К. П. Сторчак, А. П. Бондарчук, Д. Є. Василенко, О. М. Шушура, О. А. Золотухина

ФОРМАЛІЗАЦІЯ ЗНАНЬ ПРО ЗАВДАННЯ УПРАВЛІННЯ ПОВІТРЯНИМ РУХОМ ДЛЯ ПЕРСПЕКТИВНИХ СИСТЕМ УПРАВЛІННЯ ПОВІТРЯНИМ РУХОМ

Розвиток перспективних систем управління повітряним рухом потребує вивчення нових підходів до проектування і створення спеціального математичного та програмного забезпечення. Одним із перспективних напрямків удосконалення спеціального математичного забезпечення є використання методів штучного інтелекту. Для розробки баз знань і даних таких систем необхідно провести аналіз і вибір методу формалізації знань. У статті обгрунтовано вибір методу формалізації знань із урахуванням особливостей розв'язання задач управління повітряним рухом.

Ключові слова: управління повітряним рухом; штучний інтелект; формалізація завдань; АСУ; управління; прийняття рішень.

K. P. Storchak, A. P. Bondarchuk, D. E. Vasilenko, O. M. Shushura, O. A. Zolotukhina

FORMALIZATION OF KNOWLEDGE THE TASKS OF AIR TRAFFIC CONTROL SYSTEMS FOR ADVANCED AIR TRAFFIC CONTROL

Development of advanced air traffic control systems requires the development of new approaches to the design and development of special mathematical and software. One of the promising ways to improve special software is to use the methods of artificial intelligence. To develop the knowledge base and data of such systems should be analyzed and the choice of method of formalization of knowledge. We justify the choice of the method of formalization of knowledge allowing for the solution of air traffic control problems.

Keywords: air traffic control; artificial intelligence; the formalization of tasks; automation; management; decision-making.

УДК 004.052

О. О. ІЛЬЇН[1], доктор техн. наук, професор;
Д. С. КОВАЛЕНКО[2], Back-end розробник;
М. П. ГНІДЕНКО[3], канд. техн. наук, професор;
Ю. В. БЕРЕЗОВСЬКА[1],
[1,3] Державний університет телекомунікацій, Київ
[2] ТОВ «Techmedia4u», Київ

Деякі практичні аспекти реалізації мультипрограмного вирішення балансування навантаження на сервери інформаційного ресурсу

Розглянуто традиційні підходи щодо балансування навантаження на сервери в мережі Інтернет, виокремлено ситуації, в яких вони здатні вирішувати виникаючі проблеми з навантаженням та ситуації, що потребують пошуку інших шляхів, зокрема таких, які полягають у комбінації наявних вирішень, додатковому налаштуванні апаратної та програмної складових серверної інфраструктури, створенні спеціалізованих додатків. На прикладі ситуації, параметри якої задано конкретним технічним завданням, проведено експериментальне дослідження і розроблено вирішення, протестоване за реальних умов та прийняте до експлуатації замовником.

Ключові слова: навантаження; сервер; протокол; трафік; балансування; навантажувальні тести; PHP; JavaScript; web socket.

ВСТУП

З кожним роком спостерігається значне зростання кількості користувачів всесвітньої мережі Інтернет, що призводить до збільшення навантаження на веб-сайти та інші сервіси (у подальшому — сервери), які працюють у всесвітній мережі. Навантаження на сервери, як правило, спричинене великим мережним трафіком, тобто потоком запитів від користувачів та зворотних відповідей від серверів. Згідно зі статистикою в Україні за період із 2000 по 2013 рік зросла кількість користувачів, а отже, збільшився і потік даних (рис. 1). Зазвичай цей потік зростає у наближеній геометричній прогресії відносно кількості користувачів, уповільнюючи роботу серверів, через що клієнти можуть, у найкращому разі, не дочекатись повного завантаження сторінки веб-сайта, а у гіршому — втратити кошти через помилку транзакції через перевантажений, наприклад, банківський сервіс. Тому існує великий попит на різні вирішення, які дають можливість знизити навантаження на сервери, забезпечуючи функціонування сервісів.

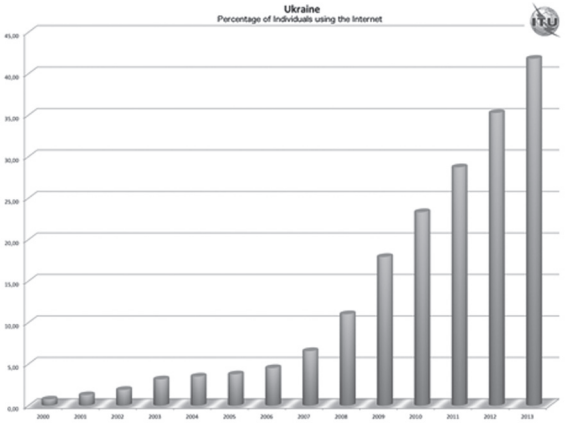

Рис. 1. Діаграма зміни чисельності інтернет-користувачів в Україні протягом 2000–2013 років [1]

Традиційні вирішення щодо управління навантаженням на сервери, які забезпечують функціонування серверів у мережі Інтернет, базуються на архітектурі, наведеній на рис. 2.

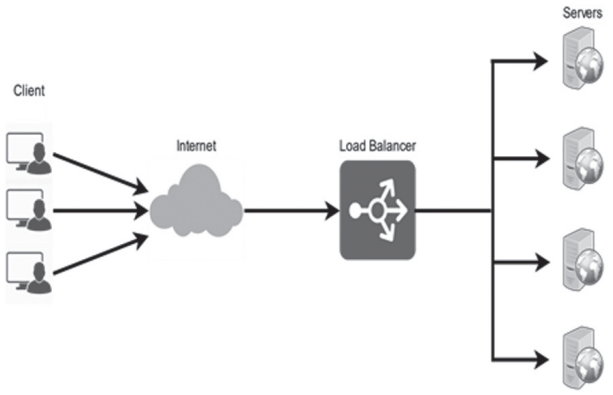

Рис. 2. Загальна архітектура системи балансування навантаження

Клієнтський запит потрапляє через мережу Інтернет не навпрямки на сервер, а на точку входу, на якій розміщено так званий балансувальник — окремий сервер або програмний процес, що перенаправляє запит. Завдання балансувальника — знайти найменш завантажений у цей час сервер і перенаправити запит на нього. У зворотному напрямі інформація буде йти від цього сервера до користувача. Реалізація балансування навантаження сервера може забезпечити такі переваги [2]:

• ***масштабованість:*** продуктивність інформаційного сервісу більше не залежатиме від продуктивності одного сервера, на якому цей сервіс розгорнуто; можливість реалізувати структуру plug-and-play, потенційно використовуючи фізичні ресурси більшої кількості менш потужних серверів;

• ***надійність:*** незалежно від того, наскільки надійним є один сервер, збій на одному сервері призведе до порушення роботи та повного вимкнення всього сервісу; розподілення навантаження на низку фізичних серверів значно зменшує цей вплив;

• ***роботоздатність і технічне обслуговування:*** під час очікування доступності 24 на 7 реалізація балансування навантаження на сервері може допомогти у створенні середовища, в якому сервери можуть бути видалені з експлуатації для планового технічного обслуговування — оновлення операційної системи, оновлення апаратних засобів тощо; уможливлюється розгортання нових версій і збірок програмних додатків, зберігаючи контроль над процесом переходу на нову версію додатків.

Водночас застосування системи балансування навантаження має свої недоліки. Для неї потрібні додаткові технічні засоби, зокрема додаткові фізичні сервери, та спеціальні налаштування програмної та апаратної інфраструктури, що можуть бути доступні лише адміністраторам серверів. Для підтримання роботи цих систем необхідно залучати додатковий вузькоспеціалізований персонал, що може бути економічно невигідно для проектів тощо.

Безумовно, додаткові вимоги до системи балансування накладають і умови технічного завдання конкретного проекту, у межах якого буде використовуватись дана система. Якщо орієнтуватись на реальні проекти та узагальнити ці вимоги, то основними з них є такі:

1. Вирішення щодо балансування навантаження має розгортатись на наявних серверах.

2. Програмна частина вирішення не має вимагати додаткового апаратного забезпечення.

3. Програмна частина вирішення не має вимагати спеціальних налаштувань середовища, окрім можливих стандартних.

4. Технічним підтриманням вирішення можуть займатись ті самі спеціалісти, які підтримують і сайт/сервер.

Такий стан речей надає підстави порушити питання щодо створення нової системи балансування навантаження на сервери, яке задовольнятиме вимоги реальних проектів. Також вирішення має базуватись на відомих відкритих алгоритмах, бути простим стосовно налаштування та не потребувати додаткових апаратних засобів. Вирішення може включати в себе додаткові програмні компоненти, зокрема написані з урахуванням конкретних вимог технічного завдання.

ОСНОВНА ЧАСТИНА

Загальні архітектурні вирішення балансування навантаження на сервер та межі їх застосування

Зазвичай існуючі архітектурні вирішення балансування реалізуються на рівнях 2, 3, 4 і 7 моделі OSI [2]. Більшість топологій балансування навантаження на сервер може належати до одного з трьох типів.

Рівень 2 або SLB перехід, де клієнт або клієнтський маршрутизатор і сервери існують в одній LLAN 2 VLAN і підмережі рівня 3 [3] (рис. 3).

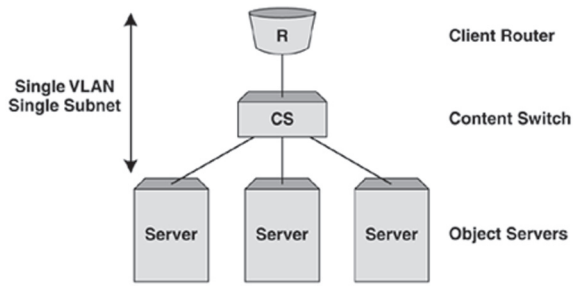

Рис. 3. Структурно-логічна схема топології SLB на рівні 2 [3]

Single Arm SLB, де перемикач вмісту додається через один логічний інтерфейс до комутатора рівня 2 або рівня 3 [3] (рис. 4).

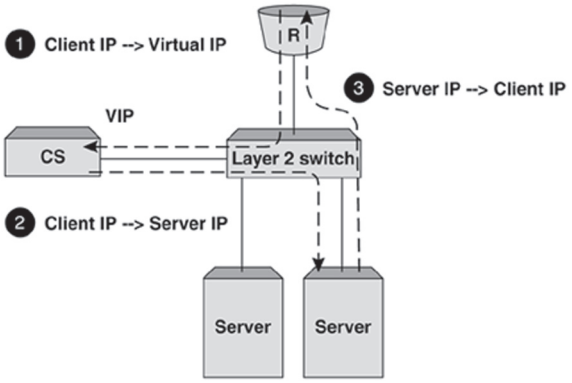

Рис. 4. Структурно-логічна схема топології Single Arm SLB [3]

Рівень 3 або SLB маршрутизація, де клієнт (або клієнтський маршрутизатор) і сервери існують у різних мережах VLAN 2 або підмережах рівня 3 [3] (рис. 5).

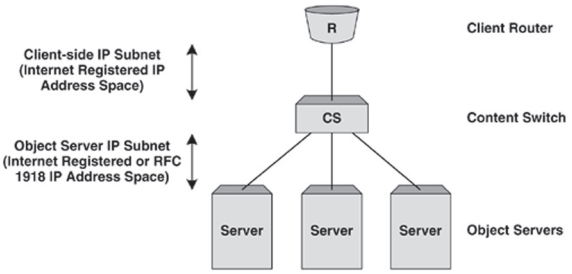

Рис. 5. Структурно-логічна схема топології SLB маршрутизації на рівні 3 [3]

На рівні 4 моделі OSI присутні власні алгоритми балансування навантаження на сервер. Усі ці алгоритми є сеансовими. Вони базуються на таких метриках [3]:

- ***Least Connections*** — найпростіший алгоритм, розподіляє навантаження на основі кількості сесій за одиницю часу;
- ***Round Robin*** — алгоритм рівномірного розподілу навантаження між усіма доступними серверами;
- ***IP Address Hashing*** — хешування використовується для позначення детермінованого алгоритмічного розрахунку, заснованого на інформації IP-адреси, що міститься в сеансі TCP, або потоку UDP, і саме така уніфікованість дає цю метричну IP-адресу на основі персистенції;
- ***Response Time ma Server Agent*** — алгоритми, в яких використовуються показники часу відповіді, що базуються на механізмі перевірки роботоздатності;
- ***Bandwidth*** — метрика, що базується на моніторингу пропускної здатності, яка має відношення до конкретної MAC-адреси сервера і вимірюється протягом певного часу;
- ***Weighting and Maximum Connections*** — призначення «ваги» сервера призведе до зсуву потоку трафіку в бік більших або потужніших реальних серверів на основі визначеного значення, зазвичай це ціле число в певному діапазоні.

Також балансування навантаження може проводитись на **рівні 7** моделі OSI. Балансування відбувається за допомогою перемикання змісту, тобто перенаправлення веб-кешу. Типові задачі щодо балансування навантаження на рівні 7 [3]:

- **розбір URL-адреси HTTP** — використання місцезнаходження та сторінки, що запитуються через HTTP;
- **перевірка заголовка HTTP** — використання іншої інформації в HTTP-запиті, наприклад мови або типу браузера;
- **файли cookie HTTP** — читання, перезапис і вставлення файлів cookie HTTP для збереження та пільгових послуг користувачів;
- **розбір FTP** — переклад інформації в FTP-керуючий канал для забезпечення правильної роботи при реалізації балансування навантаження сервера;
- **синтаксичний аналіз DNS** — використання запитаного домену або імені хоста для прийняття рішення про передачу DNS-серверу;
- **розбір потоку RTSP** — ідентифікація відео або аудіопотоку, запитаного користувачем, і обслуговуючий вміст від правильного ресурсу.

З огляду на загальні вимоги, визначені для реальних проектів, балансування на рівнях 2 і 3 є небажаним через потребу в додаткових технічних засобах. Це означає, що розглянуті топології немає сенсу реалізовувати. Але самі схеми топологій можуть бути використані для загального уявлення того, який урешті-решт має вигляд остаточне вирішення.

Балансування на рівні 7 за допомогою наявних технологій є досить сумнівним вирішенням, оскільки крім збалансування навантаження між кількома серверами, також потрібно вирішити завдання з виконання асинхронних запитів до віддаленого API (на вимогу конкретного проекту). Але сам варіант реалізації балансувальника на рівні 7 не відкидається.

Найбільш суперечливим варіантом є SLB на рівні 4 за одним із запропонованих алгоритмів. Подані алгоритми вже реалізовано в більшості наявних веб-серверів і використання одного з них залежить лише від часу та необхідного налаштування. Але оскільки перемикання алгоритмів на сервері здійснюється через конфігурації, то для доступу до глобальних конфігурацій веб-сервера потрібно мати права «суперадміністратора», яких клієнт найчастіше не має. Постачальник послуг хостингу навряд чи погодиться змінити конфігурацію веб-сервера, оскільки це може зашкодити іншим клієнтам, які також розміщують свої проекти на цьому сервері. Звісно постачальнику легше відмовити одному клієнтові, ніж розбиратись із проблемами всіх користувачів на конкретному сервері.

З огляду на зазначені проблеми у роботі запропоновано реалізувати один із алгоритмів рівня 4, але на рівні 7 із віртуалізацією топології Single Arm SLB рівнів 2 і 3. Для цього увагу зосереджено на принципах роботи алгоритмів рівня 4 — Least Connections і Round Robin через їх простоту та можливість відносно швидко реалізувати на рівні 7.

Особливості реалізації системи балансування навантаження на базі алгоритмів Round Robin і Least Connections

Технічне завдання містить набір вимог, деякі з яких такі: поєднати процеси реєстрації, авторизації та автентифікації з клієнтською базою через зовнішній API. Також завдання полягає в реалізації передбачених документацією API сценаріїв за допомогою мов PHP і JavaScript, щоб створити веб-інтерфейс для реєстрації, авторизації та автентифікації клієнтів мережі компанії на її сайті, із потенційним використанням в інших сумісних проектах цієї самої компанії. Головна складність у реалізації цього інтерфейсу полягає в тому, що API має асинхронний потік передавання інформації.

Відповідно до наданих вимог було запропоновано дизайн власного програмного вирішення для балансування навантаження на сервер, яке включає в себе браузерну (front-end) та серверну (back-end) частини (рис. 6). У front-end частині розміщено клієнтський додаток, написаний мовою програмування JavaScript. У серверній — балансувальник та вибірка фонових процесів, написані мовою програмування PHP. Зв'язок між клієнтом, балансувальником та фоновими процесами здійснюється за допомогою протоколу WebSocket.

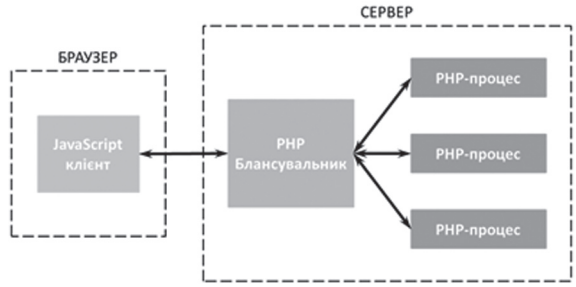

Рис. 6. Дизайн запропонованого вирішення балансування навантаження

На основі запропонованого дизайну було створено програмне вирішення, яке покладено в основу системи для роботи з базою користувачів через віддалений API сервер. Архітектуру системи у вигляді структурно-логічної схеми роботи вирішення показано на рис. 7.

Система складається з клієнтського додатку, сервера з балансувальником і підсистемою фонових процесів та віддаленого API сервера.

Процес роботи даної системи полягає ось у чому. Клієнт відправляє запит на сервер через протокол WebSocket. На сервері запит потрапляє до балансувальника, який розділяє його на два HTTP-запити, що різняться за методом відправлення — GET і POST. Ці запити балансувальник направляє до процесів у двох пулах PHP-процесів, між якими розподіляються всі вхідні запити. Ці процеси надсилають запити до API та очікують на них відповіді. Після отримання відповідей процеси надсилають їх до балансувальника, задача якого об'єднати їх у єдину відповідь та надіслати її клієнтові. Отже, одразу видно ефективність вирішення, оскільки вдалось синхронізувати роботу

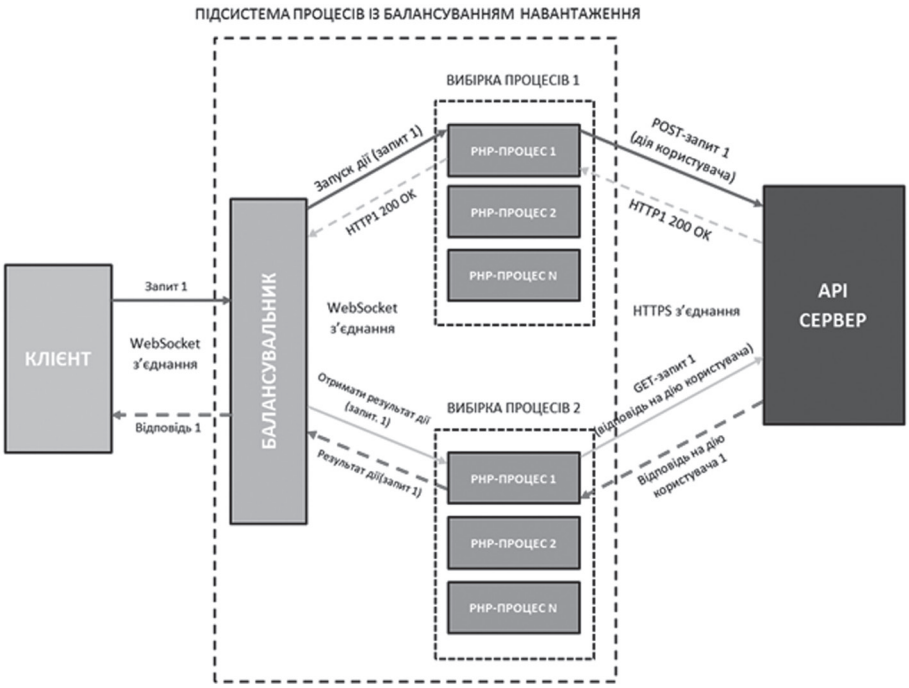

Рис. 7. Структурно-логічна схема асинхронного сервера з вбудованим балансувальником навантаження

сервера сайта, який за принципом роботи є синхронним, із сервером API, який згідно з технічним завданням є асинхронним.

Таким чином, без розробленої системи балансування навантаження від одного клієнта надсилалося б одразу два запити, замість одного, тобто навантаження на канал зв'язку між клієнтом і сервером зменшилось на 50%.

Тестування системи та обговорення результатів

Основним завданням тестування системи є з'ясування максимального навантаження, яке вона здатна витримати без втрат. Під навантаженням мається на увазі кількість одночасних сесій, тобто користувачів системи. Щоб оцінити таке навантаження, скористаємось тестами пропускної здатності [4]. Тести пропускної здатності виконувалися за допомогою додатку Apache JMeter, інструменту для проведення навантажувального тестування, розробленого Apache Software Foundation [5]. Тестовий сервер мав 64 Гбайт оперативної пам'яті та два 16-ядерних процесори Intel Xenon. Тестування здійснювалось у два етапи для 100, 1000 і 10 000 користувачів без обмеження за часом.

Методика тестування полягала у такому. На першому етапі перевірявся один тип запиту-відповіді під час роботи із сервісом. Програма відсилала 100, 1000 і 10 000 запитів на ініціалізацію процесу авторизації. Зі 100 запитами система впоралась менш ніж за 15 с, 1000 запитів було оброблено менш ніж за 25 с, 10 000 запитів система змогла обробити за 60 с (див. таблицю). Під час проведення тестування спостерігалось підвищене споживання оперативної пам'яті порівняно зі споживанням без сервісу авторизації, що відчутно помітно вже на 1000 запитах.

Результати проведення навантажувальних тестів

Етапи тестування	Кількість запитів	Час виконання, с	Навантаження на процесор, %	Витрати оперативної пам'яті, Гбайт (макс. 64 Гбайт)
1	100	15	12	0,25
	1000	25	14	1,5
	10 000	60	21	2,8
2	100	140	18	1,7
	1000	235	25	3,6
	10 000	840	34	13,4

На другому етапі JMeter проходив повний цикл авторизації, що містив послідовність кількох різних типів запитів-відповідей роботи із сервісом. Із 100 повними циклами авторизації система впоралась за 140 с, 1000 циклів було оброблено за 3 хв та 55 с, 10 000 циклів оброблювались протягом 14 хв, при цьому втрати — запити, на які не було відповіді або було отримано помилку, становили 7% від загальної кількості запитів. У процесі проведення тестування рівень споживаної оперативної пам'яті зріс ще більше порівняно з першим етапом. Моніторинг серверних ресурсів показав, що на момент перевірки 10 000 циклів запитів-відповідей рівень споживаної пам'яті становив 20-25% від загального обсягу.

ВИСНОВКИ

Сьогодні існує велика кількість вирішень для балансування навантаження на сервер. У статті розглянуто основні топології, алгоритми та технології балансування, які, у свою чергу, використовуються як основа більш спеціалізованих вирішень. Специфіка кожного конкретного завдання не дає можливості повноцінно використати одне або просте поєднання кількох готових вирішень, але відкриває простір для роздумів, аналізу та розробки можливо не нового, але і нетривіального технічного вирішення.

За результатами огляду можливих архітектур балансування та з огляду на загальні вимоги реальних проектів було обрано шлях розроблення виключно програмного вирішення, який би задовольняв усі висунуті вимоги щодо умов експлуатації, а також поєднував у собі переваги розглянутих у цій статті алгоритмів та технологій. Розроблене вирішення дає можливість розподіляти створене клієнтськими запитами навантаження безпосередньо на транспортному рівні моделі OSI. Використання технології WebSocket відкриває можливість опрацьовувати весь вхідний трафік в асинхронному режимі, при цьому не змушуючи користувача очікувати на відповідь сервера. Більш специфічні аспекти проекту, зокрема блокувальні запити, спонукали створити дві окремі вибірки воркерів (пулів php-процесів). При цьому кількість воркерів у кожній вибірці не обмежується двома або трьома. Серед переваг розробленого вирішення є і таке, що його виконано за допомогою базових ресурсів мови PHP без додаткових налаштувань та розширень, що значно спрощує налаштування та технічне підтримання під час експлуатації.

Список використаної літератури

1. ***Інтернет*** *в Україні [Електронний ресурс]. URL:*

https://uk.wikipedia.org/wiki/Інтернет_в_Україні

2. ***Олифер В. Г., Олифер Н. А.*** *Компьютерные сети. Принципы, технологии, протоколы: учеб. для вузов, 5-е изд. Питер, 2016. 992 с.*

3. ***Syme M., Goldie P.*** *Optimizing Network Performance with Content Switching: Server, Firewall and Cache Load Balancing. Pearson Education, 2004. 259 p.*

*4. **Тестування** та якість ПО [Електронний ресурс]. URL: http://www.software-testing.ru*
*5. **Офіційна** документація Apache JMeter [Електронний ресурс]. URL: https://jmeter.apache.org/*

Рецензент: доктор техн. наук, професор **В. В. Вишнівський**, Державний університет телекомунікацій, Київ.

О. А. Ильин, Д. С. Коваленко, Н. П. Гниденко, Ю. В. Березовская

НЕКОТОРЫЕ ПРАКТИЧЕСКИЕ АСПЕКТЫ РЕАЛИЗАЦИИ МУЛЬТИПРОГРАММНОГО РЕШЕНИЯ БАЛАНСИРОВКИ НАГРУЗКИ НА СЕРВЕРЫ ИНФОРМАЦИОННОГО РЕСУРСА

Рассмотрены традиционные подходы к балансировке нагрузки на серверы в сети Интернет, выделены ситуации, в которых они способны решать возникающие проблемы с нагрузкой, и ситуации, которые требуют поиска других путей, в том числе таких, которые предполагают комбинацию существующих решений, дополнительную настройку аппаратной и программной составляющих серверной инфраструктуры, создание специализированных приложений. На примере ситуации, параметры которой заданы конкретным техническим заданием, проведено экспериментальное исследование и разработано решение, которое было протестировано в реальных условиях и принято к эксплуатации заказчиком.

Ключевые слова: нагрузка; сервер; протокол; трафик; балансировка; нагрузочные тесты; PHP; JavaScript; web socket.

O. O. Ilin, D. S. Kovalenko, M. P. Hnydenko, Yu. V. Berezovska

PRACTICAL ASPECTS OF IMPLEMENTING A MULTI-PROGRAM SOLUTION FOR BALANCING THE LOAD ON AN INFORMATION RESOURCE SERVER

The article discusses traditional approaches to balancing the load on servers on the Internet. Situations in which they are able to solve emerging problems with the load, situations that require the search for other ways are highlighted. Complex solutions that involve a combination of existing solutions, additional configuration of the hardware and software components of the server infrastructure, and the creation of specialized applications are considered. On the example of a situation, parameters of which are set by a specific technical task, an experimental study was carried out and a solution was developed. The solution was subsequently tested in real conditions and accepted for use by the customer.

Keywords: load; balancing; server; protocol; load tests; PHP; JavaScript; web socket.