, то сьогодні існують обмеження СУБД. Запропонований підхід дозволяє робити практично будь-які операції, для яких в реляційних СУБД необхідно реалізовувати складні SQL-запити, використовуючи всього п'ять-шість команд.

При побудові масштабованої системи питання вартості розширення зазвичай не постає або ігнорується. Економія часу при проектуванні та під час розробки призводить до втрати часових та фінансових ресурсів. При цьому масштабуються рішення, які не пристосовані для цих завдань.

Наведене рішення дає найбільш ефективну продуктивність при мінімальній вартості впровадження і масштабування системи.

## Література

[1] Production Notes [Електронний ресурс] // Official MongoDB Project Website : [сайт]. [2012]. URL: http://www.mongodb.org/display/DOCS/Production+Notes (дата звернення: 20.02.2012 )

LIST OF NOSQL DATABASES [Електронний ресурс] // NoSQL Databases : [сайт]. [2012]. URL: http://nosql-database.org (дата звернення: 24.02.2012 )