УДК 004.382.2

DOI: http://dx.doi.org/10.20535/2219-380412201543678

А. М. Марусик[1], *бакалавр*

ДИНАМІЧНА СИСТЕМА БАЛАНСУВАННЯ НАВАНТАЖЕННЯ ВЕБ-СЕРВЕРІВ

En

Usage of Internet and frequent accesses of large amount of multimedia data increase the network traffic. Performance evaluation and high availability of server are important factors for resolving this problem using cluster based systems. There are several low-cost servers using the load sharing cluster system which are connected to high speed network, and apply load balancing technique between servers. It offers high computing power and high availability.

[1] *Національний технічний університет України "КПІ", факультет інформатики та обчислювальної техніки*

The overall increase in traffic on the World Wide Web is augmenting user-perceived response times from popular Web sites, especially in conjunction with special events.

A distributed website server can provide scalability and flexibility to manage with growing client demands. To improve the response time of the web server, the evident approach is to have multiple servers. Efficiency of a replicated web server system will depend on the way of distributed incoming requests among these replicas. A distributed Webserver architectures schedule client requests among the multiple server nodes in a usertransparent way that affects the scalability and availability.

The aim of this paper is the development of a load balancing techniques on distributed Web-server systems.

Ru Рассмотрена задача распределения нагрузки в кластере серверов. Предложена система динамической балансировки, что базируется на работе диспетчера и обеспечивает балансировку входной нагрузки на веб-сервер.

Вступ

Розробники інтернет-сервісів часто стикаються із проблемою надмірного навантаження на сервери. Для вирішення цієї проблеми існує декілька рішень. Найбільш очевидним є збільшення потужностей сервера, збільшення смуги пропускання, використання високопродуктивних програмних засобів. Але такий підхід, як правило, є доволі дорогим рішенням на фоні іншого, більш елегантного, а часом і ефективного — розподілення навантаження між декількома серверами. При цьому на кожному із серверів зберігається копія даних. Даний метод має назву балансування.

Існує багато різних алгоритмів і методів балансування навантаження. Вибираючи конкретний алгоритм, потрібно виходити, по-перше, із специфіки конкретного проекту, а по-друге – з цілей, яких планується досягти.

Найпростіший метод балансування навантаження на серверах — *Round Robin DNS* [1]. Метод працює за допомогою управління відповідями *DNS*-сервера у відповідності із деякою статистичною моделлю. В найпростішому випадку *Round Robin DNS* працює, відповідаючи на запити не одною IP-адресою, а списком із декількох адрес серверів, що надають ідентичний сервіс. *Weighted Round Robin* — вдосконалена версія алгоритму *Round Robin*. Кожному серверу присвоюється ваговий коефіцієнт відповідно до його продуктивності і потужності. Це допомагає розподіляти навантаження більш гнучко: сервери з великою вагою обробляють більше запитів.

Least Connection, на відміну від *RR* і *WRR*, згаданих вище, враховує кількість підключень, підтримуваних серверами в поточний момент часу. Кожен наступний запит передається серверу з найменшою кількістю активних підключень [2].

Існує вдосконалений варіант цього алгоритму, призначений в першу чергу для використання в кластерах, що складаються з серверів з різними технічними характеристиками і різною продуктивністю. Він називається *Weighted Least Connections* і враховує при розподілі навантаження не тільки кількість активних підключень, а й ваговий коефіцієнт серверів.

Алгоритм *Destination Hash Scheduling* був створений для роботи з кластером кешуючих проксі-серверів, але він часто використовується і в інших випадках. У цьому алгоритмі сервер, що обробляє запит, вибирається з статичної таблиці за IP-адресою одержувача.

Алгоритм *Source Hash Scheduling* ґрунтується на тих же принципах, що і попередній, тільки сервер, який буде обробляти запит, вибирається з таблиці за IP-адресою відправника.

Sticky Sessions – алгоритм розподілу вхідних запитів, при якому з'єднання передаються на один і той самий сервер групи. Сесії користувача можуть бути закріплені за конкретним сервером за допомогою методу *IP hash*. За допомогою цього методу запити розподіляються по серверах на основі IP-адреси клієнта.

Постановка задачі

У числі цілей, для досягнення яких використовується балансування, потрібно виділити наступні:

- ефективність: всі сервери, які обробляють запити, повинні бути зайняті максимально;
- скорочення часу виконання запиту: потрібно забезпечити мінімальний час між початком обробки запиту (або його постановкою в чергу на обробку) і його завершення;
- скорочення часу відгуку: потрібно мінімізувати час відповіді на запит користувача.

Також, бажано, щоб алгоритм балансування задовольняв такі властивості:

- рівномірне завантаження системних ресурсів;
- передбачуваність: в яких ситуаціях і за яких навантажень алгоритм буде ефективним для вирішення поставлених завдань;
- масштабованість: алгоритм повинен зберігати працездатність при збільшенні навантаження.

Пропонована архітектура

В процесі розробки програмного забезпечення планується реалізувати модель, що буде забезпечувати ефективне розподілення вхідного навантаження за допомогою системи веб-сервера. *HTTP*-запит, що приходить від клієнта, буде розподілений методом диспетчерування *DNS*,

використовуючи алгоритм планування, близький до *Round Robin*. Ми будемо використовувати програмне забезпечення з відкритим програмним кодом, таке, як *BIND* для *DNS*, *JMeter* для генерації запитів клієнтів, код для диспетчера та *Apache Tomcat Web Server* для моделювання.

На основі декількох архітектурних рішень різних моделей і вивчивши їх плюси і мінуси, було розроблено модель, яка має задовольняти наші вимоги, як показано на рис. 1.

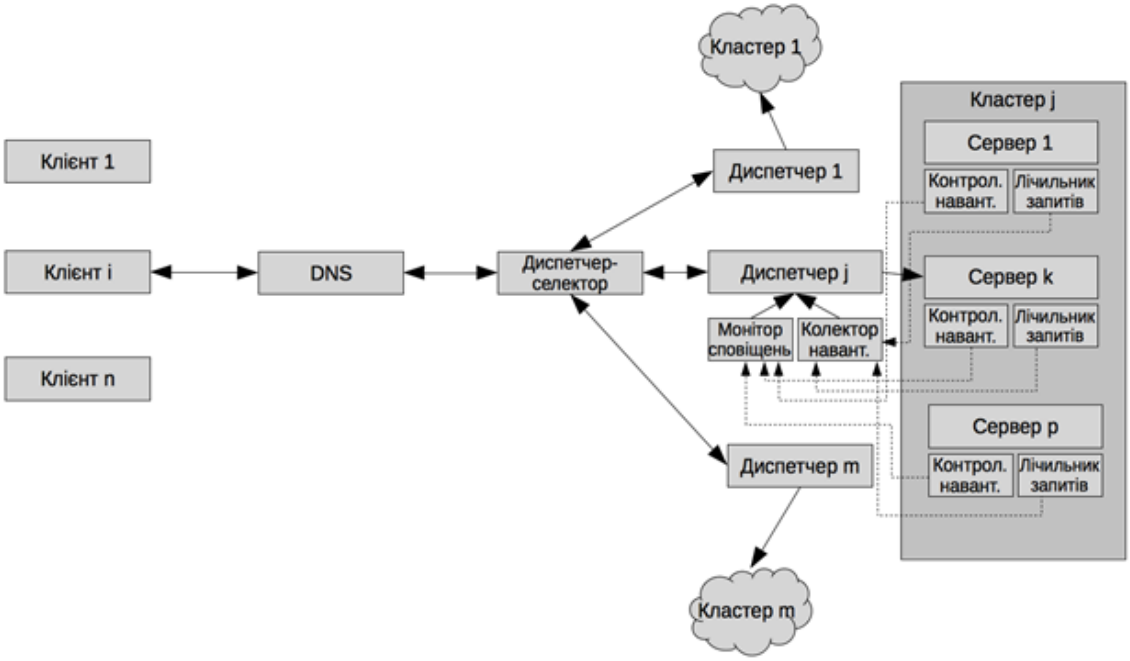

Рис. 1. Архітектурне рішення алгоритму балансування

Розроблена архітектура складається з таких компонентів:

- *DNS*-сервер, диспетчер-селектор,
- *N* диспетчерів (відповідають *N* кластерам, тобто по одному для кожного кластера),
- Колектор навантаження і монітор сповіщень для кожного диспетчера,
- Контролер навантаження і лічильник запитів для кожного сервера.

DNS-сервер єдиний, що спочатку спілкується з клієнтом, він отримує запит і перетворює необхідний *URL* у *IP*-адресу. Оскільки кожен кластер має одного диспетчера, то диспетчер-селектор напряму підключений до кожного. Кожен диспетчер безпосередньо пов'язаний з усіма веб-серверами у своєму кластері. Кожен диспетчер складається з одного колектора навантаження, який збирає інформацію про навантаження кожного сервера. Кожен диспетчер також складається з одного монітора сповіщень, який перевіряє навантаження на кожен сервер і тимчасово припиняє сервіс веб-сервера, якщо навантаження стає зависоким. Тоді як кожен сервер складається з контролера навантаження і лічильника запитів, які розраховують і передають інформацію про навантаження на сервері.

Реалізація та тестування

Алгоритм пропонованої системи є розкинутим по різних модулях, які включені в загальну архітектуру, що складається з диспетчера, сервера і клієнтів. Архітектура взмозі визначити найбільш оптимальний сервер для конкретного клієнта на основі різних параметрів, таких як час відгуку, завантаження центрального процесора (ЦП), пропускна здатність та ширина смуги. Модулі забезпечують:

- Створення архітектури за участю клієнта, сервера і диспетчера.
- Розрахунок часу відгуку всіх задіяних серверів.
- Розрахунок завантаження процесора для всіх задіяних серверів.
- Розрахунок пропускної здатності всіх задіяних серверів.
- Відправка всіх допустимих розрахункових даних диспетчеру, який виступає в якості системи, що приймає рішення і вирішує який сервер найкраще підходить для запитів клієнтів.

Завантаження процесора є одним з тих факторів продуктивності, який водночас грубо недооцінений і переоцінений. Часто здається, що більша частка тих, хто усвідомлює значення завантаження, турбується про цей фактор занадто багато. Як і в більшості проблем з продуктивністю, клопотатися із-за незначних відмінностей в числах, як правило, безглуздо; немає великого значення, якщо завантаження процесора становить 5% або 10%; але якщо це 80% або 90%, то вплив на юзабіліті системи буде значним. Ще одне ключове питання полягає в тому, що швидкі диски передають більше даних, і більша кількість даних - при інших рівних - потребує більше часу на обробку. Це абсолютно неприпустимо при порівнянні використання процесора і дисків різних поколінь без поправки на цю важливу деталь. У цій системі ми використовували процесор як фактор для прийняття рішення щодо завантаженості сервера. Нижче наведені основні кроки:

- Диспетчер є вирішальним компонентом системи, що опитує кожен з серверів для оптимального їх використання.
- Кожен з серверів розраховує завантаження процесора і відправляє його диспетчеру.
- Весь процес використовує концепцію *RMI* (виклик віддалених методів, *Remote Method Invocation*).
- Ключова формула, що описує процес:

Час відгуку характеризує продуктивність індивідуальної транзакції або запиту. Час відгуку, як правило, розглядається як час, що минув з моменту, коли користувач вводить команду або активує функцію до часу, коли застосунок показує завершення операції. Час відгуку для типового динамічного сервера застосунків включає в себе наступну послідовність дій. Кожна дія потребує певної кількості часу. Один сервер буде обслуговувати одного клієнта протягом певного періоду часу і тому ми

повинні обчислити пропускну здатність, тобто кількість успішних запитів, відправлених у певний період часу.

Обрахунок пропускної здатності включає наступні етапи:

- Призначається арбітражний період часу в 2 секунди для розрахунку пропускної здатності;
- Диспетчер записує початковий час і починає посилати запити до сервера, а разом з тим обчислює кількість успішних відповідей;
- В кінці 2 секунд кількість успішних відповідей обраховується, що є нашою пропускною здатністю.

Функція і формула, яка використовується для розрахунку пропускної здатності:

- Береться цикл від 1 до 1000000;
- Викликається довільна функція на сервері;
- Записується кількість успішно встановлених з'єднань;
- Розділяється кількість успішних підключень, щоб отримати пропускну здатність.

Apache JMeter може бути використаний для тестування продуктивності як статичних, так і динамічних ресурсів (файли, сервлети, *Perl* скрипти, *Java* об'єкти, бази і запити даних, *FTP*-сервери тощо). Він може бути використаний для імітації високого навантаження на сервер, мережу або об'єкт, щоб перевірити надійність або проаналізувати загальну продуктивність при різних видах навантаження. Це можна використовувати для створення графічного аналізу продуктивності або перевірки сервера (об'єкта) скрипта під сильним одночасним навантаженням.

План тестування

Для перевірки моделі використано підхід розподіленого тестування від *JMeter*. Підготовлено одну *master*-систему і кілька *slave*-систем. План тестів розроблений на головній системі таким чином, що кілька потоків (користувачі) генеруються на кожну *slave*-систему на задану команду, запитану через майстер-систему. Кожен потік може відправляти різні типи запитів (в нашому випадку це *HTTP*-запит) до системи, що тестується (в даному випадку це система веб-серверів). Таким чином, створюється навантаження на конкретну систему.

Тестування розробленої системи проходило згідно розробленого плану, який складається з одної майстер-системи та трьох підпорядкованих, і створено в цілому 17000 потоків (користувачів) задля створення навантаження і оцінки продуктивності системи. Ось характеристики розробленого плану тестування:

- Кількість *master*-систем: 1;
- Кількість *slave*-систем: 3;

- Кількість згенерованих потоків (користувачів): 17000;
- Ім'я сервера або *IP*: *www.webservice.com.*

Процедура тестування і результати

Щоб перевірити модель за вище розробленим планом випробувань, необхідно запустити тест у двох конфігураціях і отримати результати. Ми отримали результати у наступних формах: графік пропускної здатності, відсоток помилок у відповідях від веб-сервера, середній час відгуку системи і її графік. Ці результати будуть порівняні у цьому розділі. Було зроблено тести у двох станах і їх результати такі:

1. *Тест простої конфігурації* (*без балансування навантаження*)

У цьому стані виконується перевірка просто на одному веб-сервері *Apache Tomcat* зі встановленим на ньому простим *HTML* сайтом. Це найпростіша з можливих конфігурацій без реалізації балансування навантаження. Ось результати:

Параметри	Значення
Кількість оброблених запитів	14257
Середній час відповіді	2500 мс
Відсоток помилок	22.34
Пропускна здатність	52.6/сек

2. *Тест в умовах архітектури*

У цьому стані виконується перевірка системи веб-сервера з нашої архітектури на кластері, який має два веб-сервери *Apache Tomcat* з встановленими на них простими *HTML* сайтами. Ось результати:

Параметри	Значення
Кількість оброблених запитів	16442
Середній час відповіді	521 мс
Відсоток помилок	14.87
Пропускна здатність	59.0/сек

Висновок

Вивчення класичних підходів для балансування навантаження дозволило розробити власну модель, яка об'єднала в собі підхід *DNS* і підхід, що оснований на диспетчері. Взяті до уваги параметри використання процесора і мережевого трафіку та вибрано потрібний веб-сервер для обслуговування запиту.

Тестування моделі було здійснено в середовищі реального часу в лабораторії з використанням засобів, таких як *JMeter*. Основною метою моделі є мінімізація втрат від найгіршого сценарію, щоб уникнути втрати запиту до сервера і, врешті, падіння сервера. Для цієї мети врахували використання процесора і мережевий трафік, але є кілька інших параметрів, які не можуть бути розглянуті, наприклад, відсутність активних *TCP*/*IP* з'єднань і встановлення ліміту максимального використання кластеру.

Пропускну здатність мережі та близькість веб-сервера також слід вважати важливими аспектами. Крім того, важливого значення можна надати поліпшенню пропускної здатності і часу відгуку веб-серверів. Ця сфера має великий потенціал і складає важливий аспект розподіленої системи. Балансування навантаження є концепцією, яка все ще перебуває в стані досліджень. Кожен день розробляються нові алгоритми, а існуючі моделі вивчаються. Звідси видніється великий простір для подальшого вдосконалення.

Список використаної літератури

1. *Чжоу Т.* Системы балансировки нагрузки Web-серверов [Електронний ресурс] / Тао Чжоу // Windows IT Pro. – 2000. – Режим доступу до ресурсу: http://www.osp.ru/win2000/2000/03/174228/.
2. *Емельянов А.* Балансировка нагрузки: основные алгоритмы и методы [Електронний ресурс] / Андрей Емельянов // blog.selectel.ru. – 2015. – Режим доступу до ресурсу: http://blog.selectel.ru/balansirovka-nagruzki-osnovnye-algoritmy-i-metody/.