УДК 004.652(045)

Вакуленко М.О.,
Моденов Ю.Б., к.т.н.

ВПРОВАДЖЕННЯ NOSQL-СИСТЕМИ В СУЧАСНУ IT-ІНДУСТРІЮ

Національний авіаційний університет

mashawak@ukr.net
modenov1951@gmail.com

Розглянуто напрямок NoSQL (Not only Structured Query Language), його переваги при проектуванні та реалізації баз даних (БД) порівняно з MySQL. Розглянуто шляхи вирішення проблем з масштабованістю та доступністю, для коректнішої роботи програм і додатків, web-сайтів та інших продуктів, що мають у своїй структурі бази даних. При використанні основних стратегій масштабування: реплікації та шардингу покращуються такі характеристики, як доступність, надійність зберігання даних, зменшення часу обробки інформації

Ключові слова: бази даних, *NoSQL*, *MySQL*, масштабування, реплікація, шардинг, «ключ-значення», переваги, вертикальне масштабування, горизонтальне масштабування, сервер, доступність

Вступ

Бази даних (БД) мають велику сферу застосування, вони допомагають систематизувати і зберігати інформацію з певної предметної області, полегшують пошук, доступ до даних та інше. Сучасні БД працюють з інформацією, представленою у різному форматі: числа, текст, графічні і відеодані. На сьогодні БД використовуються у всіх сферах, де необхідно реалізовувати великі об'єми інформації.

Існують ієрархічні, реляційні, об'єктні, об'єктно-орієнтовані, об'єктно-реляційні, мережеві та функціональні БД. Більшість з них мають такі недоліки: масштабованість; жорстка фіксація складних логічних зв'язків; складність структури; відносна низька швидкість доступу; необхідність великого об'єму зовнішньої пам'яті. Перераховані мінуси систем БД ускладнюють роботу не тільки розробникам, але й завдають незручностей при їх подальшому використанні.

Основними проблемами є недостатня кількість пам'яті для розширення вже існуючих даних, низька швидкість пошуку та виконання запитів при роботі. Так як один сервер не справляється з таким навантаженням.

На даний час напрямок *NoSQL* (*Not only Structured Query Language*) розвивається. Він реалізує БД, що суттєво відрізняються від традиційних реляційних систем БД. Також *NoSQL* застосовується до БД, в яких вирішують проблеми масштабованості та доступності за рахунок атомарності та узгодженості даних.

Аналіз досліджень і публікацій

Термін «*NoSQL*» зародився у червні 2009 року на конференції у Сан-Франциско, на якій планувалося обговорити прогрес у сфері зберігання та обробки даних. Цей акронім був запропонований Еріком Евансом, і не мав глибокого смислового навантаження, але так вийшло, що він поширився у світовій мережі і став назвою цілого напряму в *IT*-індустрії.

За словами Майкла Віденіуса, автора оригінальної версії відкритої системи управління БД *MySQL* і засновника компанії *MySQL AB*, «новий рух *NoSQL*» почався з статті у мережевому журналі від співробітників *Twitter*, які вважали, що *MySQL* не був достатньо підходить для них. Вони потребували «щось краще». Таким чином, основні переваги більшості *NoSQL* рішень це: швидкий доступ до даних (якщо всі дані поміщаються в оперативній пам'яті), швидка реплікація (один

із методів масштабування) шляхом розподілу даних між багатьма вузлами, гнучка схема (можна додавати нові стовпці миттєво).

Мета

Метою статті є продемонструвати можливості масштабування, та його необхідність. Розглянути різні методи масштабування, а саме реплікацію та шардінг. Виділити основні фактори, що впливають на глобальне використання *NoSQL* у більшості сфер *IT*-індустрії і не тільки.

Методи дослідження

Найпростішою базою (сховищем) даних *NoSQL*, з точки зору інтерфейсу прикладного програмування, є база типу «ключ-знання». Тип «ключ-знання» завжди використовує доступ до первинних ключів, він зазвичай має високу продуктивність і легко масштабується.

Масштабування БД – найскладніше завдання під час еволюції проекту. 90% всіх зусиль зазвичай припадає на роботу, пов'язану із зростанням обсягу даних і операцій з ними.

Один сервер БД у деякий момент часу перестає справлятися з навантаженням. У цей момент і слід застосовувати техніки масштабування.

Результати дослідження

Питання підвищення продуктивності роботи програми часто постає при створенні програмного продукту, тому в такий момент і виникає необхідність у масштабуванні баз даних. Проблеми масштабування виникають не відразу – вони з'являються раптово. Якщо до цього бути непідготовленими, то постають великі проблеми.

У наш час збільшення інформації відбувається душе швидко, існує необхідність швидкого її опрацювання. Розвиток у напрямку збільшення швидкості процесора пригальмований на 3,5 ГГц, швидкість читання з диску також зростає малими темпами, а ціна потужного сервера набагато більша, ніж сумарна ціна декількох менш потужних.

Масштабованість – це можливість системи бути працездатною зі збільшенням навантаженнями (зазвичай – шляхом нарощування апаратних джерел).

Масштабування буває вертикальне і горизонтальне.

Вертикальне масштабування (*Vertical scaling*) полягає у збільшенні продуктивності кожної складової системи з ціллю підвищення сумарної продуктивності. У такому разі є можливість змінювати елементи системи на більш потужніші, тому це найпростіший спосіб без необхідності змін у прикладних програмах.

Горизонтальне масштабування (*Horizontal scaling*) – розбитті системи на більш дрібні структурні елементи та розосередження їх по окремим фізичним машинам, і збільшення кількості серверів, які паралельно виконують одну й ту ж функцію. Цей принцип досить простий, а у результаті ми отримаємо підвищення потужності сховища (ємність, пропускна здатність, час відповіді).

Стратегії масштабування: реплікація і шардінг. Шляхом використання процедур шардінгу, реплікації, забезпечується стійкість до відмов системи (результат буде отриманий навіть якщо один або кілька серверів перестали відповідати), перерозподілом даних у разі додавання вузла займається сама *NoSQL* база.

Реплікація

Надійна робота в умовах, коли відмова апаратного забезпечення або мережі неможливі – звичайна справа і одним з властивостей багатьох рішень *NoSQL*.

Основний спосіб її забезпечення є реплікація – це синхронне чи асинхронне копіювання даних між декількома серверами. Реплікація дозволяє створити копію вже наявної БД, через це використовують декілька серверів для цих БД. Таким чином ми досягаємо більшої масштабованості, підвищення доступності та надійного збереження даних.

Існує така модель реплікації, як *Master-Slave*.

Вона має такі компоненти:

1. ***Master*** (мастер) – це головний сервер БД, до якого надходять дані. Тут вони змінюються (видаляються, оновлюються чи поповнюються).

2. ***Slave*** (слейв) – сервер, що копіює всі дані і є допоміжним. Їх може бути декілька.

Найчастіше у схемі реплікації один ***Master*** і декілька ***Slave*** через те що, операцій читання (***SELECT***) більше, аніж запису чи інших модифікацій (***INSERT/UPDATE***).

Але у складнішій конфігурації може бути і більше ніж один ***Master***. Якщо одного ***Slave*** не вистачає – ставиться другий, третій і т.д. Тому існують 3 схеми реплікації:

1. Один майстер, багато слейвів.

2. Ланцюг майстер серверів.

3. Два майстри, багато слейвів.

Крім того, реплікація використовується для географічного розподілу серверів (наприклад один сервер в Америці, інший в Європі).

Принцип реплікації полягає у тому, що ***Master*** сервер при виконанні модифікацій записує всі зроблені зміни у лог-файл (журнал), ***Slave*** сервера з деякою періодичністю перевіряють цей журнал на предмет появи нових даних і якщо вони є – виконують аналогічні дії зі своїми даними.

З точки зору роботи додатку. У додатки ми повинні мати два з'єднання з БД.

```
<?
$master = mysql_connect( '10.10.0.1' , 'root' , 'pwd');
$slave = mysql_connect( '10.10.0.2' , 'root' , 'pwd');
# деяка части програмного коду
$q = mysql_query( 'INSERT INTO users ...' , $master);
# ще деяка части програмного коду
$q = mysql_query( 'SELECT * FROM users WHERE...' , $ slave);
```

При виконанні запитів, потрібні використовувати відповідні з'єднання.

Насправді, реплікація, як і шардінг не є унікальною особливістю ***NoSQL***, але тут, важливу роль відіграють ефективність і легкість внесення змін в існуючу інсталяцію. Перехід БД до роботи в режимі реплікації – це просте завдання для більшості ***NoSQL***-рішень. Якщо звернути увагу на надійність і безпеку БД, то реплікація застосовується зазвичай сумісно з шардінгом.

Шардінг являє собою розбиття даних на групи і зберігання кожної групи на окремому сервері (Шарден). Розподіл повинен бути виконаний так, щоб обов'язково можна було кожну із отриманих груп помістити на сервер. У даному випадку група не обов'язково включає одну таблицю, декілька таблиць можуть містить одне ціле. Шардінг залежить від структури даної бази і виконується він безпосередньо у самому додатку.

Вертикальний шардінг. Реалізується виділенням таблиці або групи таблиць на окремий сервер. Наприклад, в додатку є такі таблиці:

1. ***users*** - дані користувачів;
2. ***photos*** - фотографії користувачів;
3. ***albums*** - альбоми користувачів.

Таблицю ***users*** ми залишаємо на одному сервері, а таблиці ***photos*** і ***albums*** переносимо на інший. У такому випадку в додатку необхідно буде використовувати відповідне з'єднання для роботи з кожною таблицею. На відміну від реплікації, ми використовуємо різні з'єднання для будь-яких операцій, але з певними таблицями.

Горизонтальний шардінг. Здійснюється поділ однієї таблиці на різні сервери. Такий метод використовується для величезних таблиць, які не поміщаються на одному сервері. Поділ таблиці на частини робиться за такими пунктами:

1. На кількох серверах створюється одна і та ж таблиця (тільки структура, без даних).

2. У додатку вибирається умова, за якою визначатиметься потрібне з'єднання.

3. Перед кожним зверненням до таблиці відбувається вибір потрібного з'єднання.

Припустимо, наш додаток працює з величезною таблицею, яка зберігає фотографії користувачів. Ми підготували два сервери для цієї таблиці. Для непарних користувачів ми будемо працювати з першим сервером, а для парних – з другим. Таким чином, на кожному з серверів буде тільки частина всіх даних про фотографії користувачів. У *NoSQL* БД шардінг, як і реплікація, проводяться автоматично самою базою і користувальницький додаток відокремлено від цих складних механізмів. Це ще один плюс *Not Only SQL*.

Висновки

Популярність *NoSQL*-рішень зростає і на це є підстави. По-перше нові завдання з обробки неймовірних обсягів даних, з новими вимогами щодо доступності та масштабованості цих даних. По-друге простота розробки та адміністрування *NoSQL*-технологій. Також важливий цілий ряд завдань, пов'язаних з масштабуванням і реплікацією, що представляють велику складність і вимагають великої спеціальної експертизи на традиційних системах управляння базами даних, у *NoSQL* займає лічені хвилини. Завдання встановлення і налаштування, саме з допомогою *NoSQL*-рішень зазвичай істотно простіше і менш трудомістке, ніж у випадку з реляційними системами баз даних. Тому *NoSQL*-системи стали очевидним вибором для багатьох молодих компаній, де швидкість розробки та впровадження є ключовим фактором.

Не можливо передбачити, що буде новою сходинкою у розвитку напрямку NoSQL, навіть через місяць. І це є приводом до нових досліджень, наукових спостережень, розвитку нових думок на тему «Впровадження *NoSQL*-системи в сучасну *IT*-індустрію».

Список літератури

1. Дж. Садаладж, Макти Фаулер NoSQL: новая методология разработки нереляционных баз данных. : Пер. с англ. - М.: ООО "И.Д. Вильяме", 2013. – 192 с.: ил. - Парал. тит. англ.

2. [Електронний ресурс] Масштабирование баз данных. Режим доступу: http://www.slideshare.net/andrewavdeev3/ss-32639341

3. [Електронний ресурс] Шардинг и репликация. Режим доступу: http://ruhighload.com/index.php/2009/05/06/шардинг-партиционирование-репликац/

4. [Електронний ресурс] Сильные и слабые стороны NoSQL. Режим доступу: http://www.jetinfo.ru/stati/silnye-i-slabye-storony-nosql

Статтю подано до редакції 19.12.2015