УДК 004.658.3(045)

Жуков І.А., д-р техн. наук

Кравець І. М.

ПОСТІЙНІ З'ЄДНАННЯ З БАЗАМИ ДАНИХ, ЯК ОДИН ІЗ МЕТОДІВ РОЗПОДІЛЕННЯ НАВАНТАЖЕННЯ НА WEB-СЕРВЕРАХ

Інститут комп'ютерних технологій
Національного авіаційного університету

Створено кластерну систему, що складається із WEB-серверів, сегментованої бази даних і балансувальника навантаження. Запропоновано модель і специфічну конфігурацію для використання спеціалізованих «проксі-серверів», що суттєво розподіляють навантаження для баз даних за допомогою постійних з'єднань. Експериментальні результати довели ефективність використання постійних з'єднань

Вступ

Постійні з'єднання (ПЗ) є концепцією, яка покликана підвищити продуктивність деяких додатків. Замість з'єднання з базою даних (БД), що встановлюється для кожної операції, воно відкривається один раз і зберігається в "пулі" (*pool*, частині пам'яті) протягом існування додатка. Також ця концепція дозволяє ліквідувати різні види накладних витрат, наприклад, аутентифікація виконується тільки один раз, і тому додатковий мережевий трафік зберігається. Деякі внутрішні структури мають бути виділені для обробки додаткового з'єднання. І на кінець, мали б бути деякі рівні кешування, які можна було б заповнити для належного виконання додатка. ПЗ не надають можливості відкривати "користувальницькі сесії" з використанням того ж самого зв'язку *SQL*, не дають можливості накопичувати транзакції і не дозволяють багатьох інших речей. Точніше сказати, ПЗ не забезпечують ніяких додаткових можливостей у порівнянні з непостійними з'єднаннями. Звісно, виникають запитання: «чому», «для чого» і «як» тоді використовувати ці ПЗ? Перш ніж дати на них відповідь, варто взяти до уваги те, як працюють *WEB*-сервери. Саме вони для генерації *WEB*-сторінок можуть використовувати різні мігруючі програми (*PHP*, *Perl*, *Python*, *ASP*, *Java* та інші), а ті у свою чергу уже взаємодіють із БД через відповідне *API* (від англ. *Application Programming Interface*, прикладний програмний інтерфейс, ППІ).

Взаємодія БД та WEB-серверів з мігруючими програмами

Більшість мігруючих програм (МП) на сьогодні мають у своєму складі модулі (*API*), що дозволяють маніпулювати з БД. Також, деякі із них мають вбудовані функції для створення ПЗ, як це для прикладу в *PHP* реалізується через метод *%DB_NAME%_pconnect()*. Де *%DB_NAME%_* – це назва БД. Скажімо, для *MySQL* це буде *mysql_pconnect()* чи для Oracle – *oci_pconnect()*. Але найбільшою похибкою більшості спеціалістів являється те, що програмна реалізація створення ПЗ не буде виконана. Це все пов'язано з тим, що більшість методів взаємодії тієї чи іншої МП з *WEB*-сервером не передбачають цього. На сьогодні існує безліч типових взаємодій *WEB*-сервера і МП [1]. Найбільшого використання на практиці набули: - взаємодія через *CGI*. При цьому екземпляр інтерпретатора МП створюється і знищується при кожному запиті сторінки у *WEB*-сервера. Тому зрозуміло, що і ПЗ будуть також видалятися з пам'яті [4];

- підключення МП як модуля на багатопроцесному *WEB*-сервері. Якщо в цьому випадку використовується ПЗ, то одному дочірньому процесу досить встановити зв'язок з сервером БД тільки при обслуговуванні першої сторінки, що вимагає з'єднання з БД. Якщо воно знадо биться

іншій сторінці, можна буде використовувати з'єднання, що було встановлене раніше;

- використання МП в якості вбудованого модуля (*plug-in*) для мультипотокового (багатопотокового) *WEB*-сервера. Цей метод має місце для тих МП, у яких є підтримка *ISAPI*, *WSAPI* чи *NSAPI* (у *Windows*). Робота в цьому випадку мало відрізняється від описаної раніше багатопроцесної моделі.

Проаналізувавши ці варіанти, можна зробити висновок, що створення ПЗ чітко залежить від створеної моделі для взаємодії *WEB*-сервера і мігруючої програми. Інколи це виявляється глобальною проблемою коли то й чи інший *WEB*-сервер конфліктує з мігруючою програмою, або тип їхньої взаємодії неможливо змінити. Альтернативою всім цим методам є «проксі-сервера» (ПС), що суттєво абстрагують інтерфейс взаємодії з системою управління БД (СУБД) і створення ПЗ по відношенню до мігруючих програм.

СУБД і «проксі-сервери» – економія часу і ресурсів

На сьогодні ПС відіграють важливу роль в мережі *Internet.* Як правило, вони економлять час який абонент витрачає на очікування відповіді з боку сервера (залежно від часу доби популярні сервери можуть бути переобтяжені) і доставку інформації по глобальній мережі. Крім того, іноді ПС дозволяють отримувати інформацію навіть з віддаленого сервера, недоступного зараз, завдяки тому, що ця інформація була раніше ПС закешована.

По аналогічному принципу існують «проксі-сервери» і для баз даних. В принципі, відомі розробники *СУБД* пропонують кожен свій варіант ПС, що працює тільки з конкретною БД. Наприклад, для *MySQL* існує власний ПС *MySQL Proxy* [8], що дозволяє розподілювати навантаження між різними СУБД якими він керує, аналізувати та фільтрувати запити та багато іншого. Для *Oracle* використовується *Oracle Application Server Proxy Plugin.* Але одним із недоліків є те, що всі запити які поступають на ПС просто транслюються на інші СУБД з метою зменшення навантаження. І після завершення операції обміну даними – з'єднання закривається. Тому в таких випадках ПЗ не існує.

Та і доречнішим буде коли можна повністю абстрагуватися під типу *WEB*-сервера, від мігруючих програм і їх *API,* і саме важливе – від типу БД.

Звісно, що ідеальним рішенням в такій ситуації був би апаратний комплекс, який би мав мінімальні залежності. Але для експериментального досліду можна скористатися безкоштовним програмним комплексом – *SQLRelay* [7]. Це «проксі сервер», що транслює запити для більшості СУБД (від *Sqlite, MySQL* і *PostgreSQL* до *Oracle*). Має засоби балансування навантаження на декількох серверах БД (запити на запис і зміни відзеркалюються).

Основними перевагами є:

1) гнучкі правила для обмеження числа запитів в од. часу і одночасних (корисно для *PostgreSQL*);

2) може виступати як сервер в мережі для *Sqlite* бази;

3) гнучкі правила перенаправлення на потрібну БД (наприклад, після оновлення СУБД деяких клієнтів можна перенаправляти на старий сервер, а останніх на новий);

4) емуляція *API* бібліотек *MySQL* і *PostgreSQL*, підтримка *Perl DBD, Python DB, Ruby DBD, PHP Pear DB.*

ПЗ у традиційній кластер ній системі (реплікації)

Також слід врахувати те, як розподілені дані в БД. Якщо кластерна система ґрунтується на принципі «реплікації» (синхронізація інформації для декількох різних БД), то в цьому випадку слід розглядати модель із «Головної» БД і «Підлеглих» їй БД (рис. 1).

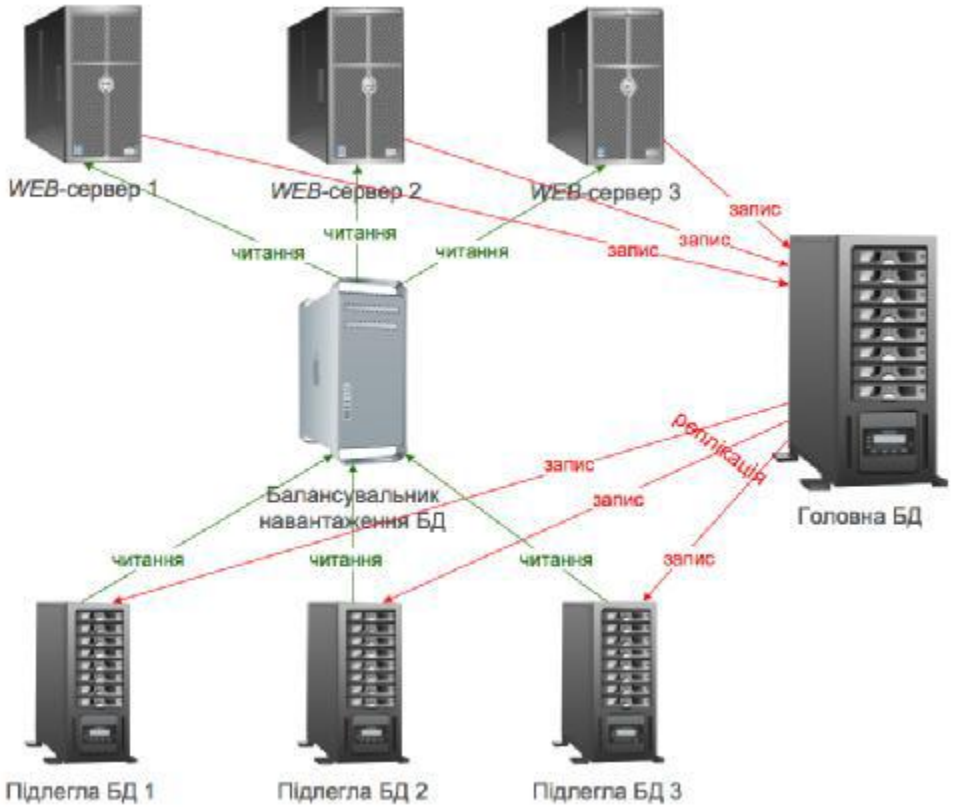

Рис. 1. Традиційна кластерна система, що базується на принципі «реплікації»

Основна проблематика в такій системі полягає в тому, що зазвичай кожна із БД має як мінімум два користувача із різними правами. Це все реалізовується в цілях безпеки, так як виконувати операції на "запис" має право тільки "Головна БД", а "Підлеглі БД" – синхронізуються з нею. Тому щоб уникнути колізій коли один із WEB-серверів спробує зробити операцію на «запис» – варто його обмежити правами WRITE.

Отже задача щодо створення ПЗ має виконуватися обережно і в таких випадках варто проаналізувати чи вона є взагалі доцільною [5].

Виходячи з даної схеми (рис. 1) доречно було б створити ПЗ з БД1-3 тільки для операцій «читання». Але в цьому випадку вся головна біль має лягати на розробника програмного забезпечення що взаємодіє з цими БД. Так як в такій системі саме йому приходиться контролювати в яких діях треба буде робити звернення до «Головної БД» і вносити зміни [6].

Тому, реплікація БД – це не проста задача що вирішується в розробці розподілених систем, і ще складнішою вона стає тоді, коли піднімається питання про постійні з'єднання з БД.

ПЗ у сегментованій БД

Сегментація БД, а саме її розподіл та фрагментація розглянуті у попередніх статтях[1, 2]. Доцільність «еволюційного алгоритму» ще раз підкреслює його ефективність, адже при такій кластерній системі де кожна БД може взаємодіяти в незалежності від іншої завжди буде доречним питання про постійні з'єднання.

Щоб більш об'єктивно оцінити дану проблематику – слід розглянути ось такий приклад: нехай кластерна система складається із трьох WEB-серверів і трьох – серверів БД (рис. 2). На кожен із WEB-серверів буде поступати по 500 клієнтів. Звідси, мінімальна кількість підключень до БД – 1500, або іншими словами – одне з'єднання на кожного клієнта. Тобто, уже із цих простих цифр можна зрозуміти настільки було б доречним ПЗ і яка б була його ефективність, адже не було б потре би виконувати 1500 раз авторизацію на сервері СУБД, вибір тієї чи іншої БД і тд.

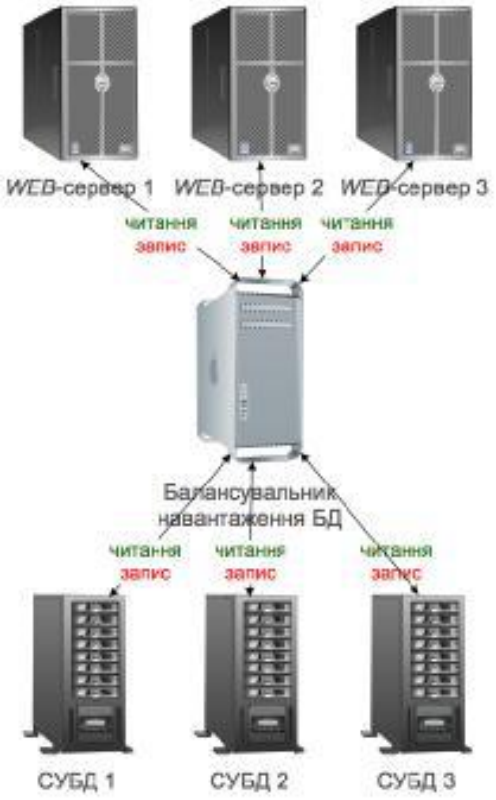

Рис. 2. Кластерна система при сегментованій БД

Експериментальні результати

Для експерименту ПЗ було взято систему, що зображена на рис. 3, де:

- Web-сервер – testweb.ikravets.com;
- «Балансувальник навантаження СУБД» – test-db-balance.ikravets.com;
- «СУБД в ЛМ» (testdblocal.ikravets.com) – це СУБД, що знаходиться в локальної мережі із «Балансувальником навантаження».
- «СУБД за межами ЛМ» (testdbinet.ikravets.com) – це СУБД, що знаходиться за межами локальній мережі від «Балансувальника навантаження».

Фактично для тестування ПЗ не було необхідності для створення широкомасштабної кластерної системи із кількома WEB-серверами та СУБД, оскільки важливим було не розподілення навантаження між ними, а саме, визначення часової затримки із ПЗ і без них. Слід зауважити, що тестування проводилось при двох варіантах підключення СУБД в системі:

1) СУБД знаходилась в одній і тій же локальній мережі, що і “Балансувальник навантаження СУБД”;

2) СУБД знаходилась за межами локальної мережі від “Балансувальника навантаження СУБД”.

Ці два варіанти дали змогу провести більш детальний аналіз щодо того, як впливає мережевий канал зв’язку і його швидкість на з’єднання із СУБД.

Рис. 3. Експериментальна система для тестування ПЗ

Щодо тестового програмного забезпечення: Web-сервер – працював на Apache 2.2.10 із підтримкою PHP5, який підключений як модуль; балансувальник навантаження – SQLRelay; СУБД – системи управління базами даних MySQL 5.1.30. В якості операційної системи було використано CentOS GNU/Linux.

Перш за все було створено скрипт, який був доступний для відкриття із довільного «браузера» через testweb.ikravets.com, також, маніпулював із БД, щоб можна було протестувати його роботу з ПЗ і без них. Алгоритм його роботи зображено на рис. 4:

Рис. 4. Алгоритм роботи WEB-скрипта для test-web.ikravets.com

В принципі, можна було б відмовитись від другого і третього логічних блоків (адже нас тільки цікавили часові характеристики ПЗ) і обмежитися підключенням та відключенням від БД. Але на сьогодні важко уявити на практиці WEB-сервер, який тільки виконує такі операції. Зазвичай за один запит від клієнта до сервера може бути виконано близько 10-20 SQL запитів, місцями і більше. Цей показник залежить від концепції самого WEB-ресурсу. І якщо у нас той чи інший сайт містить багато інформації різного виду, яка теоретично не може бути сумісною в одній таблиці в СУБД, то можна уже робити припущення, що на її отримання були використані різні, а точніше – окремі запити. Все це формально представляє собою своєрідне розгалуження. Тому в даному експерименті обмежились двома запитами, адже нас цікавив коефіцієнт корисної дії від ПЗ не тільки в момент коли сервер знаходився в «холостому ході» (не мав навантаження), а саме тоді, коли він виконував якісь операції, і з яким часовим показником він проводив авторизацію, іншими словами – встановлював з'єднання між мігруючою програмою.

Для цього було створено тестову БД «article_pc» і робочу таблицю «work_table» над якою виконувались 2-3-ій логічні блоки із рис. 4:

mysql> CREATE DATABASE `article_pc`; Query OK, 1 row affected (0.15 sec) mysql> CREATE TABLE `work_table`

(`id` int(11) unsigned NOT NULL auto_increment,`rand` smallint(6) unsigned NOT NULL default '0',`time` int(10) unsigned NOT NULL default '0', PRIMARY KEY (`id`)) ENGINE=MyISAM DEFAULT CHARSET=utf8; Query OK, 0 rows affected (0.00 sec)

WEB-скрипт, index.php, що реалізований мовою PHP і взаємодіяв із SQLRelay, який знаходився на сервері «test-dbbalance.ikravets.com» і мав наступний вигляд:

```
<?php
$connection = sqlrcon_alloc("test-dbbalance. ikravets.com", 9000, "", "sqlr_user", "sqlr_pass", 0, 1);
$cur=sqlrcur_alloc($con);
sqlrcur_prepareQuery($cur, "INSERT INTO work_table SET rand=:rand, time=:time)");
for ($i = 0; $i < 5; $i++)
{sqlrcur_clearBinds($cur);
sqlrcur_inputBind($cur, "rand", rand(0, 10000));
sqlrcur_inputBind($cur, "time", time());
sqlrcur_executeQuery($cur);}
sqlrcon_endSession($con);
sqlrcur_sendQuery($cur, "DELETE FROM work_table ORDER BY RAND() LIMIT 2");
sqlrcon_endSession($con);
sqlrcur_free($cur);
sqlrcon_free($con);
?>
```

Для аналізу часу, що був витрачений на взаємодію із Web-сервером testweb. ikravets.com було використано програмний комплекс "ab" (Apache http server benchmarking tool) [9]. А для побудови графіків на основі вихідних даних від "ab" – Gnuplot. Кількість запитів до Web-сервера при тестуванні (-n requests) за один сеанс – 1000, а кількість одночасних (паралельних) запитів в один і той же момент часу (-c concurrency) – 100. Тобто, нас цікавило як будуть поводити себе сер

вери в системі (рис. 3), коли в один і той же момент часу запити будуть виконувати 100.

Експеримент не постійних з'єднань з СУБД, що знаходиться в локальній мережі

В даному експерименті було налаштовано «Балансувальник навантаження СУБД» «*test-db-balance.ikravets.com*» так, аби при кожному зверненні *WEB*-сервера *test-web.ikravets.com* до СУБД *testdblocal. ikravets.com* він закривав з'єднання одразу після завершення виконання запитів. Тобто, це якраз і було моделлю не постійних з'єднань, так як вони відкривалися і закривалися при кожному сеансі. І це все було зроблено за допомогою додаткових атрибутів *SQLRelay*::*Instances* і *SQLRelay*::*Instances*::*connections* (більш детальний опис атрибутів можна знайти в *DTD* схемі *SQLrelay.dtd*):

- <!***ATTLIST*** *instance connections* ***CDATA*** *"1">* – кількість одночасних ПЗ які необхідно встановити при старті;

- <!***ATTLIST*** *instance maxconnections* ***CDATA*** *"1">* – максимальна кількість одночасних ПЗ які можуть бути встановлені за всю роботу;

- <!**ELEMENT** connection EMPTY> <!***ATTLIST*** *connection metric* ***CDATA*** *"1">* – число, що впливає на те, стільки ПЗ буде створено саме із цією БД.

Загальний конфігураційний файл *SQLRelay* отримав такий вигляд:

```
<?xml version="1.0"?>
<!DOCTYPE instances SYSTEM "sqlrelay.dtd">
<instances>
<instance id="test-db-
balance.ikravets.com" port="9000"
socket="/tmp/mysqltest.socket" dbase="mysql"
connections="0" maxconnections="0">
<users><user user="sqlr_user"
password="sqlr_pass"/></users>
<connections>
<connection connectionid="testlocal"
string="user=root;db=article_pc;host=testdblocal.
ikravets.com" metric="0"/>
</connections>
</instance>
</instances>
```

Експеримент із тестуванням *WEB*-сервера *test-web.ikravets.com* відбувся із такою командою: *"ab -c 100 -n 1000 -g without_pc_local.dat http://test-web.ikravets.com/index.php"*. Після завершення 1000 запитів отримали статистичні дані (табл. 1):

Таблиця 1. Статистичні дані при тестуванні не постійних з'єднань з СУБД, що знаходилась в ЛМ

Параметр	Значення
Загальний час виконання тесту, сек.	5.443
Середня к-сть запитів за 1 сек.	183.72
Середній затрачений час на один запит, мс.	544.311
Середній затрачений час при одночасних запитах, мс.	5.443

На виході із програмного комплексу тестувань "*ab*" отримали *without_pc_local.dat* файл із більш дтальними показниками. За допомогою програмного комплексу "*Gnuplot*" побудували графік результатів (рис. 5):

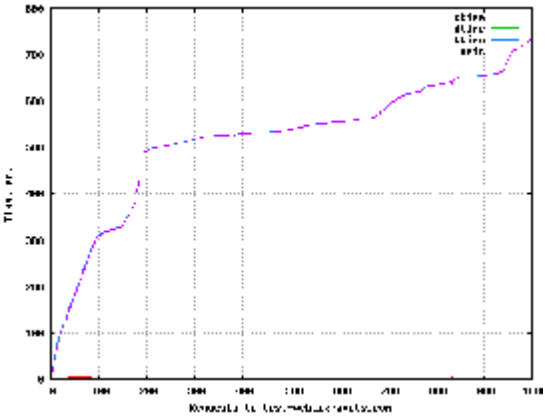

Рис. 5. Результати оброблення запитів при тестуванні не постійних з'єднань з СУБД, що знаходилась в ЛМ

Експеримент постійних з'єднань з СУБД, що знаходиться в локальній мережі

В цьому експерименті було налаштовано «Балансувальник навантаження СУБД» «*test-db-balance.ikravets.com*» так, аби при кожному зверненні *WEB*-сервера *test-web.ikravets.com* до СУБД *test-dblocal.ikravets.com* він навпаки не закривав з'єднання одразу після завершення виконання запитів. Тобто, це якраз і було моделлю постійних з'єднань, так як вони відкривалися і залишалися в «*pool*» (в

пам'яті) для наступних сеансів. Для цього ми встановили *«connections», «maxconnections» і «metric»* еквівалентними одиниці. Це означає, що *SQLRelay* має зберігати одне ПЗ і не більше.

Експеримент із тестуванням *WEB*-сервера *test-web.ikravets.com* відбувся із такою командою: *"ab -c 100 -n 1000 –g with_pc_local.dat http://test-web. ikrvets.com/index.php"*. Після завершення 1000 запитів отримали статистичні дані (табл. 2):

Таблиця 2. Статистичні дані при тестуванні постійних з'єднань з СУБД, що знаходилась в ЛМ

Параметр	Значення
Загальний час виконання тесту, сек.	2.343
Середня к-сть запитів за 1 сек.	426.85
Середній затрачений час на один запит, мс.	234.274
Середній затрачений час при одночасних запитах, мс.	2.343

На виході із програмного комплексу тестувань "*ab*" отримали *with_pc_local.dat* файл із більш детальними показниками. За допомогою програмного комплексу "*Gnuplot*" побудували графік результатів (рис. 6):

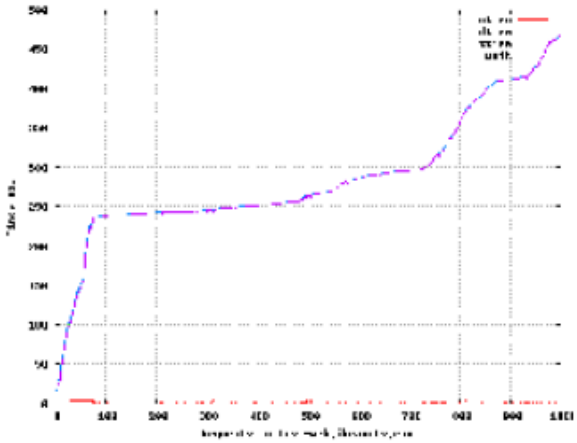

Рис. 6. Результати оброблення запитів при тестуванні постійних з'єднань з СУБД, що знаходилась в ЛМ

Експеримент не постійних з'єднань з СУБД, що знаходиться за межами локальної мережі

Основним фактором в даному експерименті будуть затримки між пересиланням інформації, адже це у більшій мірі буде залежати від навантаження мережі.

Тобто, не можливо пропорційно порівнювати показники між експериментами з СУБД, що розташовані в локальній мережі та за її межами. Але наближену інформацію про завантаженість мережі на маршруті від «Балансувальника навантаження СУБД» «*test-dbbalance. ikravets.com*» до СУБД «*testdbinet. ikravets.com*» ми отримали за допомогою команди *«ping»*:

9 packets transmitted, 9 packets received,0% packet loss

round-trip min/avg/max/stddev = 185.031/188.584/194.864/2.817 ms

Тобто, із відправлених дев'яти пакетів всі повернулись успішно, і середня затримка склала 188.584мс. Цей показник відіграє важливу роль при ПЗ. Щодо налаштування *SQLRelay*, то воно було таким же самим як і при тестуванні ПЗ з СУБД, що знаходилась в локальній мережі. Єдиною відмінністю було те, що змінили "хост"(адрес) сервера СУБД, а саме з *testdblocal. ikravets.com* на *testdbinet. ikravets.com.*

Експеримент із тестуванням *WEB*-сервера *test-web.ikravets.com* відбувся із такою командою: *"ab -c 100 -n 1000 –g without_pc_inet.dat http://testweb. ikrvets.com/index.php "*. Після завершення 1000 запитів отримали статистичні дані (табл. 3):

Таблиця 3. Статистичні дані при тестуванні не постійних з'єднань з СУБД, що знаходилась за межами ЛМ

Параметр	Значення
Загальний час виконання тесту, сек.	1149.898
Середня к-сть запитів за 1 сек.	0.87
Середній затрачений час на один запит, мс.	114989.837
Середній затрачений час при	1149.898
одночасних запитах, мс.	

На виході із програмного комплексу тестувань "*ab*" отримали *without_pc_inet.dat* файл із більш детальними показниками. За допомогою програмного комплексу "*Gnuplot*" побудували графік результатів (рис. 7):

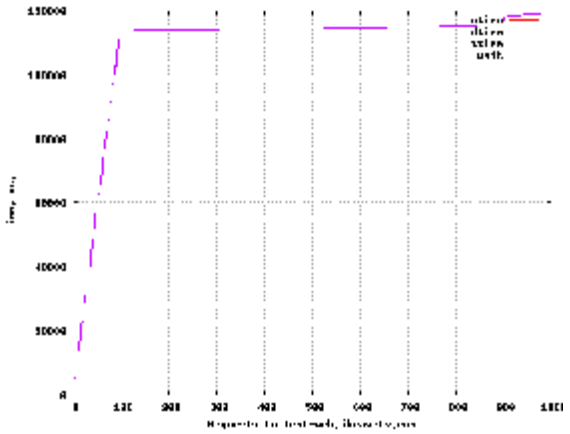

Рис. 7. Результати оброблення запитів при тестуванні не постійних з'єднань з СУБД, що знаходилась за межами ЛМ

Експеримент постійних з'єднань з СУБД, що знаходиться за межами локальної мережі

Очевидно, що в даному експерименті важливу роль відіграє акумуляція ПЗ в пам'яті, так як із попереднього експерименту видно, що основну частину складає час на підключення до СУБД та з'єднання з *WEB*-сервером.

Експеримент із тестуванням *WEB*-сервера *test-web.ikravets.com* та відбувся із такою командою: *"ab -c 100 -n 1000 –g with_pc_inet.dat http://test-web.ikravets.com/index.php "*. Після завершення 1000 запитів отримали статистичні дані (табл. 4):

Таблиця 4. Статистичні дані при тестуванні постійних з'єднань з СУБД, що знаходилась за межами ЛМ

Параметр	Значення
Загальний час виконання тесту, сек.	234.086
Середня к-сть запитів за 1 сек.	4.27
Середній затрачений час на один запит, мс.	23408.613
Середній затрачений час при одночасних запитах, мс.	234.898

На виході із програмного комплексу тестувань "*ab*" отримали *with_pc_inet.dat* файл із більш детальними показниками. За допомогою програмного комплексу "*Gnuplot*" побудували графік результатів (рис. 8):

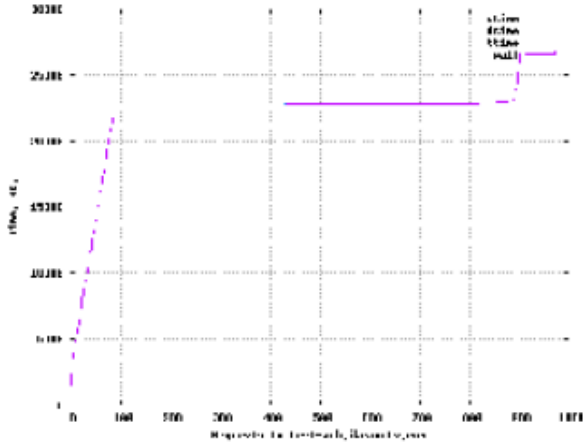

Рис. 8. Результати оброблення запитів при тестуванні постійних з'єднань з СУБД, що знаходилась за межами ЛМ

Висновки

Порівнюючи дані експерименти, можна одразу зробити висновки, що ПЗ мають велику ефективність. Але найбільшого значення вони набувають тоді, коли СУБД знаходиться не в межах локальної мережі. Також слід зауважити, що тестування було проведено при 100 паралельних запитах. На сьогодні на реальних потужних кластерних системах це число сягає кількох тисяч.

При наших експериментах коли сервери знаходились в локальній мережі (рис. 5 і рис. 6) ми бачимо, що при ПЗ СУБД може прийняти в 2.32 рази більше запитів на 1 сек. (544.311 мс / 234.274мс). Причому, якщо глянути на криву, що зображена на рис. 6, при більшій кількості запитів часові показники залишаються незмінними. Тобто, це свідчить про те, що якщо проводити експеримент при більшій кількості паралельних запитів на сервер, ККД від ПЗ буде ще більший, що і доводить ефективність ПЗ.

Але найбільш цікавим був експеримент, коли СУБД знаходилась за межами локальної мережі. Статистичні дані, що приведені в таблицях 3-4, чітко дають зрозуміти, настільки ефективним є ПЗ. Для

того щоб встановлювати з'єднання з СУБД при кожному запиті, «Балансувальника навантаження СУБД» «*test-dbbalance. ikravets.com*» встановив його один раз при першому сеансі і в подальшому акумулював його у пам'яті. Це дало змогу зекономити мережевий трафік між двома серверами і зменшити навантаження на СУБД, адже їй не потрібно було проводити авторизацію на кожному з'єднанні.

В загальному, постійні з'єднання корисні, якщо створювати новий зв'язок з сервером *SQL* важко або накладно. Це залежить від багатьох факторів. Від того, яка використовується СУБД, чи знаходиться вона на тому ж самому комп'ютері, що й *WEB*-сервер, яке завантаження машини, на якій установлено *SQL-сервер* і тд. Висновок: якщо створення нового з'єднання відбувається довго – постійні з'єднання будуть дуже корисним. Вони забезпечують дочірні процеси підключенням до СУБД протягом всього часу їхньої роботи, а не тільки на період обробки сторінки, що вимагає підключення до *SQL-сервера*. Також варто звернути увагу, що при використанні БД з обмеженим числом ПЗ можуть бути незручні. Якщо в БД встановлено обмеження на 13 одночасних з'єднань і в період складного навантаження з'єднання спробують встановити ще 14 дочірніх процесів, одному з них у з'єднанні буде відмовлено. Якщо в скрипті є помилки, що не дозволяють закривати з'єднання (наприклад, нескінченні цикли), доступ до БД з обмеженням в 32 з'єднання може бути дуже швидко заблоковано. Інформацію про обробку зайвих з'єднань наведено в документації по БД.

Постійні з'єднання призначені для забезпечення відповідності "один до одного" з регулярними з'єднаннями. Це означає, що у вас завжди є можливість перейти від постійних з'єднань до не постійних, і це не вплине на роботу скрипта. Ефективність роботи скрипта може змінитися (і, швидше за все, зміниться), але поведінка залишиться незмінною.

Список літератури

1. *Жуков І.А., Кравець І.М.* Методи балансування навантаження для *Web*-серверів // Проблеми інформатизації та управління: Зб. наук. пр.: Вип. 3(21). – К.: НАУ, 2007. – С. 46–54.

2. *Жуков І.А., Кравець І.М.* Розподілення навантаження баз даних в інформаційно-аналітичній системі // Проблеми інформатизації та управління: Зб. наук. пр.: Вип. 4(22). – К.: НАУ, 2007. – С. 56–61.

3. *Ульман Д.* Основы систем баз данных. – М.: Финансы и статистика. – 1983. – 335 с.

4. *Мейер Д.* Теория реляционных баз данных. – М.: Мир.1987. – 608 с.

5. *Fang M.T., Lee R.C.T., Chang C.C.* The Idea of De-Clustering and Its Applications // Proc. 12th Int. Conf. Very Large Data Bases, Kyoto, Japan, Aug. 25-28, 2004. Los Altos, Calif., 2004. – С. 181–188.

6. *Yu C.T., Jiang T.M. Adaptive* Algorithms for Balanced Multidimensional Clustering // Proc. 4th Int. Conf. Data Eng., Los Angeles, Calif., Feb. 2006. Washington, D.C., 2006. – С. 386–391.

7. *http://sqlrelay.sourceforge.net* – persistent database connection pooling, proxying and load balancing system for Unix and Linux.

8. *http://forge.mysql.com/wiki/MySQL_Proxy* – MySQL Proxy is a simple program that sits between your client and MySQL server(s) that can monitor, analyze or transform their communication

9. *http://httpd.apache.org/docs/2.0/programs/ab.html* – tool for benchmarking your Apache Hypertext Transfer Protocol (HTTP) server.