МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Литвиненко О.Є.

«___» _____2020 р.

ДИПЛОМНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ "МАГІСТР"

Тема: Методи агрегації інформації профілів соціальних мереж

Виконавець: Панченко І.О..

Керівник: Артамонов Є.Б..

Нормоконтролер: Тупота Є.В.

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління
Освітнього ступеня магістр
Спеціальність 123 "Комп'ютерна інженерія"
(шифр, найменування)
Спеціалізація 123.02 "Системне програмування"
(шифр, найменування)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Литвиненко О. Є.

«____»_____2020 р.

ЗАВДАННЯ
на виконання дипломної роботи (проекту)

Панченка Іллі Олександровича
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема роботи: Методи агрегації інформації профілів соціальних мереж

затверджена наказом ректора від "07" вересня 2020 року № 1410 /ст.
2. Термін виконання роботи: з 05.10.2020 до 31.12.2020
3. Вихідні дані до роботи: нейронна система для реалізації модклі спільної фільтрації
4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):
1) аналіз формальних понять;
2) методи побудови системи визначення користувацьких переваг;
3) проектування нейронної мережі визначення користувацьких переваг.
5. Перелік обов'язкового графічного матеріалу:

6. Календарний план

№ п/п	Етапи виконання дипломної роботи	Термін виконання етапів	Примітка
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

7. Дата видачі завдання 05.10.2020

Керівник Артамонов Є.Б..
(підпис)

Завдання прийняв до виконання Панченко І.О.
(підпис студента)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи “Методи агрегації інформації профілів соціальних мереж”: 88 с., 30 рис., 22 літературних джерел, 1 додаток.

КОРИСТУВАЦЬКА ПЕРЕВАГА, ГРУПОВА ФІЛЬТРАЦІЯ, НЕЙРОННА МЕРЕЖА, АНАЛІЗ ПОПИТУ, ВЕБ-ПОРТАЛ.

Мета дипломної роботи – аналіз методів визначення користувацьких вподобань для впровадженні на сайті рекомендаційної системи.

Об'єкт дослідження – визначення користувацьких вподобань.

Предмет дослідження – програмний засіб агрегації інформації профілів соціальних мереж.

Практична значимість полягає у розробці програмного забезпечення, що планується використовувати на веб-русурсах для формування списку пропоозицій за вподобаннями користувачів на основі аналізу їх профілів у соціальних мережах.

Публікації: Панченко І.О., Голего Н.М. Програмний засіб автоматичного визначення вподобань відвідувачів веб-порталів// Тези доповідей наук.-практ. конф. “Сучасні тенденції розвитку системного програмування” (25-26 листопада 2020 р.). – К.: НАУ, 2020. – С. 25.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

ВСТУП

Завдання рекомендаційної системи - проінформувати користувача про товар, який йому може бути найбільш цікавий в даний момент часу. Клієнт отримує інформацію, а сервіс заробляє на наданні якісних послуг. Послуги - це не обов'язково прямі продажі пропонованого товару. Сервіс також може заробляти на комісійних або просто збільшувати лояльність користувачів, яка потім виливається в рекламні та інші доходи.

Залежно від моделі бізнесу рекомендації можуть бути його основою, як, наприклад, у TripAdvisor, а можуть бути просто зручним додатковим сервісом (як, наприклад, в якомусь інтернет-магазині одягу), покликаним поліпшити Customer Experience і зробити навігацію по каталогу більш зручною.

Персоналізація онлайн-маркетингу - очевидний тренд останнього десятиліття. По оцінкам Маккінсі, 35% виручки Amazon або 75% Netflix припадає саме на рекомендовані товари і відсоток цей, ймовірно, буде рости. Рекомендаційні системи - це про те, що запропонувати клієнту, щоб зробити його щасливим.

Щоб проілюструвати все різноманіття рекомендаційних сервісів, наведу список основних характеристик, за допомогою яких можна описати будь-яку рекомендаційну систему.

1. предмет рекомендації- що рекомендується. Тут велика різноманітність - це можуть бути товари (Amazon, Ozon), статті (Arxiv.org), новини (Surfingbird, Яндекс.Дзен), зображення (500px), відео (YouTube, Netflix), люди (Linkedin, LonelyPlanet), музика (Last.fm, Pandora), плейлисти та інше. В цілому, рекомендувати можна що завгодно.

2. мета рекомендації- навіщо рекомендується. Наприклад: покупка, інформування, навчання, заклад контактів.

3. контекст рекомендації- що користувач в цей момент робить. Наприклад: дивиться товари, слухає музику, спілкується з людьми.

4. джерело рекомендації- хто рекомендує: аудиторія (середній рейтинг ресторану в TripAdvisor), - схожі за інтересами користувачі, - експертне співтовариство (буває, коли мова про складне товар, такому, як, наприклад, вино).

5. ступінь персоналізації. Неперсональное рекомендації - коли вам рекомендують те ж саме, що всім іншим. Вони допускають таргетинг по регіону або часу, але не враховують ваші особисті переваги.

Більш просунутий варіант - коли рекомендації використовують дані з вашої поточної сесії. Ви подивилися кілька товарів, і внизу сторінки вам пропонуються схожі.

Персональні ж рекомендації використовують всю доступну інформацію про клієнта, в тому числі історію його покупок.

6. прозорість. Люди більше довіряють рекомендації, якщо розуміють, як саме вона була отримана. Так менше ризик нарватися на «недобросовісні» системи, які просувають проплачений товар або ставлять більш дорогі товари вище в рейтингу. Крім того, хороша рекомендаційна система сама повинна вміти боротися з купленими відгуками і накрутками продавців.

Маніпуляції до речі бувають і ненавмисними. Наприклад, коли виходить новий блокбастер, насамперед на нього йдуть фанати, відповідно, першу пару місяців рейтинг може бути сильно завищений.

7. формат рекомендації.

Це може бути спливаюче віконце, що з'являється в певному розділі сайту відсортований список, стрічка внизу екрана або щось ще.

8. алгоритми.

Незважаючи на безліч існуючих алгоритмів, всі вони зводяться до декількох базових підходів, які будуть описані далі. До найбільш класичним відносяться алгоритми Summary-based (неперсональное), Content-based (моделі засновані на описі товару), Collaborative Filtering (коллаборативна фільтрація), Matrix Factorization (методи засновані на матричному розкладанні) і деякі інші.

У центрі будь-рекомендаційної системи знаходиться так звана матриця переваг. Це матриця, по одній з осей якої відкладені всі клієнти сервісу (Users), а

по інший - об'єкти рекомендації (Items). На перетині деяких пар (user, item) дана матриця заповнена оцінками (Ratings) - це відомий нам показник зацікавленості користувача в даному товарі, виражений за заданою шкалою (наприклад від 1 до 5).

Веб-сервери використовуються для надання різноманітних послуг або як частина системи для моніторингу та адміністрування пристроїв, баз даних, архівів, складних системних служб та локальних мереж тощо, всього, що є або може бути підключене до мережі. Вони настільки популярні, оскільки на комп'ютері клієнта не потрібно встановлювати додаткове програмне забезпечення, оскільки воно працює через веб-браузер.

З ростом арифметичної прогресії користувачів, зростання веб-трафіку до серверів популярних служб, які обробляють та генерують динамічний вміст, почав реагувати з помітною затримкою або просто не спрацьовувати.

Таким чином, питання ефективності веб-платформ під великим навантаженням є актуальним на сьогоднішній день, і його слід розглядати з різних можливих точок зору.

Веб-додатки з великим навантаженням - це сайти з великою аудиторією і, як наслідок, з великим навантаженням, що вимагає оптимізованого фонового режиму (типові системи управління веб-контентом є дуже функціональними та хорошими для відносно невеликих сайтів завантаження) [2]. Різницю між веб-додатком із високим навантаженням та ненавантаженням можна описати як різницю між вагоном та швидкісним поїздом (вони обидва доставляють вміст, але кількість залучених програмістів, інженерів та адміністраторів та принцип реалізації різні) . Очевидними прикладами дуже завантажених веб-сайтів є: Google, Facebook, Flickr, Amazon, Twitter, Youtube тощо. Але існує також ряд невідомих для очей громадськості серверів із високим навантаженням, наприклад ігрові сервери, сервери dns та антивірус. сервери баз даних. Всі вони мають різні цілі та зміст, але мають подібні проблеми із високим навантаженням.

Не існує чіткої (формальної) межі між системою з високим навантаженням і не з великим навантаженням. Одне з пояснень полягає в наступному: проект можна розглядати як високонавантажувальний, якщо він був побудований за

допомогою нетривіальних технічних рішень для задоволення потреб поточної або найближчої майбутньої аудиторії.

З технічної точки зору не можна сказати, що таке застосування з високим навантаженням [2], оскільки немає твердих цифр. Але можна вказати на деякі фактори:

Висока щоденна кількість звернень;

Високе пікове навантаження (короткочасне навантаження, наприклад, вечірні години);

Неможливість кешування сторінок через складні бізнес-логіки;

Великий інтерактивний проект (форуми, блоги, журнали);

Величезні обсяги даних;

Недостатньо ресурсів для задоволення будь-якого з вищезазначених факторів.

Веб-платформи, що використовуються у проектах із великим навантаженням, вибираються індивідуально та ретельно відповідно до потреб, але також використовуються деякі загальні пакети. Це пакет програмного забезпечення LAMP (Linux, Apache, MySQL / PostgreseSQL та PHP / Python / PERL) та стеки Microsoft / Oracle. Більшість сучасних мов програмування та супутніх технологій мають усі необхідні інструменти для побудови проекту зі стабільною, безпечною та масштабованою архітектурою. Це не обмежується вищезазначеними продуктами, оскільки їх можна назвати тривіальними, які згодом перекодуються та переконфігуруються зі зростанням навантаження. Але навіть за допомогою цих найпростіших програмних комплексів навантаження на систему можна зменшити за допомогою кешування на різних рівнях.

У більшості випадків вплив на проект починається тоді, коли початкової (тривіальної) реалізації функціональних можливостей недостатньо, і розробникам потрібно впроваджувати нові або нетривіальні технічні рішення для підтримки потреб зростаючої аудиторії. Оптимізація високого навантаження - це плавний процес, який виконується разом із зростанням навантаження.

Незалежно від вибору веб-платформи виникають проблеми із великим навантаженням - недостатня кількість ресурсів порівняно з обсягом, який

потрібно обробити, хоча питання стабільності продуктивності при великих навантаженнях все ще залишається відкритим.

У проекті із великим навантаженням виникають дві проблеми: масштабованість та надійність [2]. Масштабованість - це здатність системи обробляти зростаючий обсяг роботи здатним способом або її здатність розширюватися, щоб пристосувати цей ріст. Система називається масштабованою системою, якщо продуктивність покращується після модифікації (оновлення) [2, 6]. Надійність або збій в роботі вважається вирішеною, коли відмова будь-якого сервера не спричиняє фатального впливу на систему в цілому [2].

Мета дипломної роботи – аналіз методів визначення користувацьких вподобань для впровадженні на сайті рекомендаційної системи.

Об'єкт дослідження – визначення користувацьких вподобань.

Предмет дослідження – програмний засіб агрегації інформації профілів соціальних мереж.

Практична значимість полягає у розробці програмного забезпечення, що планується використовувати на веб-русурсах для формування списку пропоозицій за вподобаннями користувачів.

Публікації за темою роботи: Панченко І.О., Голего Н.М. Програмний засіб автоматичного визначення вподобань відвідувачів веб-порталів// Тези доповідей наук.-практ. конф. “Сучасні тенденції розвитку системного програмування” (25-26 листопада 2020 р.). – К.: НАУ, 2020. – С. 25.

РОЗДІЛ 1
АНАЛІЗ МЕТОДІВ ВИЗНАЧЕННЯ КОРИСТУВАЦЬКИХ ПЕРЕВАГ

Користувачі зазвичай оцінюють лише невелику частину товарів, що є в каталозі, і завдання рекомендаційної системи - узагальнити цю інформацію і передбачити ставлення клієнта до інших товарів, про які нічого не відомо. Іншими словами потрібно заповнити всі незаповнені клітинки на зображенні вище.

Шаблони споживання у людей різні, і не обов'язково повинні рекомендуватися нові товари. Можна показувати повторні позиції, наприклад, для поповнення запасу. За цим принципом виділяють дві групи товарів.

– повторювані. Наприклад, шампуні або бритвені верстати, які потрібні завжди.

Неповторювані. Наприклад, книги або фільми, які рідко набувають повторно.

Якщо продукт не можна явно віднести до одного з класів, має сенс визначати допустимість повторних покупок індивідуально (хтось ходить в магазин тільки заради арахісового масла певної марки, а кому-то важливо спробувати все, що є в каталозі).

Поняття «цікавинки» теж суб'єктивне. Деяким користувачам потрібні речі тільки з їхньої улюбленої категорії (conservative recommendations), а хтось, навпаки, більше відгукується на нестандартні товари або групи товарів (risky recommendations). Наприклад, відеохостинг може рекомендувати користувачеві тільки нові серії улюбленого серіалу, а може періодично закидати йому нові шоу або взагалі нові жанри. В ідеалі варто вибирати стратегію показу рекомендацій під кожного клієнта окремо, за допомогою моделювання категорії клієнта.

Призначені для користувача оцінки можна отримати двома способами:

– явно (explicit ratings) - користувач сам ставить рейтинг товару, залишає відгук, лайкать сторінку,

– неявно (implicit ratings) - користувач явно своє ставлення не виражає, але можна зробити непрямий висновок з його дій: купив товар - значить він йому подобається, довго читав опис - значить є інтерес і т.п.

Звичайно, явні переваги краще - користувач сам говорить про те, що йому сподобалося. Однак на практиці далеко не всі сайти надають можливість явно виражати свій інтерес, та й не всі користувачі мають бажання це робити. Найчастіше використовуються відразу обидва типи оцінок і добре доповнюють один одного.

Також важливо відрізняти терміни Prediction (прогноз ступеня інтересу) і власне Recommendation (показ рекомендації). Що і як показувати - це окреме завдання, яка використовує отримані на кроці Prediction оцінки, але може бути реалізована по-різному.

Іноді термін "рекомендація" вживають в ширшому сенсі і мають на увазі будь-яку оптимізацію, будь то вибірка клієнтів для рекламної розсилки, визначення оптимальної ціни пропозиції або просто вибір найкращої стратегії комунікацій з клієнтом. У статті я обмежуся класичним визначенням даного терміна, що означає вибір найбільш цікавого товару для клієнта.

1.1. Основні можливості неперсоналізованих рекомендацій

Почнемо з неперсоналізовані рекомендацій, оскільки вони найпростіші в реалізації. У них потенційний інтерес користувача визначається просто середнім рейтингом товару: «Усім подобається - значить сподобається і вам». За цим принципом працює більшість сервісів, коли користувач не авторизується в системі, наприклад, той же TripAdvisor.

Показуватися рекомендації можуть по-різному - як банер збоку від опису товару (Amazon), як результат запиту, відсортоване за певним параметру (TripAdvisor), або якось ще.

Середній рейтинг від покупців також може зображуватися різними способами. Це можуть бути зірочки поруч з товаром, кількість лайків, різниця

позитивних і негативних голосів (як зазвичай роблять на форумах), частка високих оцінок або взагалі гістограма оцінок. Гістограми - найбільш інформативний спосіб, але у них є один мінус - їх складно порівнювати між собою або сортувати, коли потрібно вивести товари списком.

Холодний старт - це типова ситуація, коли ще не накопичено достатню кількість даних для коректної роботи рекомендаційної системи (наприклад, коли товар новий або просто його дуже рідко купують). Якщо середній рейтинг пораховано за оцінками всього трьох користувачів (igor92, xyz_111 і oleg_s), така оцінка явно не буде достовірною, і користувачі це розуміють. Часто в таких ситуаціях рейтинги штучно коректують.

Перший спосіб - показувати не середнє значення, а згладжені середнє (Damped Mean). Сенс такий: при малій кількості оцінок відображається рейтинг більше тяжіє до нікому безпечного «середнього» показнику, а як тільки набирається достатня кількість нових оцінок, «усереднюються» коригування перестає діяти.

Інший підхід - розраховувати по кожному рейтингу інтервали достовірності (confidence Intervals). Математично, чим більше оцінок, тим менше варіація середнього і, отже, більше впевненість в його правильності. А в якості рейтингу можна виводити, наприклад, нижню межу інтервалу (Low CI Bound). При цьому зрозуміло, що така система буде досить консервативною, з тенденцією до заниження оцінок з нових товарів (якщо, звичайно, це не хіт).

Оскільки оцінки обмежені певною шкалою (наприклад від 0 до 1), звичайний спосіб розрахунку інтервалу достовірності тут погано застосуємо: через хвостів розподілу, що йдуть на нескінченність і симетричності самого інтервалу. Є альтернативний і більш точний спосіб його порахувати -Інтервал довіри Вільсона. При цьому виходять несиметричні інтервали приблизно такого вигляду.

1.2. Сучасні виклики систем рекомендацій

Проблема рекомендування елементів користувачам полягає у передбаченні значень невідомих записів на основі значень відомих записів.

– Два класи систем рекомендацій: Ці системи намагаються передбачити відповідь користувача на елемент шляхом виявлення подібних елементів та відповіді користувача на них. Один клас системи рекомендацій ґрунтується на вмісті; він вимірює подібність, шукаючи загальні риси предметів. Другий клас системи рекомендацій використовує спільну фільтрацію; вони вимірюють схожість користувачів за їх уподобаннями щодо товарів та / або вимірюють схожість елементів за користувачами, які їм подобаються.

– Профілі елементів: Вони складаються з особливостей предметів. Різні види предметів мають різні особливості, на яких може базуватися схожість на основі вмісту. Особливості документів - це, як правило, важливі або незвичні слова. Вироби мають такі атрибути, як розмір екрану для телевізора. Такі засоби масової інформації, як фільми, мають жанр та деталі, такі як актор чи виконавець. Теги також можна використовувати як функції, якщо їх можна отримати від зацікавлених користувачів.

– Профілі користувачів: Система спільної фільтрації на основі вмісту може будувати профілі для користувачів, вимірюючи частоту, з якою функції відображаються в елементах, які сподобаються користувачеві. Потім ми можемо оцінити ступінь сподобання елементу користувачеві за тісністю профілю елемента до профілю користувача.

– Класифікація предметів: Альтернативою побудові профілю користувача є побудувати класифікатор для кожного користувача, наприклад, дерево рішень. Рядок матриці утиліти для цього користувача стає навчальними даними, і класифікатор повинен передбачити відповідь користувача на всі елементи, незалежно від того, чи був у рядку запис для цього елемента.

– Подібність рядків і стовпців матриці службових програм : Спільні алгоритми фільтрування повинні вимірювати подібність рядків та / або стовпців

матриці корисності. Відстань Жаккарда доречна, коли матриця складається лише з одиниць і пропусків (для "не оцінено"). Відстань косинусів працює для більш загальних значень у матриці корисності. Часто корисно нормалізувати матрицю корисності, віднімаючи середнє значення (або за рядком, за стовпчиком, або за обома) перед вимірюванням відстані косинуса.

– Кластеризація користувачів та предметів: Оскільки матриця корисності, як правило, є більшою пробіли, міри відстані, такі як жаккард або косинус, часто мають занадто мало даних, з якими можна порівняти два рядки або два стовпці. Попередній крок або етапи, на яких подібність використовується для кластеризації користувачів та / або елементів у невеликі групи із сильною подібністю, може допомогти забезпечити більш загальні компоненти, з якими можна порівняти рядки або стовпці.

– УФ-розкладання: Один із способів передбачити порожні значення в утиліті матриця - це знайти дві довгі, тонкі матриці U і V, добуток яких є наближенням до даної матриці корисності. Оскільки матриця продукту

дає значення для всіх пар елементів-користувачів, це значення можна використовувати для прогнозування значення порожнього місця в матриці утиліти. Інтуїтивна причина, за якою цей метод має сенс, полягає в тому, що часто виникає відносно невелика кількість проблем (ця кількість є «тонким» виміром U та V), які визначають, подобається користувачеві чи ні.

– Помилка середньоквадратичного значення : Хороший показник того, наскільки близький продукт УФ є для даної матриці корисності RMSE (середньоквадратична помилка). RMSE обчислюється шляхом усереднення квадрата відмінностей між УФ та матрицею корисності в тих елементах, де матриця корисності не пуста. Квадратним коренем цього середнього є RMSE.

– Обчислення U і V : Один із способів знайти хороший вибір для U і V в УФ-розкладання починається з довільних матриць U і V. Повторно відрегулюйте один з елементів U або V, щоб мінімізувати середньоквадратичну ефективність між ультрафіолетовим випромінюванням продукту та заданою матрицею корисності. Процес сходиться до локального оптимуму, хоча, щоб мати хороші

шанси отримати загальний оптимум, ми мусимо або повторити процес із багатьох вихідних матриць, або здійснити пошук з початкової точки різними способами.

– Виклик Netflix: Важливим рушієм досліджень систем рекомендацій був виклик Netflix. Приз у розмірі 1 000 000 доларів був призначений учаснику, який міг створити алгоритм, який був на 10% кращим за власний алгоритм Netflix при прогнозуванні рейтингу фільмів користувачами. Премія була вручена у вересні 2009 року.

Значний поштовх для досліджень систем рекомендацій було дано, коли Netflix вручив приз у розмірі 1 000 000 доларів США першій людині чи команді, яка перемогла власний алгоритм рекомендацій, який називається CineMatch, на 10%. Після понад трьох років роботи премію було вручено у вересні 2009 року.

Виклик Netflix складався з опублікованого набору даних, який давав оцінки приблизно півмільйона користувачів (як правило, невеликі підгрупи) приблизно 17000 фільмів. Ці дані були відібрані з більшого набору даних, і запропоновані алгоритми були перевірені на їх здатність передбачати рейтинги в секретному залишку великого набору даних. Інформація для кожної пари (користувача, фільму) у опублікованому наборі даних включала рейтинг (1–5 зірок) та дату, коли було складено рейтинг.

RMSE використовувався для вимірювання продуктивності алгоритмів. CineMatch має RMSE приблизно 0,95; тобто, типовий рейтинг давав би майже одна повна зірка. Щоб виграти приз, потрібно було, щоб ваш алгоритм мав RMSE, який становив не більше 90% RMSE від CineMatch.

Бібліографічні примітки до цього розділу містять посилання на описи виграшних алгоритмів. Тут ми згадуємо деякі цікаві та, можливо, неінтуїтивні факти про виклик.

• CineMatch був не дуже хорошим алгоритмом. Насправді, рано було виявлено, що очевидний алгоритм прогнозування для оцінки користувачем u фільму m середнього:

1. Середня оцінка, дана u у всіх рейтингових фільмах та

2. Середнє значення оцінок фільму m від усіх користувачів, які оцінили цей фільм.

було лише на 3% гірше, ніж CineMatch.

- Троє студентів (Майкл Харріс, Джерей Ван та Девід Камм) виявили, що алгоритм УФ-розкладання, описаний у Розділі 9.4, дає 7% поліпшення порівняно з CineMatch у поєднанні з нормалізацією та кількома іншими трюками.
- Виграшний запис насправді був поєднанням декількох різних алгоритмів, які були розроблені самостійно. Друга команда, яка подала заявку, яка б виграла, якби вона була подана кількома хвилинами раніше, також була сумішшю незалежних алгоритмів. Ця стратегія - поєднання різних алгоритмів - раніше використовувалась у ряді важких проблем і є про що варто пам’ятати.
- Було зроблено кілька спроб використати дані, що містяться в IMDB, базі даних Інтернет-фільмів, для узгодження назв фільмів із виклику Netflix з їхніми іменами в IMDB, і таким чином витягти корисну інформацію, яка не міститься в самих даних Netflix. IMDB має інформацію про акторів та режисерів та класифікує фільми на один або більше з 28 жанрів. Було виявлено, що жанрова та інша інформація не була корисною. Однією з можливих причин є те, що алгоритми машинного навчання так чи інакше змогли виявити відповідну інформацію, а друга полягає в тому, що проблему розв’язання сутності, пов’язану з іменами фільмів, як зазначено в даних Netflix та IMDB, не так просто вирішити точно.
- Час оцінки виявився корисним. Здається, є фільми, які швидше за все оцінять люди, які оцінюють його відразу після перегляду, ніж ті, хто чекає деякий час, а потім оцінює. Як приклад такого фільму був наведений "Патч Адамс". І навпаки, є й інші фільми, які не сподобались тим, хто оцінив їх відразу, але їх краще оцінили через деякий час; Як приклад було наведено “Memento”. Незважаючи на те, що неможливо почерпнути інформацію про те, скільки часу була затримка між переглядом та рейтингом, загалом можна впевнено вважати, що більшість людей бачать фільм незабаром після його виходу. Таким чином, можна вивчити рейтинги будь-якого фільму, щоб побачити, чи мають його рейтинги вгору чи вниз з часом.

1.3. Системи агрегації даних

1.4. Класифікація програмних екосистем

Поняття екосистем відбувається з екології. Визначення з Вікіпедії визначає екосистему як природну одиницю, що складається з усіх рослин, тварин та мікроорганізмів (біологічних факторів) в районі, що функціонує разом з усіма неживими фізичними (неживими) факторами навколишнього середовища [8].

Хоча вищезазначене є чудовою рішучістю, менш придатною для суперечок у цій операції, і, отже, ми починаємо з концепції людських екосистем.

Екосистема людини складається з суб'єктів, взаємозв'язку між ними, дій цих суб'єктів та трансакцій, разом із цими комунікаціями щодо фізичних або нефізичних факторів. Для аргументування в цій операції ми виділимо подальші комерційні та соціальні екосистеми. У комерційній екосистемі дійовими особами є фірми, постачальники та клієнти, чинниками є товари та послуги, а операції включають фінансові операції, а також обмін інформацією та знаннями, запитами, передпродажними та післяпродажними операціями тощо. Соціальні екосистеми користувачів, їх соціальних відносин та обміну різними формами інформації. Екосистема програмного забезпечення складається з набору програмних рішень, які включають, підтримувати та автоматизувати дії та операції агентів у пов'язаній соціальній чи діловій екосистемі та організацій, які приймають ці рішення. Звичайно, програмна екосистема - це також екосистема, безумовно, комерційна екосистема, і, отже, товари та послуги - це програмні рішення та послуги програмного забезпечення, які включають, підтримують або автоматизують дії та транзакції.

Хоча програмні екосистеми приділяють важливу увагу в контексті Web 2.0, ця категорія екосистем була незмінною протягом багатьох десятиліть. Особливо наочно почався 1990 рік, коли різні компанії за допомогою хука або жулика боролися за домінування настільних програм для операційних систем, отримуючи підтримку найвпливовіших, таких як IBM і Microsoft [7].

Тздатний 1.1

Класифікація екосистеми програмного забезпечення

Категорія / Платфор	**Операційна система**	**Застосування**	**Програмування для кінцевих користувачів**
Робочий стіл	Windows, * nix, Apple OS X	OpenOffice	Mathematica, VHDL, MS Excel
Інтернет	Google AppEngine, Bungee Labs, розробник Yahoo	Amazon, eBay, Ning, SalesForce	Yahoo! Pipes, Microsoft PopFly, Google Mashup Editor
Мобільний	Android, Palm, iPhone, Nokia S60, Symbian	Жоден	Жоден

Програмні екосистеми існують у повторюваних областях, і Ян Бош пропонує класифікацію [7] екосистем програмного забезпечення, яка представлена в таблиці 1.1. Ця систематизація прагне організувати екосистеми у двовимірному просторі. Перше вимірювання краще, це характеризується як абстракційний рівень, на якому існує програмна екосистема, визначена на трьох рівнях, тобто операційна система, програма та програми кінцевого користувача. Друге вимірювання охоплює розвиток обчислювальної галузі з точки зору домінуючої обчислювальної платформи, тобто робочого столу, мережі та мобільних платформ. Хоча можна було б стверджувати, що перед настільними комп'ютерами стояли мейнфрейми та мінікомп'ютери, і що в епоху всюдисущих розрахунків можна визначити інші платформи, крім мобільних,

У наступному розділі моделі екосистем програмне забезпечення буде побудовано відповідно до класифікації, зазначеної вище. Створення моделей буде здійснено за допомогою Єдиної мови моделювання (UML), яка наближається для створення схем, що відображають артефакти реального світу.

Уніфікована мова моделювання (UML) - стандартизована мова загального призначення в галузі розробки програмного забезпечення. Стандарт створений OMG (Object Management Group).

UML включає ряд графічних рішень для створення візуальних моделей систем програмного забезпечення.

Він також включає методи моделювання даних (діаграми взаємозв'язків сутності), моделювання бізнесу (робочі потоки), цільове моделювання та моделювання компонентів. UML може бути використаний для моделювання всіх процесів, що витікають протягом життєвого циклу розробки програмного забезпечення, за допомогою різних технологій реалізації.

UML - це стандартна мова моделювання, здатна обробляти паралельні та розподілені системи і є фактичним галузевим стандартом, що розробляється під керівництвом Групи управління об'єктами (OMG) [7].

Середовищем для створення моделей було обрано Gliffy, яке є безкоштовним рішенням для побудови UML.

Орієнтована на операційну систему програмна система є першою, де програмні екосистеми були чітко визначені та контрольовані. Яскравими прикладами є Windows, Linux та Apple OS X. Хоча Windows на даний момент можна розглядати як операційну систему з лідируючими позиціями, на початку дев'яностих вона конкурувала за частку на ринку з ОС / 2 IBM і Mac OS . За останній час Linux та Apples OS X відповідно збільшили власні частки на ринку серверів та клієнтських машин.

У мережі деякі платформи прагнуть стати базовими операційними системами. З доступних серверних платформ це Google AppEngine, розробник Yahoo, Coghead та Bungee Lab. Всі вони працюють над системою використання браузера на стороні клієнта, потенційно з одним або кількома плагінами. Деякі платформи існують для мобільних пристроїв з певним успіхом, наприклад, Palm OS протягом 1990 року.

Однак за останні роки конкуренція також посилила платформи, які в даний момент змагаються за лідируючі позиції, це Series 60 від Symbian / Nokia, iPhone та Google Android [7].

Описати екосистему, засновану на операційних системах, можна на наступних особливостях:

- Такі системи не залежать від полів додатків і припускають, що непрямі розробники створюють додатки, які надають значення для клієнтів.
- У випадку з працівником настільних та мобільних варіантів, операційна система встановлюється для кожного пристрою, який намагається запустити програми, розроблені для екосистеми. Отже, успіх екосистеми, заснованої на операційних системах у великому рівні, залежить від продажів пристроїв, що підтримують цю ОС.
- Операційні системи, як правило, оптимізують для розробки незалежних додатків і пропонують підтримку перехресної інтеграції програм та компонентів.
- Постачальники операційної системи пропонують засоби розвитку для екосистем для спрощення роботи розробникам [12].

Можна вважати таким видом переваги екосистем:

- Успіх екосистем, заснованих на операційних системах, визначається програмами, побудованими на їх основі. Однак фактором успішності стимулювання є зменшення витрат, необхідних розробникам на програмування програм на базі цих ОС.

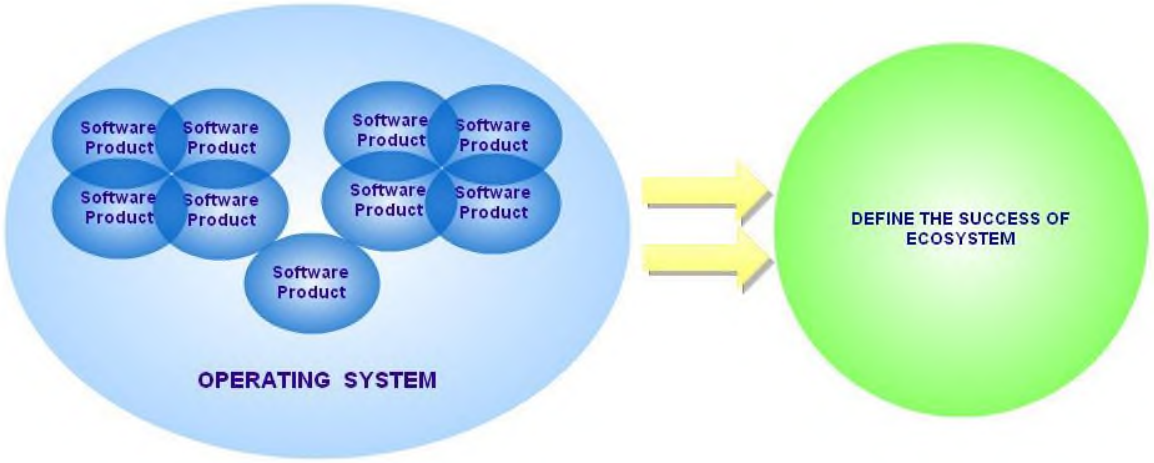


Рис. 1.1 Програмні рішення, як фактор успіху екосистеми

- Хоча операційна система забезпечує загальнодоступну функціональність, вона постійно повинна розширювати набір функцій, щоб підтримувати привабливість для розробників. Можливість операційної системи включити функціонал для адаптації є важливим фактором успіху або, інакше, занурення.
- Нарешті, мабуть, найважливішим фактором успіху є кількість клієнтів, які використовують ОС та забезпечують монетизацію для розробників.

В екосистемі прикладних програм такі властивості:

- Як вже було сказано вище, платформа, що базується на програмі, зазвичай запускається з успішного он-лайн додатку.
- постачальник платформи часто пропонує хостинг або інший підхід, щоб зробити взаємодію між платформою та непрямим додатком настільки жорсткою, наскільки це можливо для клієнта.
- Непрямі розробники додатків розширюють функціональність домену, що надається платформою. Ця відмінність від екосистем ОС, де розробники створюють власні програми.
- Щоб досягти щільної інтеграції та більших витрат для клієнта, інтеграція між платформою та розширеннями сприяє потоку даних, операціям та покращенню взаємодії з користувачем [7].

Фактори успіху екосистеми прикладних програм:

– Розширеним фактором успіху будь-якої екосистеми програмного забезпечення є велика група клієнтів або використання тими клієнтами, у яких є справжня причина розширити платформу за допомогою додаткових функціональних можливостей.
– По-перше, компанія-розробник платформи повинна прагнути спростити взаємодію з непрямими розробниками за допомогою дозволу використання загальних цілей, популярних середовищ розробки, стійких та орієнтовних інтерфейсів, а у випадку веб-додатків, легкої розробки та інтеграція з платформою.
– Хоча більшість компаній розробників платформ для додатків концентрується на забезпеченні доступу до даних, надання рішень для моделі даних та розширення робочого потоку так само, як досвід інтеграції в тій самій користувальницькій платформі важливий для досягнення тісної взаємодії з точки зору клієнт.

1.6. Висновки до розділу

У другому розділі представлено основні визначення рекомендаційних систем, які будуються на агрегаторах даних. Додатково розглянуто модифікацію візерункових структур та методи виведення програмного модуля в окрему програмну ескосистему.

РОЗДІЛ 2
МЕТОДИ ПОБУДОВИ СИСТЕМИ ВИЗНАЧЕННЯ КОРИСТУВАЦЬКИХ ВПОДОБАНЬ

2.1. Можливосты сучасних веб-технологій для побудови агрегаторів інформації

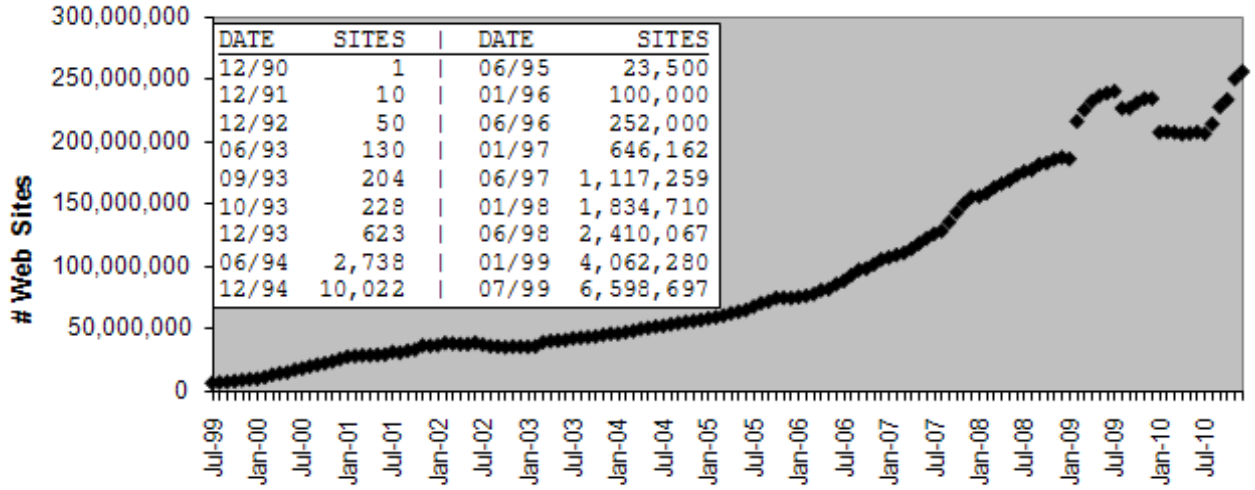
Среда зростання закону Мура породила революцію. Сучасні обчислювальні середовища набагато складніші та хаотичніші, ніж у перші дні вимірюваного підключення, дорогоцінних циклів процесора та обмеженої ємності. Дані та послуги мобільні та широко відтворюються для забезпечення доступності, продуктивності, довговічності та місцевості. Компоненти цієї інфраструктури, навіть перебуваючи в постійному русі, взаємодіють багатим і складним чином між собою, намагаючись досягти послідовності та корисності в умовах постійно мінливих обставин. Динамічний характер середовища багато в чому підкреслює традиційні підходи до надання послуг з імен об'єктів, узгодженості, розташування та маршрутизації.

У наш час Інтернет став неминучою частиною нашого життя. Щогодини дедалі більше людей відкривають для себе щось нове та цікаве в Інтернеті та діляться цим з іншими. У період з 2005 по 2010 рік кількість користувачів Інтернету подвоїлася [3], і, як очікувалося, у 2010 році вона перевищить два мільярди. Щороку переглядати та відкривати Інтернет стає простіше і легше. Але лише технічні фахівці бачать, наскільки складним є процес обслуговування веб-сайтів з року в рік.

В останні кілька років збільшення використання веб-додатків спричинило зростаючі потреби користувачів та попит на швидкий розвиток. Це наслідок багатьох факторів: встановлення нульового клієнта, розгортання лише на сервері, потужні засоби розробки, зростаюча кількість користувачів тощо.

Зі зростанням веб-трафіку навантаження на популярний веб-сервер швидко зростає. Щоб підтримувати популярний сайт, адміністраторам та розробникам потрібно задуматися про багато речей, і іноді їм доводиться мати справу з науковими проблемами, а не технічними, які можуть виникнути внаслідок технічних. Рисунок 1.1 демонструє зростання Інтернету за останнє десятиліття.

Можна помітити великий стрибок, починаючи з 2007 року, вплив на нього зробили послуги зв'язку та обміну, такі як Facebook [3]. Нова ера інтерактивних засобів масової інформації та комунікаційних сайтів розвивалася дуже швидко. Сама ідея мультимедійної інтерактивності в Інтернеті не була новою, але в той час вона стала поширюватися між людьми, що не пов'язані з комп'ютером, оскільки нова дуже важлива особливість сучасних сайтів почала займати своє місце - веб-дизайн. Тож маркетологи позначили ці «нові» сайти як web 2.0.



DATE	SITES	DATE	SITES
12/90	1	06/95	23,500
12/91	10	01/96	100,000
12/92	50	06/96	252,000
06/93	130	01/97	646,162
09/93	204	06/97	1,117,259
10/93	228	01/98	1,834,710
12/93	623	06/98	2,410,067
06/94	2,738	01/99	4,062,280
12/94	10,022	07/99	6,598,697

Рис. 2.1. Зростання WWW за 2000 - 2010 роки
Кількість веб-сайтів означає кількість веб-серверів (один хост може мати кілька сайтів за допомогою різних доменів або номерів портів) [3].

Термін Web 2.0 асоціюється з веб-додатками, що сприяють спільному обміну інформацією, сумісності, орієнтованому на користувача дизайну та співпраці у Всесвітній павутині. Сайт Web 2.0 дозволяє користувачам взаємодіяти та співпрацювати один з одним у діалозі соціальних медіа як творці створеного користувачами вмісту у віртуальному співтоваристві, на відміну від веб-сайтів, де користувачі (споживачі) обмежуються пасивним переглядом створеного вмісту для них. Прикладом Web 2.0 є сайти соціальних мереж, блоги,

вікі, сайти для обміну відео, розміщені послуги та веб-програми. Хоча цей термін пропонує нову версію Всесвітньої павутини, він не стосується оновлення будь-якої технічної специфікації, а сукупних змін у способах використання розробниками програмного забезпечення та кінцевими користувачами Інтернету.

Іншими словами, користувачі можуть надавати дані, що знаходяться на сайті, і отримувати певний контроль над цими даними. Ці веб-сайти можуть мати "архітектуру участі", яка заохочує користувачів додавати вартість додатку, коли вони використовують його та обговорюють.

Постановка проблеми

Більшість використовуваних існуючих моделей програмування та структур операційної системи не відповідають належним чином потребам складних, динамічних Інтернет-серверів, які повинні підтримувати надзвичайну одночасність (близько десятків тисяч клієнтських з'єднань) і відчувати стрибки навантаження, які на порядок перевищують Середня.

Асинхронна керована подіями архітектура з асинхронним неблокуючим введенням-виведенням представляє нову точку проектування для великомасштабних Інтернет-служб, які повинні підтримувати значну паралельність та несподівані зміни навантаження. Він об'єднує аспекти потокової передачі, керовану подіями паралельність, динамічне управління ресурсами та адаптивне управління перевантаженням у цілісну веб-платформу.

У цій моделі кожна стадія втілює надійний багаторазовий програмний компонент, який виконує підмножину обробки запитів. Виконуючи контроль над допуском для кожної черги подій, послуга може бути добре підготовлена до завантаження, запобігаючи надмірній передачі ресурсів, коли попит перевищує потужність послуги.

Архітектура, керована подіями, використовує динамічний контроль для автоматичної настройки параметрів виконання (таких як параметри планування кожного етапу), а також для управління навантаженням, наприклад, шляхом адаптивного скидання навантаження.

Дослідження - це діяльність, заснована на зборі інформації, що стосується швидкості, масштабованості та характеристик стабільності випробовуваного

товару для визначення та подальшого поліпшення якості продукції. Дослідження застосовується для доведення або спростування гіпотез щодо придатності різних архітектурних моделей шляхом спостереження за кількома проблемами продуктивності.

Ця робота повинна виявити, чи керується подіями архітектурний дизайн вищих чи таких самих відповідних показників, ніж традиційні сервісні проекти, одночасно демонструючи стійкість до величезних коливань навантаження. Дослідження починаються з оцінки існуючих підходів до проектування послуг, включаючи паралельність на основі потоків та подій.

Теорія черг може допомогти більш об'єктивно проаналізувати основні причини продуктивності програмного забезпечення та проблем масштабованості. Факторів продуктивності програмного забезпечення та масштабованості досить багато, і недоцільно випробовувати всі можливі перестановки або використовувати тактику спроб і помилок, поки не будуть виявлені справжні основні причини.

Сучасні обчислювальні середовища набагато складніші та хаотичніші, ніж у перші дні вимірюваного підключення, дорогоцінних циклів процесора та обмеженої ємності. Дані та послуги мобільні та широко відтворюються для забезпечення доступності, продуктивності та довговічності. Компоненти цієї інфраструктури, навіть перебуваючи в постійному русі, взаємодіють багатим і складним чином між собою, намагаючись досягти послідовності та корисності в умовах постійно мінливих обставин.

Зі зростанням важливості Інтернету як Інтернет-додатків виникає все більша потреба точно моделювати та відтворювати типові робочі навантаження в Інтернеті. Зокрема, здатність генерувати потік запитів HTTP, що імітує сукупність реальних користувачів, важлива для оцінки продуктивності та планування потужності серверів, проксі-серверів та мереж. Створення репрезентативних веб-посилань може бути складним, оскільки робочі навантаження в Інтернеті мають ряд незвичних функцій.

По-перше, емпіричні дослідження роботи веб-серверів показали, що вони відчувають надзвичайно різноманітні вимоги, що проявляється як мінливість навантажень процесора та кількості відкритих з'єднань [10].

Проблеми з великим навантаженням

У проекті із великим навантаженням можуть виникнути дві основні проблеми: масштабованість та надійність [2].

Масштабованість - це здатність системи обробляти зростаючий обсяг роботи здатним способом або її здатність розширюватися, щоб пристосувати цей ріст. Система називається масштабованою системою, якщо продуктивність покращується після модифікації (оновлення) [2, 6].

Проблему масштабованості слід завжди враховувати при розробці складних та високонавантажених проектів, оскільки вона пояснює, як розподілити навантаження на декілька процесорних блоків, що в багатьох випадках є єдиним способом підвищення продуктивності. Це властивість системи щодо управління зростанням навантаження та збереження пропускної здатності, незважаючи на обмежені ресурси.

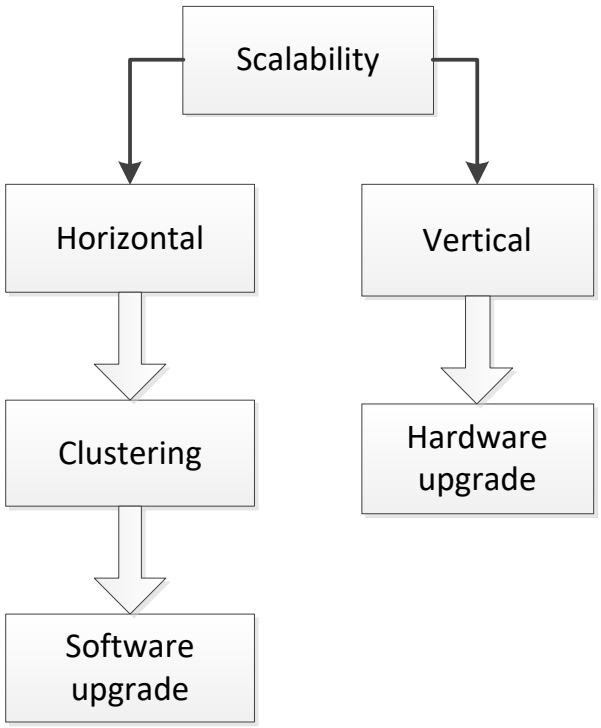


Рис. 0.1 Підходи до масштабованості

Існує два методи масштабування: вертикальний та горизонтальний (див. Рисунок 1.2). Вертикальна масштабованість покращує продуктивність за рахунок

оновлення фізичних ресурсів машини (ЦП, пам’ять, пам’ять тощо) у домені 1 фізичного вузла. Хоча горизонтальна масштабованість більше схожа на розподілені обчислення, що покращує продуктивність програми за допомогою клонування атомних блоків обробки системи. Отже, масштабованість - це здатність системи ефективно управляти розширенням ресурсів для підтримки більших навантажень.

Звичайно, вертикальний підхід є найпростішим, оскільки це просте очевидне рішення для оновлення апаратного забезпечення серверів без необхідності міняти програмне забезпечення, але все ж потік підключень користувачів настільки великий, що подальше оновлення апаратного забезпечення неможливе через до фізичних обмежень. Але це лише короткострокове рішення, яке швидко досягає свого максимуму, тому що світ переступив межу, коли закон Мура не працює.

З іншого боку, горизонтальний підхід створює додаткові проблеми, після реплікації серверів слід продумати спосіб, як оновити дані між блоками, щоб зберегти фактичну інформацію. Іншою серйозною проблемою горизонтальної масштабованості є збалансування навантаження - методологія розподілу робочого навантаження між кількома кластерами (мережеві зв'язки, центральні процесори, дисководи або інші ресурси) для досягнення оптимального використання ресурсів, максимізації пропускної здатності, мінімізації часу відгуку та уникнення перевантаження.

Для веб-сайтів балансування навантаження зазвичай є програмним забезпеченням, яке обробляє певний порт, де зовнішні клієнти підключаються до служб доступу. Балансир навантаження на основі запрограмованої логіки пересилає запити на один із серверних блоків серверного сервера, який зазвичай є блоком з найменшим навантаженням. Потім пристрій відповідає балансувачу навантаження, який, у свою чергу, відповідає клієнту, без того, щоб клієнт ніколи не знав про складне внутрішнє розділення функцій. Він також може мати переваги безпеки, що заважає клієнтам безпосередньо контактувати із серверними серверами, приховуючи структуру внутрішньої мережі та

запобігаючи атакам на мережевий стек ядра або не пов'язаних між собою служб, що працюють на інших портах.

Іноді балансир навантаження забезпечує механізм виконання деяких спеціальних функцій, коли всі серверні сервери недоступні або недоступні. Це може включати переадресацію на резервний сервер або балансир навантаження або відображення повідомлення про збій.

Більш складні балансири навантаження можуть враховувати додаткові фактори, такі як повідомлене завантаження сервера, останній час відгуку, сервери одиниць вгору / вниз, а також кількість активних з'єднань, географічне розташування та будь-який інший параметр моніторингу стану сервера, наприклад, скільки трафіку він нещодавно було призначено. Складні високопродуктивні системи можуть використовувати кілька шарів балансування навантаження.

Другу проблему, надійність або відмову, вважають вирішеною, коли відмова будь-якого сервера не спричиняє фатального впливу на систему в цілому [2]. Отже, відмовостійкість - це автоматичний перехід на резервний комп’ютерний сервер, систему чи мережу в резервному режимі або в режимі очікування при відмові або ненормальному завершенні роботи раніше активного сервера, системи чи мережі. Дизайнери або архітектори систем сайтів із великим навантаженням повинні це враховувати та забезпечувати можливість відмов серверів; оскільки такі проекти вимагають постійної доступності та високого ступеня надійності. Властивість також називається відмовостійкістю, що пояснюється наступним чином: систему можна назвати відмовостійкою, якщо вона продовжує працювати належним чином у разі виходу з ладу деяких її компонентів.

Відмовостійкість - це не просто властивість окремих серверів або блоків, вона характеризується загальною системою та правилами, за якими вони взаємодіють. В межах окремого блоку відмовостійкості можна досягти, передбачивши виняткові умови та побудувавши систему для боротьби з ними, і, загалом, прагнучи до самостабілізації, щоб система сходилася до стану без помилок. Системи, стійкі до несправностей, характеризуються як з точки зору

планових відключень послуг, так і незапланованих відмов у роботі. Вартість вимірювання називається доступністю і виражається як відсоток робочих годин до загальної кількості годин. Висока доступність дуже важлива для хостингових програмних компаній, оскільки вона демонструє їхній професіоналізм та стабільність роботи на їх серверах при мінімальному простої.

Надійність може бути досягнута різними апаратними та програмними шляхами, наприклад, реплікація ведучий-підлеглий або shardig. Реплікація означає паралельне використання однакових екземплярів одних і тих самих систем, тоді як надмірність використовує однакові ідентичні екземпляри з однаковими системами, але вони не працюють паралельно і перемикаються на інший у разі відмови. Використання декількох компонентів із балансуванням навантаження замість одного компонента може підвищити надійність завдяки надмірності. Система повинна мати можливість ізолювати неробочий блок з робочого процесу, що вимагає додаткового механізму ізоляції несправностей. Це також запобігає стримуванню несправностей, які можуть бути спричинені вірусами або атаками переповнення буфера, що може призвести до того, що один блок не може поширити несправність на інші блоки.

Найбільш поширеним розміром кластера є кластер із двома вузлами, оскільки це мінімум, необхідний для забезпечення надмірності, але багато кластерів складаються з набагато більше, іноді з десятків вузлів. Такі конфігурації можна класифікувати на одну з наступних моделей [2]:

Активний / активний - невдалий вузол виключається із циркуляції робочого циклу, а його трафік розподіляється між існуючими вузлами або збалансованим навантаженням за іншими вузлами. Зазвичай це можливо лише тоді, коли вузли використовують однорідну конфігурацію програмного забезпечення.

Активний / пасивний - забезпечує повністю надлишковий екземпляр кожного вузла, який працює лише тоді, коли пов'язаний з ним основний вузол виходить з ладу. Ця конфігурація, як правило, вимагає найбільшого додаткового обладнання.

N + 1 - забезпечує єдиний додатковий вузол, який підключається до мережі, щоб взяти на себе роль вузла, який не вдався. Зазвичай це стосується

кластерів, які мають кілька служб, що працюють одночасно; в одному випадку обслуговування це перетворюється на активне / пасивне.

N + M - у випадках, коли один кластер управляє багатьма службами, наявність лише одного виділеного вузла відмови може не забезпечити достатню надмірність. У таких випадках включено та доступно більше одного (M) резервних серверів. Кількість резервних серверів - це компроміс між вимогами щодо вартості та надійності.

Додатковими рідко використовуваними моделями є комбінації з базових, таких як: N-to-1 (резервний вузол переходу до режиму відмови стає тимчасово активним, поки оригінальний вузол не може бути відновлений або повернути в Інтернет) та N-to-N - Комбінація активних / Active та N + M кластери, N до N кластери перерозподіляють послуги, екземпляри або з'єднання з невдалого вузла серед решти активних вузлів, таким чином усуваючи (як і при активному / активному) необхідність у `` резервному " вузлі, але вводячи потреба в додатковій ємності на всіх активних вузлах.

Загалом, кластери, як правило, використовують усі доступні методи, щоб зробити окремі модулі та інфраструктуру спільних систем максимально надійними.

Супутні роботи

У попередніх роботах [7, 9] була виявлена проблема прикладної теорії управління зворотним зв'язком до якості кешування проксі-кешу (QoS), а також проблема автоматизації налаштування контролера. На мережевому рівні теорія управління була застосована до управління потоком пакетів в Інтернет-маршрутизаторах [7, 9]. Завдяки корисності теоретико-контрольного підходу та його універсальних застосувань з’явились проміжні середовища для гарантій QoS на основі управління. Автори [3, 4, 9] надали інструменти, що допомагають застосовувати теоретично-теоретичні методи проектування до більшого класу систем. Робота продемонструвала, що додавання можливостей теоретичного передбачення черг для управління циклами може підвищити продуктивність циклу зворотного зв'язку нетривіальним способом. Цей прийом матиме велике значення, коли контролюватимуться показники QoS, пов’язані з часом.

У роботі [3] автори використовували підходи управління зворотним зв'язком для досягнення захисту від перевантаження та гарантій продуктивності веб-серверів. Стратегія була заснована на теорії планування в режимі реального часу, яка стверджує, що час відгуку може бути гарантований, якщо використання сервера підтримується нижче попередньо обчислених меж. Таким чином, теоретично-контрольні підходи, в поєднанні зі стратегіями адаптації вмісту, були сформульовані для збереження використання сервера на рівні або нижче межі.

Теорія черг забезпечує передбачуваний каркас, необхідний таким чином, щоб очікувані затримки можна було зробити безпосередньо з вхідного навантаження. У попередній роботі [9] автори описали першу спробу поєднати передбачення черг та контроль зворотного зв'язку в програмних послугах в контексті надання абсолютних гарантій затримки веб-серверів. Відносні гарантії особливо корисні для диференціації послуг на перевантажених системах. Вони набули великої популярності з моменту формулювання пропорційно диференційованої системи послуг [4, 9]. На відміну від інших моделей диференційованих послуг, пропорційна диференційована послуга забезпечує як передбачувану, так і контрольовану відносну диференціацію. Це передбачувано, оскільки диференціація є послідовною (тобто вищі класи кращі або, принаймні, не гірші), незалежно від варіацій навантаження класу. Це можна контролювати, це означає, що оператори мережі можуть регулювати інтервал якості між класами на основі обраних критеріїв. Вони емпірично порівнюють три різні варіанти загальної постановки проблем відносної затримки, поєднання теоретичного передбачення черг та контролю зворотного зв'язку.

Передбачувач черги розміщується на шляху прямої подачі, щоб зробити оцінки розподілу ресурсів по класу, що задовольняє вимогам затримки. Кілька циклів управління зворотним зв'язком коригують це розподіл для відповідних класів у відповідь на вимірювання відхилень продуктивності для кожного класу, що виникають внаслідок неточності предиктора. Варіанти цієї архітектури, розглянуті в цій роботі, відрізняються співвідношенням між кількістю циклів зворотного зв'язку та кількістю класів клієнта, визначенням заданих точок для

циклу, визначенням помилки продуктивності та способом виведення контролера зворотного зв'язку математично поєднаний з тим, що передбачає чергування.

Стратегії планування запитів на основі пріоритетних напрямків були досліджені для диференціації часу відгуку на веб-серверах [3, 4, 7, 8, 9].

У роботі [8] автори розглядали суворі стратегії планування пріоритетів для контролю використання центрального процесора на веб-серверах. Вхідні запити були класифіковані у відповідні черги з різним рівнем пріоритету для відповідних послуг. Запити нижчих пріоритетних класів виконувались лише у тому випадку, якщо жодного запиту не існувало в жодному з вищих пріоритетних класів. Результати показали, що диференціація часу відгуку може бути досягнута в тому сенсі, що вищі класи отримують менше часу відгуку, ніж нижчі класи. Однак якісний інтервал між різними класами не може бути гарантований строгим плануванням пріоритетів. Отже, такого роду стратегії планування на основі пріоритетів не можуть досягти пропорційного розмежування часу відгуку на веб-серверах.

Опис, як ці варіанти впроваджені всередині веб-сервера Apache, аналізуючи їх переваги та недоліки, і повідомляє про досвід налаштування їх параметрів на практиці таким чином, щоб досягти найкращої продуктивності.

Теорія управління зворотним зв'язком нещодавно набула великої популярності як аналітична основа для надання м'яких гарантій якості обслуговування в обчислювальних системах, таких як Інтернет-сервери [3, 8]. Перевага управління зворотним зв'язком полягає в тому, що він змушує продуктивність системи виявляти самокорегуючу, самостабілізуючу поведінку. Інтерпретуючи специфікації QoS як задані точки контуру управління, система сходить до рівноваги навколо робочої точки, визначеної специфікацією, тим самим виробляючи бажаний QoS.

Завдяки надійності загальних контролерів зворотного зв’язку, збіжність системи спостерігається навіть за наявності неточностей моделювання, властивих нелінійності системи та варіацій параметрів системи в часі. Ці бажані властивості викликали великий інтерес до теорії управління як засобу досягнення програмних гарантій якості обслуговування.

Недоліком контролю зворотного зв'язку є те, що він, по суті, є реактивним підходом. Цикл зворотного зв'язку працює, реагуючи на виміряні відхилення від бажаної продуктивності, тобто здійснюючи коригувальні дії, які намагаються зменшити відхилення до нуля. Однак у багатьох випадках можливо передбачити, що продуктивність системи скоро погіршиться до того, як погіршення відбудеться насправді. Наприклад, у системі управління затримкою веб-сервера, де час відгуку сервера є контрольованим параметром, збільшення поточного навантаження на вхід може спричинити збільшення середнього часу відгуку найближчим часом. На жаль, контролер зворотного зв'язку, який вимірює поточну затримку, залишатиметься поза увагою щодо майбутнього перевантаження, доки воно не відбудеться.

Ця затримка у відповіді особливо значна, оскільки час відповіді сервера є ковзним середнім, що повільно реагує на зміни. Посилення управління зворотним зв'язком за допомогою передбачуваної системи дозволило б уникнути цієї проблеми, тим самим запобігаючи виникненню перевантаження.

Висновок

Сьогодні кількість веб-користувачів зростає дуже швидко, хоча навантаження на популярні інформаційні та медіа-ресурси, а також на соціальні мережі різко зросла. На пікових навантаженнях виникає проблема перевантаження, таким чином час відгуку неминуче збільшується, хоча може статися тимчасова нестабільність роботи або навіть аварія.

Є дві основні проблеми, які виникають під час вирішення проблеми із високим навантаженням: масштабованість та надійність. Масштабованість - це здатність системи забезпечити оновлення апаратного забезпечення (вертикальний підхід) або розподіл на декілька обробних блоків (горизонтальний підхід) із покращенням продуктивності. Горизонтальна масштабованість має проблему балансування навантаження між блоками і в більшості випадків призводить до оновлення програмного забезпечення. Вертикальний підхід простіший, але дуже швидко досягає свого максимуму.

Зростає попит на забезпечення пропорційної диференціації реагування для різних клієнтів на масштабованих веб-серверах, щоб задовольнити мінливу доступність ресурсів та задовольнити різні потреби клієнта.

У попередніх роботах проблему перевантаження частково вирішували за допомогою якості обслуговування, теорій обслуговування та зворотного зв'язку. За допомогою методів аналізу та прогнозування черги або підходів управління зворотним зв'язком специфікація заданих значень контуру управління використовувалася для досягнення захисту від перевантаження та гарантій продуктивності, так що система сходиться до рівноваги навколо робочої точки, визначеної специфікацією, тим самим створюючи бажана якість обслуговування.

2.1. Розробка архітектури клієнт-серверної системи агрегації даних з соціальних мереж

Терміни Інтернет та Інтернет часто використовуються у повсякденній промові без особливих відмінностей; проте вони не однакові. Інтернет - це глобальна система взаємопов’язаних комп’ютерних мереж. На відміну від них, Інтернет - одна із служб, що працюють в Інтернеті. Це колекція текстових документів та інших ресурсів, пов’язаних гіперпосиланнями та URL-адресами, і переданих веб-серверами веб-браузерам. Перегляд веб-сторінки у Всесвітній павутині зазвичай починається або з введення URL-адреси сторінки у веб-браузер, або з гіперпосилання на цю сторінку або ресурс.

Потім веб-браузер ініціює серію комунікаційних повідомлень за лаштунками, щоб отримати та показати їх.

Веб-агенти (включаючи браузери) спілкуються за допомогою стандартизованих протоколів, що забезпечують взаємодію шляхом обміну повідомленнями за допомогою певного синтаксису та семантики. Вводячи URL-адресу в діалогове вікно пошуку або вибираючи гіпертекстове посилання, користувач повідомляє браузеру виконати дію пошуку для ресурсу, визначеного URI (уніфікований ідентифікатор ресурсу).

Спочатку браузер перетворює частину URL-адреси з іменем сервера (наприклад, "weather.com") на адресу Інтернет-протоколу (IP), використовуючи базу даних, відому як Система доменних імен (DNS), і як результат пошуку повернуто IP-адресу.

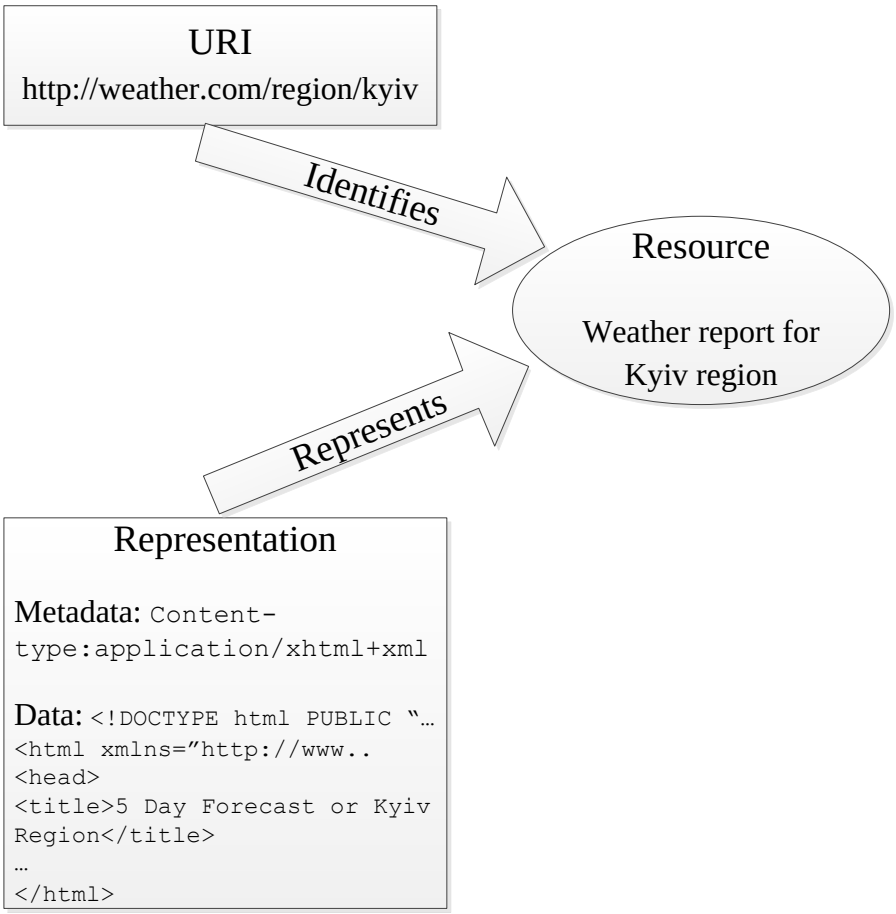


Рис. 0.2 Зв'язок між ідентифікатором, ресурсом та поданням

Потім браузер надсилає запит GET HTTP (Hyper Text Transfer Protocol) (частина протоколу HTTP) на сервер за повернутою IP-адресою через порт TCP / IP 80 (стандартний порт, використовується, коли URL-адреса не вказує точно), а сервер відправляє назад повідомлення, що містить те, що, як він визначає, є поданням ресурсу на момент створення цього представлення. На рисунку 2.1 подання (відповідь сервера) інформації - це гіпертекстові дані, але можливі й інші види взаємодії.

Основною функцією веб-сервера (може бути програмне чи апаратне забезпечення) є надання відповіді (веб-сторінки) в результаті запитів клієнтів. Зокрема - доставка документів HTML та будь-якого додаткового вмісту, який

може бути включений у документ, наприклад, зображень, таблиць стилів та сценаріїв.

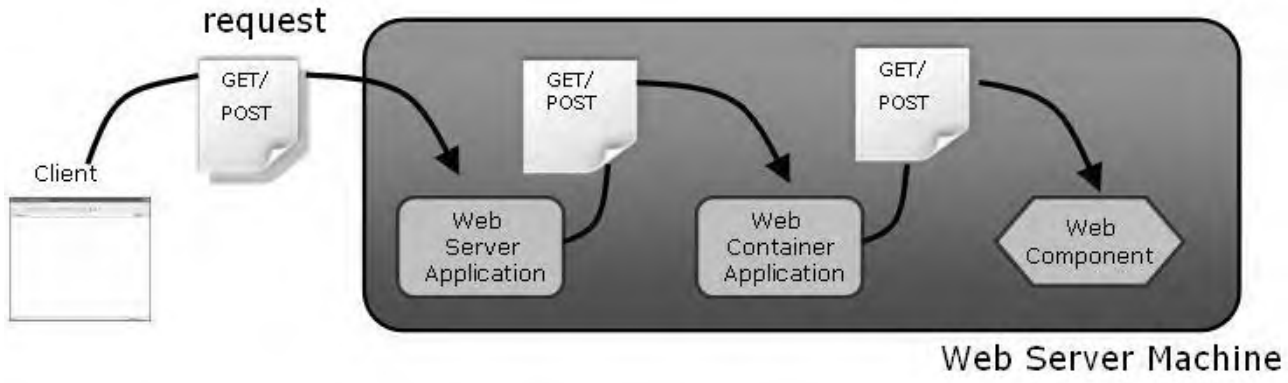


Рис. 0.3 Ілюстрація запиту клієнта до веб-сервера

Діаграма показує загальновживаний сценарій, коли клієнт через веб-браузер або веб-сканер ініціює зв'язок з веб-сервером, подаючи запит на певний ресурс за допомогою HTTP, і сервер відповідає (див. Рис. 2.4) із вмістом цього ресурсу або повідомлення про помилку, якщо це неможливо зробити. На ілюстрації можна помітити, що запит клієнта породив кілька підзапитів до внутрішніх компонентів. Більшість загальних веб-серверів складаються з 2 частин - обробника запитів та програми, яка генерує відповідну відповідь.

Обробник запитів - це програма для ОС, функції якої можуть відрізнятися, але загальні включають наступне:

приймати HTTP-з'єднання на вказаному порту;

перекласти шлях (шлях із URL-адреси) у локальне представлення ОС, яке може бути локальним файлом або додатком, що генерує дані відповідей;

балансове навантаження (обмежити кількість клієнтів та одночасні з'єднання клієнтів тощо);

обробляти відповідь HTTP.

Дані динамічної відповіді генеруються відповідно до шляху та параметрів, що передаються серверу, і можливо завдяки підтримці сценаріїв на стороні сервера (наприклад, Active Server Pages (ASP), PHP тощо). Це означає, що поведінка веб-сервера може бути написана окремими файлами (див. Рис. 2.3), тоді як власне програмне забезпечення сервера залишається незмінним. Зазвичай ця функція використовується для створення HTML-документів "на льоту", на

відміну від повернення виправлених документів. Це називається динамічним та статичним вмістом відповідно. Перший використовується в основному для отримання та / або модифікації інформації з баз даних. Однак останнє, як правило, набагато швидше і легше кешується.

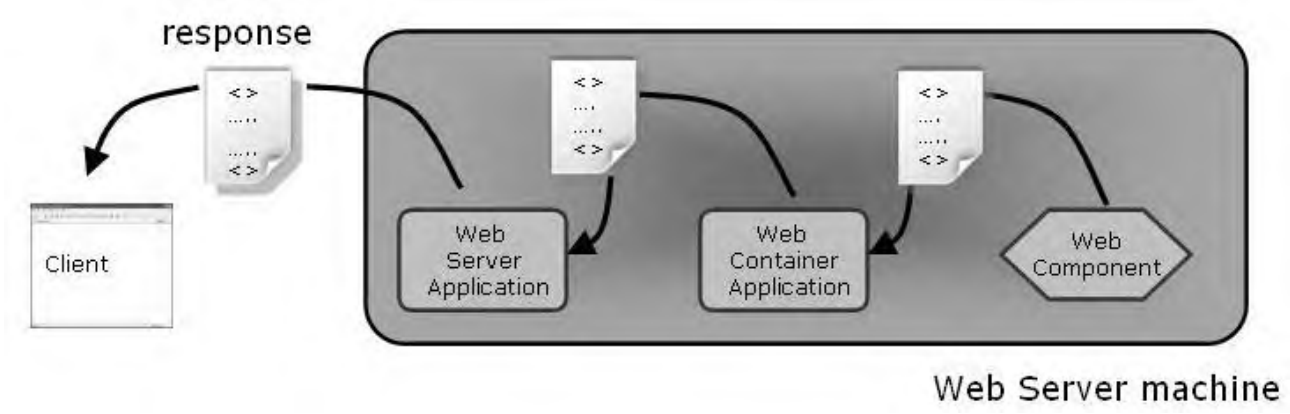


Рис. 0.4 Ілюстрація відповіді веб-сервера на клієнта

Більшість веб-серверів працюють за циклом запитів / відповідей HTTP. Це означає, що сервер приймає запит і надсилає відповідь у формі HTTP.

Хоча основною функцією є обслуговування вмісту, повна реалізація HTTP також включає способи отримання вмісту від клієнтів. Ця функція використовується для надсилання веб-форм, включаючи завантаження файлів.

2.2. Теорія черг

Теорія черг - це добре розроблений предмет, що має принципове значення для інформатики, забезпечуючи математичну основу та основні аналітичні інструменти для оцінки продуктивності комп'ютерних систем. Теорія черг забезпечує всі інструменти, необхідні для аналізу [8, 12].

Теорія черг - це математичне вивчення ліній очікування (черг). Це дозволяє проводити математичний аналіз кількох пов'язаних процесів, включаючи надходження в (задню частину) черги, очікування в черзі та обслуговування напередодні черги [11, 12].

Затримку повідомлень можна класифікувати на такі категорії [11]:

Затримка обробки - затримка між часами отримання пакета для передачі до точки введення його в чергу передачі. На кінці прийому це затримка між часом

прийому пакету в черзі прийому до моменту фактичної обробки повідомлення. Ця затримка залежить від швидкості обробки та завантаження системи.

Затримка черги - затримка між точкою введення пакета в черзі передачі і фактичною точкою передачі повідомлення. Ця затримка залежить від навантаження на лінію зв'язку.

Затримка передачі - різниця в часі між передачею першого біта пакету в передачу останнього біта. Ця затримка залежить від швидкості лінії зв'язку.

Теорія дозволяє виводити та обчислювати декілька показників продуктивності, включаючи середній час очікування в черзі або системі, очікуване число очікування або отримання послуги та ймовірність зустріти систему в певних станах, таких як порожня, повна, що має доступний сервер або потрібно чекати певного часу, щоб його обслужили.

2.2.1. Теорема Літтла

У теорії черг теорема Літтла стверджує [11, 12]:

Середньорічна кількість клієнтів у стабільній системі (L) дорівнює довгостроковій середній ефективній швидкості прибуття λ, помноженій на середній час очікування в системі T:

$$L = \lambda T$$

Середнє значення T надає однакову вагу часу очікування всіх споживачів, а не середнє значення часу очікування споживачів, які зараз перебувають у системі.

Це означає, що поведінка повністю не залежить від будь-якого розподілу ймовірностей, і, отже, не вимагає припущень щодо графіка, згідно з яким клієнти прибувають або обслуговуються.

У деяких випадках можна математично пов’язати не лише середнє число в системі із середнім очікуванням, але і пов’язати весь розподіл ймовірності (та моменти) числа в системі із очікуванням.

Цей закон застосовується до будь-якої системи, і зокрема, це стосується систем всередині систем. Єдині вимоги полягають у тому, щоб система була

стабільною та не мала переваги; це виключає перехідні стани, такі як початковий запуск або зупинка.

За допомогою теореми Літтла можна пояснити характеристики системи масового обслуговування та їх вплив на продуктивність системи.

2.2.2. Класифікація

Найважливіші характеристики системи масового обслуговування:

Процес прибуття - пояснюється розподілом щільності ймовірності або розподілом ймовірності прибуття, який визначає прибуття клієнта в систему;

Процес обслуговування (час обробки) - пояснюється розподілом щільності ймовірності, що визначає час обслуговування споживачів у системі. У системі обміну повідомленнями це стосується розподілу часу передачі повідомлення. Оскільки передача повідомлення прямо пропорційна довжині повідомлення, цей параметр опосередковано відноситься до розподілу довжини повідомлення;

Кількість одиниць обробки - кількість обробників, доступних для обслуговування споживачів. (У системі обміну повідомленнями це може стосуватися кількості посилань між вихідним та кінцевим вузлами).

Виходячи з вищезазначених характеристик, системи масового обслуговування можуть бути класифіковані за наступними умовами:

A / S / n

Де A - процес прибуття, S - процес обслуговування, а n - кількість серверів. A і S можуть бути будь-яким із наступного:

M (Марков)	Експоненціальна щільність ймовірності
D (детермінований)	Усі клієнти мають однакову вартість
G (загальне)	Будь-який довільний розподіл ймовірностей

Процес Маркова характеризується своєю безпам'ятністю: майбутні стани процесу не залежать від його минулої історії і залежать виключно від теперішнього стану.

Загальний процес не характеризується жодним розподілом ймовірностей, оскільки він абсолютно довільний.

Детермінований процес передбачуваний і характеризується різними константами, наприклад, коли час міжприбуття постійний від одного прибуття до іншого.

Приклади систем обслуговування з такою умовою та відповідно до характеристик можна визначити як:

M / M / 1: Це найпростіша система черги для аналізу. Тут час прибуття та обслуговування є негативними експоненціально розподіленими (процес Пуассона). Система складається лише з одного сервера. Ця система масового обслуговування може застосовуватися до найрізноманітніших проблем, оскільки будь-яка система з дуже великою кількістю незалежних клієнтів може бути апроксимована як процес Пуассона. Однак використання процесу Пуассона для часу обслуговування не застосовується у багатьох додатках і є лише грубим наближенням.

M / D / n: Тут процес прибуття є пуассоновим, а розподіл часу обслуговування детермінованим. Система має n серверів; час обслуговування передбачається однаковим для всіх клієнтів.

G / G / n: Це найзагальніша система обслуговування, де процеси прибуття та часу обслуговування є довільними. Система має n серверів. Для цієї системи масового обслуговування не відомо жодне аналітичне рішення.

2.2.3. Служба обробників

Робочий процес відповідає за надання окремих послуг, тобто за отримання запитів, їх обробку, генерування відповідей та надсилання їх запитуваному клієнту.

Працівник автоматично виконує балансування навантаження, динамічно регулюючи порогове значення, визначене як

$$T_{new} = f(T_{old}, Q_{(x)}) - (T_{old} - Q_{(x)}) \quad ; \; if \quad (Q_{(x)} < T_{old})$$
$$f(T_{old}, Q_{(x)}) - floor(T_{old}/10); \; if \quad (Q_{(x)} = T_{old}) \qquad (1)$$

where,

$f(T_{old}, Q_{(x)}) = | T_{old} - 2Q_{(x)} |$

$Q_{(X)}$ *is the number of queries processed in the previous X seconds.*

Таким чином службовий персонал динамічно пристосовується до навантаження мережі та обслуговування.

Щоразу, коли робоча служба отримує запит, вона витягує ідентифікатор труби клієнта-рівноправного користувача та запит. Запит надсилається до фактичної служби, яка розміщена на робочому рівні. Отримана відповідь надсилається рівному клієнту через його Pipe.

Служба працівника надсилає свій поріг та відмітку часу службі моніторингу, що працює на рівні її RV. Щоразу, коли запит обробляється, повідомлення “оброблений запит” надсилається службі точки входу для оновлення таблиці розкладу.

Якщо зв’язок із побаченням втрачено, тоді працівник чекає деякий час, якщо жодного побачення не знайдено, тоді він сам робить це побачення та рекламує свій новий статус. Під час цього очікування всі повідомлення про підтвердження запиту та відповіді кешуються, це зберігає стан партнера та, в свою чергу, всієї групи.

Навантаження, або те, що технічно називається пропонованим навантаженням, залежить від процесу прибуття та часу обслуговування, необхідного для кожного запиту.

Загальноприйнятим підходом для досягнення якості та ефективності послуг є правило прямого штатного розкладу. Припустимо для простоти постійну швидкість прибуття λ дзвінків на хвилину, і нехай середній час обслуговування буде E [S] (у секундах). Очікуване навантаження R визначається як

$R = \lambda \times E[S]$

Це середній обсяг роботи (за час обслуговування), який надходить до системи (за одиницю часу). З прогнозом R, один встановлює кількість агентів бути

$N = R + \beta\sqrt{R}$

для заздалегідь визначеного параметра β (зазвичай $-1 \leq \beta \leq 2$), який відображає необхідний баланс між якістю обслуговування (короткий час очікування, мало відмов) і ефективністю обслуговування (використання агентів). Вибір значення для β - це спосіб архітектора досягти необхідної продуктивності системи.

Гауссові лінійні змішані формулювання моделей, що використовуються для опису як відповідно трансформованої версії процесу прибуття, так і середнього часу обслуговування. Змішані моделі дозволяють досягти необхідної гнучкості для опису різних ефектів, використовуючи кореляційні структури в обох моделях. Більше того, вони забезпечують більш реалістичні інтервали прогнозування, ніж ті, що отримані, ігноруючи кореляції в серії. Змішані моделі також дозволяють нам включати екзогенні змінні «природним» способом.

Фактори тестування

Тест веб-серверів - це процес оцінки продуктивності веб-сервера, щоб з’ясувати, чи може сервер обслуговувати досить велике навантаження. Він призначений для визначення швидкості реагування, пропускної здатності та надійності системи за певного навантаження.

Тестування продуктивності - це широка та складна діяльність, яка може приймати різні форми, усунути багато ризиків та забезпечити широкий спектр застосування.

Зазвичай тести ефективності описуються як такі, що належать до однієї з наступних трьох категорій:

Тестування продуктивності. Цей тип тестування визначає або підтверджує характеристики швидкості, масштабованості та / або стабільності системи чи програми, що тестується. Ефективність стосується досягнення часу відгуку, пропускної здатності та рівня використання ресурсів, які відповідають цілям ефективності проекту або продукту.

Тестування навантаження. Ця підкатегорія тестування продуктивності орієнтована на визначення або перевірку робочих характеристик системи або додатка, що випробовується, при дії навантажень та обсягів навантаження, передбачених під час виробничих операцій.

Стрес-тестування. Ця підкатегорія тестування продуктивності орієнтована на визначення або перевірку характеристик продуктивності системи чи програми, що тестується, за умови, що вона перебуває в умовах, що перевищують звичайні схеми використання. Стрес-тести можуть також включати тести, орієнтовані на визначення або перевірку характеристик продуктивності системи або додатка, що тестується, під впливом інших стресових умов, таких як обмежена пам'ять, недостатня кількість місця на диску або збій сервера. Ці тести призначені для того, щоб визначити, за яких умов програма вийде з ладу, як вона вийде з ладу та які показники можна контролювати, щоб попередити про майбутній збій.

Коли навантаження, покладене на систему, зазвичай навантаження настільки велике, що в результаті очікуються умови помилок, хоча немає чіткої межі, коли діяльність перестає бути випробуванням навантаження і перетворюється на стрес-тест.

В ідеалі відсоток помилок повинен бути нульовим протягом пробного запуску.

Щоб бути найбільш ефективним, автоматизоване тестування програмного забезпечення слід розглядати як проект розробки програмного забезпечення.

Додаток, який вважається ефективним, - це додаток, який дозволяє кінцевому користувачеві виконувати задане завдання без зайвої сприйнятливої затримки чи подразнення.

Сприйняття кінцевим користувачем - це остання лінія, але для точного вимірювання ефективності існує ряд ключових показників, які необхідно враховувати. Ці показники можна розділити на два типи: орієнтовані на послуги та на ефективність.

Сервісно-орієнтованими показниками є доступність та час відгуку; вони вимірюють, наскільки програма надає послугу кінцевим користувачам.

Орієнтовані на ефективність показники - це пропускна здатність та використання; вони вимірюють, наскільки добре (чи ні) програма використовує ландшафт програми. Ми можемо коротко визначити ці терміни таким чином:

Доступність: кількість часу, коли програма є доступною для кінцевого користувача. Відсутність доступності є значною, оскільки багато додатків матимуть значні витрати на бізнес навіть за невеликого відключення. З точки зору тестування продуктивності, це означало б повну нездатність кінцевого користувача ефективно використовувати додаток.

Час відповіді: кількість часу, протягом якого програма відповідає на запит користувача. Для тестування продуктивності зазвичай вимірюється час реакції системи, тобто час між запитом користувача про відповідь програми та повною відповіддю, що надходить на робочу станцію користувача.

Швидкість пропускання показує кількість активних паралельних з'єднань, що обслуговуються щосекунди; це міра того, скільки віртуальних користувачів активно в даний момент часу. Отже, це реальна продуктивність обробки веб-платформи на даному обладнанні.

Швидкість пропускної здатності можна обчислити за такою формулою:

$$\text{Throughput rate} = \frac{\text{Number of samples}}{\text{Test duration}}$$

Це швидкість, з якою відбуваються орієнтовані на програми події. Хорошим прикладом може бути кількість звернень на веб-сторінку за певний проміжок часу.

Досягнення певної кількості одночасних віртуальних користувачів є рівновагою між часом, необхідним для завершення ітерації транзакції, та затримкою між ітераціями транзакцій. Його також називають пропускною здатністю транзакцій.

Досягнувши максимальної паралельності сервера (пропускної здатності), додаток повинен дозволити кінцевим користувачам своєчасно виконувати свої завдання.

$$\text{Response time} = \text{Network latency} + \text{Connection wait time}$$

+ Time of content generation

Використання показує відсоток теоретичної потужності ресурсу, який використовується. Прикладами є кількість пропускної здатності мережі, яку витрачає трафік додатків, і обсяг пам'яті, що використовується на сервері, коли активна тисяча відвідувачів.

Ефективність зазвичай визначається як пропускна здатність, поділена на використання. Якщо порівнювати два компоненти, якщо один має більшу пропускну здатність на тому ж рівні використання, це вважається більш ефективним. Якщо обидва мають однакову пропускну здатність, але один має нижчий рівень використання, це вважається більш ефективним.

$$\text{Efficiency} = \frac{\text{Throughput rate}}{\text{Utilization}}$$

Під великим навантаженням використання можна вважати однаковим, оскільки використання процесора є максимальним (100%) у кожному випадку. Тому ефективність прямо пропорційна пропускній здатності.

$$\text{Utilization is } const, \quad \Rightarrow \quad \text{Efficiency} \sim \text{Throughput rate}$$

Веб-сервери, як правило, стають нелінійними при завантаженні, що перевищує їх пропускну здатність, швидко погіршуючи продуктивність.

Середній час очікування E (W) для системи M / M / 1 - це середній час, протягом якого клієнти чекають обслуговування системою. Це дорівнює середньому системному часу, віднімаючи середній час обслуговування, який дорівнює 1 / μ. З цього випливає, що:

$$E(W) = E(S) - \frac{1}{\mu} = \frac{(1/\mu)\rho}{1-\rho}$$

де, E (S) - середній системний час, p - пропускна здатність

Середній час відгуку - це середнє арифметичне набору значень.

Медіана - це середнє значення в наборі чисел. Це корисно в ситуаціях, коли розраховане середнє арифметичне перекошене через невелику кількість викидів, що призводить до значення, яке не є справжнім відображенням середнього.

Медіана- число, яке ділить зразки на дві рівні половини. Половина зразків менше медіани, а половина більша. Деякі вибірки можуть дорівнювати медіані. Медіана така ж, як і 50-й процентиль.

Стандартне відхиленняє мірою мінливості набору даних. Стандартне відхилення - це показник того, наскільки широко розподілені значення від середнього значення (середнього).

Це стандартний статистичний показник. Він обчислюється з використанням методу "упередженого" або "n" на основі всієї сукупності заданих аргументів за наступною формулою:

$$s_N = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2}.$$

де N - загальна кількість значень, а x - значення.

Стандартне відхилення вказує на те, що існує відхилення від середньої (середньої) реакції. Чим вище стандартне відхилення, тим далі елементи даних, як правило, лежать від середнього.

Перцентиль визначає, куди потрапляє певний відсоток результатів.

P-тий процентиль (0 <= P <100) N, упорядкований за збільшенням значень порядку, отримується шляхом обчислення рангу, округлення результату до найближчого цілого числа, а потім прийняття значення, яке відповідає цьому рангу, за такою формулою:

$$n = \frac{P}{100} * N + 1$$

Тут буде 90-й процентиль:

$$n = 0.9 * N + 1$$

90% лінія (90-й процент) - значення, нижче якого опускається 90% зразків. Решта зразків мають принаймні стільки, скільки значення. Це стандартний статистичний показник.

У сукупності ці показники можуть дати нам чітке уявлення про ефективність програми та її вплив, що стосується можливостей, на ландшафт програми.

2.3. Веб-платформи на основі ниток

Щомісяця англійська компанія веб-сервісів Netcraft проводить опитування веб-серверів. За його даними, найзайнятіші сайти працюють на таких веб-серверах: HTTP-сервер Apache, Microsoft IIS (Інформаційний сервер Інтернету) та nginx.

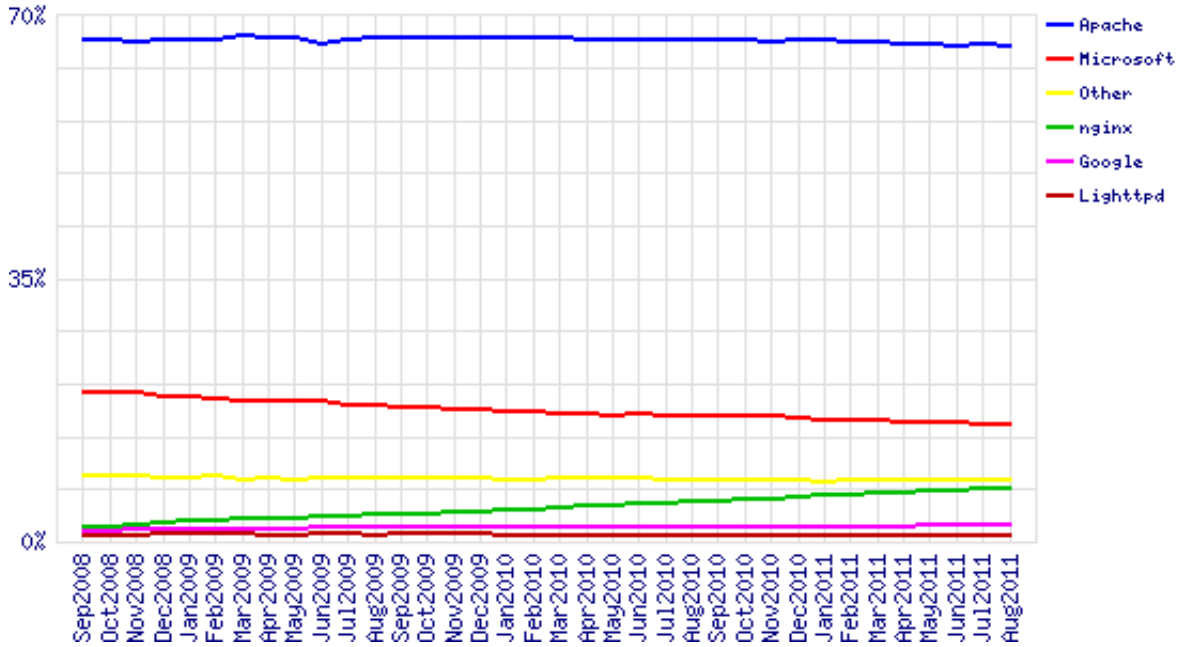


Рис. 0.5 Частка ринку для найкращих серверів на мільйоні найбільш зайнятих сайтів (Вересень 2008 - серпень 2011)

Діаграма показує відсоток веб-сайтів, що використовують різні веб-сервери. Згідно з опитуванням w3techs.com, яке, як стверджується, оновлюється щодня, відсотки цифр дещо відрізняються, але провідні сервери такі самі, як у таблиці 2.1.

Таблиця 2.1

Частка ринку для найкращих серверів на мільйоні найбільш зайнятих сайтів

(Серпень 2011 р.)

Розробник	Активні веб-сайти	Відсоток
Апачі	655 553	65,91%
Microsoft	154 685	15,55%
Nginx	71 517	7,19%
Google	21871	2,20%
Інший	91444	9,15%

Згідно з даними, представленими на малюнку 2.4 та в таблиці 2.1, навіть за значного збільшення кількості сайтів за останні 3 роки [3], частка ринку різного серверного програмного забезпечення дещо змінилася. Лише один із провідних веб-серверів - nginx, отримав певну частку цього місяця, тоді як Apache, Microsoft та Google показали невеликі втрати.

Огляд IIS

Інформаційні служби Інтернету (IIS) - технологія, що входить до складу Windows Server 2008 R2 та Windows 7, яка забезпечує підвищений рівень безпеки, простий в управлінні веб-сервер для розробки та розміщення веб-додатків та веб-служб. Це дозволяє надавати багатий веб-досвід і відіграє центральну роль в об'єднанні технологій веб-платформ на базі Microsoft - ASP.NET, веб-служб Windows Communication Foundation та Windows SharePoint Services.

IIS 7.5 пропонує розширену підтримку для багатьох платформ розробки додатків та доставки медіа-контенту. Модульна архітектура дозволяє легко додавати, видаляти та замінювати будь-який вбудований модуль або сторонні модулі. Завдяки підтримці FastCGI програми на базі CGI та FastCGI працюють

швидше, зберігаючи стабільність. Власні та керовані запити коду обробляються через інтегрований конвеєр, що дозволяє різним фреймворкам додатків працювати в межах одного конвеєру запитів веб-сервера.

2.4. Ключові риси:

Централізоване управління Інтернетом (розширене локальне та віддалене адміністрування можливо за допомогою консолі адміністрування або менеджера IIS на основі графічного інтерфейсу);

Модульна архітектура (забезпечує єдину серверну платформу для розміщення програм PHP та ASP.NET із використанням загального набору інструментів управління);

Масштабована інфраструктура (інтегрована балансування навантаження, віртуальний хостинг та маршрутизація на основі правил);

Високошвидкісне динамічне кешування та стиснення (включає потужне стиснення для динамічного та статичного вмісту та інше кешування на виході);

Удосконалена технологія безпеки та гнучкий контроль захисту сервера.

Огляд Apache

Сервер Apache HTTP - це програмне забезпечення веб-сервера з відкритим кодом, яке зіграло помітну роль у початковому зростанні Всесвітньої павутини. Початковий випуск був у 1995 році і спочатку базувався на коді NCSA HTTPd. З 1996 року він став найпопулярнішим програмним забезпеченням веб-сервера, що використовується. Apache написаний мовою C і переноситься на більшість операційних систем. Він випускається за ліцензією Apache та підтримується Apache Software Foundation.

Сервер Apache HTTP розроблений як потужний та гнучкий веб-сервер, який може працювати на самих різних платформах та в різних середовищах. Продуктивність Apache порівнянна з високопродуктивними веб-серверами, хоча основними цілями є забезпечення надійної обробки запитів у розумні терміни, зменшення затримки та збільшення пропускної здатності. Він не реалізує єдиної архітектури; Apache надає різноманітні модулі багатопроцесорної обробки, що

дозволяє працювати як заснований на процесі, гібридний (потоковий і технологічний) або подібно-гібридний режим, щоб краще відповідати вимогам різних інфраструктур. Отже, вибір правильної робочої архітектури та правильної конфігурації дуже важливий.

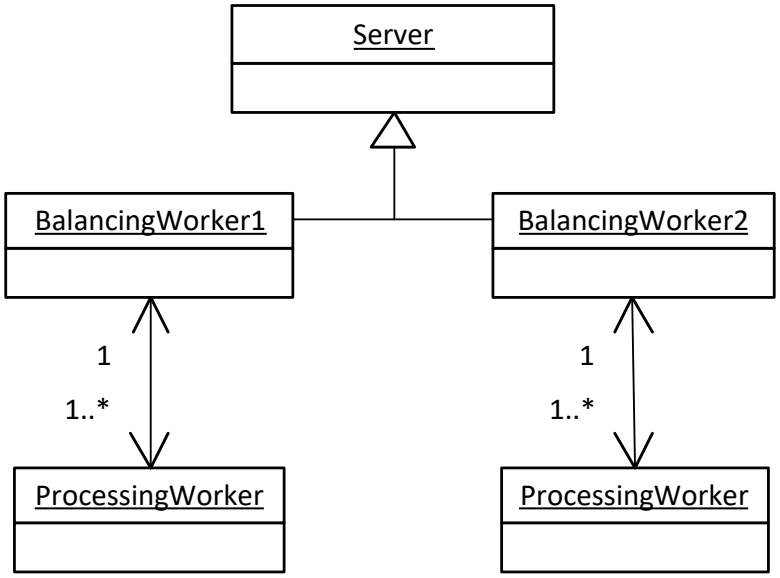


Рис. 0.6 Балансування навантаження за допомогою apache

Багатопроцесорна архітектура працює повільніше, ніж багатопотокова (оскільки потоки мають нижчі накладні витрати, ніж процеси), але вона все одно не відповідає характеристикам архітектур, заснованих на подіях, особливо поєднання подій процесу з декількома робочими потоками.

Врівноваження навантаження (див. Рисунок 2.5) здійснюється за допомогою конфігураційного файлу і, як правило, має 2 робочих процеси балансування, які, в свою чергу, мають декілька обробників. Балансування здійснюється через основний (батьківський) серверний процес.

Однією з найпотужніших функцій є можливість розширити основну функціональність за допомогою скомпільованих модулів. Вони можуть варіюватися від загальної підтримки мови програмування на стороні сервера (інтерфейси для Perl, Python або PHP) до схем автентифікації (mod_access, mod_auth та mod_auth_digest). Інші функції включають: підтримку рівня захищених сокетів і транспортного рівня (mod_ssl), модуль проксі-сервера (mod_proxy), механізм перезапису URL-адрес (mod_rewrite), стиснення вмісту (mod_gzip), механізм виявлення та запобігання вторгненню (mod_security),

підтримка реєстрації та фільтрації. Усі модулі є відкритими та безкоштовними, але їх потрібно скомпілювати у формат динамічної бібліотеки цільової системи, хоча популярні мають попередньо скомпільовані файли.

Функція віртуального хостингу зробила Apache дуже популярним. Це дозволяє обслуговувати (розміщувати) кілька доменних імен (фактичні веб-сайти), з окремим контролем кожного сайту на одному сервері (або пулі серверів). Отже, серверні ресурси можна розподіляти між кількома одночасно запущеними веб-сайтами. Віртуальний хостинг широко використовується провайдерами хостингу і називається загальним веб-хостингом (декілька користувачів використовують один фізичний сервер), він ефективніше використовує апаратні ресурси та забезпечує низьку вартість платформи хостингу веб-сайтів для клієнтів.

Існує кілька типів віртуального хостингу: на основі імен, портів та IP.

Віртуальний хостинг на основі імен обслуговує кілька імен хостів для одного порту та IP-адреси. Він зберігає IP-адресу і може обслуговувати різний реальний шлях для відповідного віртуального шляху.

У віртуальному хостингу на основі IP кожен сайт вказує на окрему (унікальну) IP-адресу, яка вимагає виділеної IP-адреси для кожного доменного імені.

Портуальний віртуальний хостинг обслуговує різні порти для однієї IP-адреси або доменного імені. Він рідко використовується на практиці, за винятком ігрових серверів, оскільки існує стандартизований список портів для різних програм, а за замовчуванням номер порту для HTTP - 80. Крім того, деякі брандмауери блокують зв'язок нестандартних портів, що спричиняє недоступність сайту .

Для досягнення бажаної гнучкості можна поєднати кілька типів віртуального хостингу. Наприклад, сервер може мати кілька IP-адрес і обслуговувати кілька імен на деяких або всіх цих IP-адресах.

Огляд веб-платформи PHP

PHP (PHP: Hypertext Preprocessor) - широко використовувана відкрита кодова мова загального призначення, яка добре підходить для веб-розробки і

може бути вбудована в HTML. Його синтаксис спирається на C, Java та Perl, і його легко вивчити. Документація PHP - це одне з найбільш зручних для розробників посібників, яке містить робочі приклади для кожного запису документації та багато прикладів використання в коментарях. Основна мета - дозволити веб-розробникам швидко писати динамічно створювані веб-сторінки, але це може зробити і набагато більше.

Найбільша перевага PHP перед Perl полягає в тому, що PHP був розроблений для створення сценаріїв для Інтернету, тоді як Perl був розроблений набагато більше. Через це Perl може дуже ускладнитися. Гнучкість / складність Perl може ускладнити співпрацю розробників різних рівнів кваліфікації. PHP має менш заплутаний і жорсткий формат написання, не втрачаючи гнучкості, і має одну з найнижчих перешкод для входу. PHP також простіше інтегрувати в існуючий HTML, ніж Perl. Значною мірою PHP має найкращі функціональні можливості Perl - конструкції, синтаксис тощо, не роблячи це настільки складним, як може бути Perl. Хоча інтерпретатори Perl та Python, поряд із компіляторами C / C ++ входять до складу всіх дистрибутивів ОС Linux, а також у системах Mac OS.

Ці 3 причини, багатоплатформова, безкоштовна та чудова документація призводять до низьких бар'єрів для входу, що робить PHP однією з найпопулярніших мов та мовою веб-розробки №1 [6].

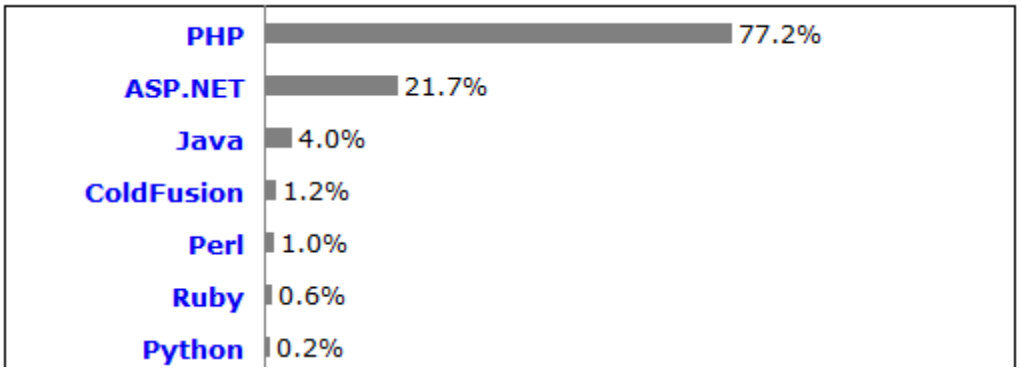


Рис. 0.7 Статистика використання серверних мов програмування веб-сайтів [6,9]

Діаграма показує відсоток веб-сайтів, що використовують різні мови програмування на стороні сервера, що оновлюється щодня.

PHP є інтерпретатором, тому кожного разу, коли обробляється запит, вихідний код інтерпретується та компілюється на льоту у внутрішній формат, який може виконуватися механізмом PHP. Це може бути досить повільним, коли є багато коду та включених файлів. Для того, щоб пришвидшити час виконання та не потрібно компілювати вихідний код при кожному запиті, PHP-сценарії можна розгорнути у виконуваному форматі за допомогою компілятора PHP (акселератора). На відміну від Python, де попередньо скомпільований код створюється після першої компіляції, PHP має зовнішні рішення - прискорювачі PHP. Такі оптимізатори спрямовані на підвищення продуктивності скомпільованого коду шляхом об'єднання надлишкових інструкцій, зменшення його розміру та внесення інших змін для скорочення часу виконання.

Іншим підходом для зменшення накладних витрат на компіляцію є використання кешу коду opcode, який кешує скомпільовану форму PHP-скрипта (opcode) у спільній пам'яті, щоб уникнути накладних витрат на синтаксичний аналіз та компіляцію. Прикладом є альтернативний кеш PHP (APC), безкоштовний фреймворк з відкритим кодом, який оптимізує проміжний код PHP, а також кешує дані та скомпільований код із компілятора байтового коду PHP у спільну пам’ять. APC стає фактичним стандартним механізмом кешування PHP, оскільки він буде включений в ядро PHP, починаючи з версії 5.4 PHP.

Кешування Opcode та оптимізація коду можуть поєднуватися для кращої ефективності, оскільки модифікації відбуваються на різних етапах компіляції. Звідси, мабуть, найбільш функціональна платформа Zend, комерційний продукт, який має повний набір можливостей продуктивності, що включає кешування та прискорення коду, кешування даних, кешування вихідного вмісту, обфускатор коду та файлів, офлайн (асинхронна) обробка та завантаження можливості оптимізації, які можуть призвести до значного покращення продуктивності для більшості програм PHP.

Спочатку розроблений для створення динамічних веб-сторінок, зараз PHP фокусується на сценаріях на стороні сервера, що схоже на інші мови сценаріїв на стороні сервера, які забезпечують динамічний вміст від веб-сервера до клієнта (наприклад, ASP.NET від Microsoft, Java Tomcat від Oracle, тощо). PHP має ряд

основних плат, спрямованих на зменшення накладних витрат, пов'язаних із загальними заходами, забезпечуючи будівельні блоки та різні дизайнерські структури для сприяння швидкій розробці додатків.

2.5. Середовище PHP

Існує кілька різних версій двійкових файлів для ОС PHP Windows - слід вибрати версію, яка підходить для використовуваного веб-сервера:

Якщо PHP використовується з IIS7 (або вище) та PHP 5.3+, тоді слід використовувати бінарні файли VC9 Non Thread Safe PHP.

Якщо PHP використовується з Apache, тоді слід використовувати бінарні файли Thread Safe PHP.

VC9 - це версії, скомпільовані за допомогою компілятора Visual Studio 2008, і мають покращення продуктивності та стабільності, тому вони потребують встановленого середовища виконання Microsoft C ++.

Непотокова безпечна збірка PHP забезпечує підвищення продуктивності порівняно з безпечною збіркою потоків, не виконуючи жодних перевірок безпеки потоків, які не є необхідними, коли FastCGI забезпечує однопотокове середовище виконання.

Існує кілька виконуваних файлів PHP: php.exe - це консольна програма, яка використовується для налагодження та безпосереднього виконання сценарію, тоді як виконувана програма php-cgi.exe використовується під час запуску PHP на веб-серверах через CGI або FastCGI.

CGI (Common Gateway Interface) визначає спосіб веб-сервера взаємодіяти із зовнішніми програмами, що генерують вміст, які часто називають програмами CGI або сценаріями CGI. Це найстаріший, найпростіший і найпоширеніший спосіб розміщення динамічного вмісту на веб-сайті.

При запуску PHP через веб-сервер Apache є два варіанти: запуск через CGI або як модуль для веб-сервера. Apache має процедурний батьківський / дочірній дизайн, тоді як слухач на передній панелі розміщує запити в черзі FIFO (перший у першому вихідному). На даний момент серверний сервер перевіряє наявність

нових процесів, вибирає запит із черги, обробляє запити, а потім Apache вирішує, як з ним обробляти.

Mod_php

Mod_php був розроблений, щоб служити рідним модулем для сервера Apache розробниками програмного забезпечення PHP. У цьому випадку інтерпретатор PHP завантажується в пам'ять із запуском сервера і дозволяє Apache інтерпретувати php-файли (він стає вбудованим всередину сервера). Відсутній зовнішній процес інтерпретації PHP, що призводить до розширеного спілкування.

Недоліки:

PHP із власними розширеннями завантажиться, навіть якщо процес Apache був запущений для сервера статичних файлів (наприклад, зображень або відео), для запуску яких знадобиться порівняно більше часу;

Неефективне використання пам'яті Apache (з інтерпретатором PHP), що може призвести до вимкнення Apache через витоки PHP;

Уповільнює Apache (оскільки це робить роботу PHP);

Помилка PHP в більшості випадків призводить до архітектурного блокування (неможливість обробки запитів);

Неможливо ефективно розподілити навантаження, що балансує ділянку, на окремі машини;

Не підходить для масштабування на декількох серверах.

Переваги:

PHP завантажується як дочірній процес Apache;

Низька затримка зв'язку між PHP та Apache;

Спільна пам’ять;

Середовище PHP із розширеннями завантажується лише один раз.

Коли PHP використовується з mod_php, він успадковує дозволи користувача Apache (як правило, непривілейованих користувачів), що впливає на проблеми безпеки та авторизації; це можна трактувати як перевагу, так і недолік.

CGI

Програми CGI працюють в окремих процесах, які створюються на початку кожного запиту і завершуються в кінці. Ця модель "одного нового процесу на запит" робить програми CGI дуже простими у впровадженні, але обмежує ефективність та масштабованість. За великих навантажень накладні витрати на створення та знищення операційної системи стають значними та обмежують масштабованість. Крім того, модель процесу CGI обмежує методи повторного використання ресурсів (наприклад, повторне використання з'єднань з базою даних, кешування в пам'яті тощо).

На CGI, коли сервер отримує запит, він завантажує середовище PHP, виконує запит, генерує вихідні дані HTML і заголовки, вбиває середовище і очищає все це. Процес повторюється для кожного запиту, що явно не ефективно, оскільки він створює новий процес для кожного запиту, незалежно від того, чи з mod_php він запускається лише один раз. Це також означає, що кожного разу, коли PHP потрібно прочитати файл конфігурації php.ini (див. Рис. 2.7), налаштувати його налаштування, завантажити розширення, а потім розпочати розбір сценарію - багато повторюваних робіт. Початок нового процесу може зайняти набагато більше часу та пам'яті, ніж фактична робота з генерації вихідних даних, особливо коли програму потрібно інтерпретувати.

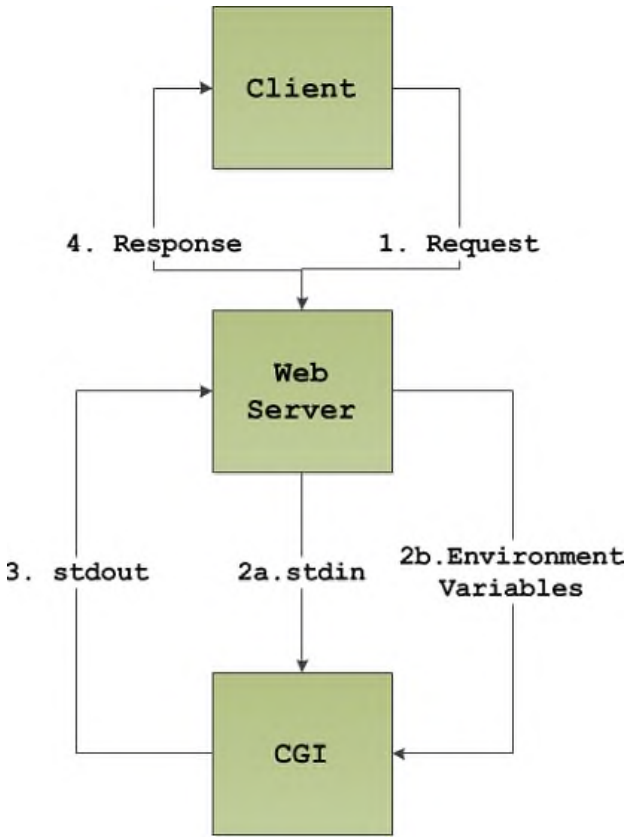


Рис. 0.8 Обробка запитів через CGI

Тож є один величезний недолік - накладні витрати на постійне створення нових процесів.

Переваги перед mod_php:

Завдяки постійному інтерпретованому навантаженню налаштування PHP можна застосовувати на льоту;

Можливість ефективно збалансувати завантаження сайту на різних фізичних машинах, які повністю присвячені PHP;

Адміністратор може призначити різні дозволи користувача для Apache та PHP;

Хороші можливості масштабування.

По суті, запуск від CGI означає повідомлення веб-серверу розташування виконуваного файлу PHP, його запуск та спілкування через загальний інтерфейс шлюзу.

FastCGI

FastCGI - це протокол для взаємодії інтерактивних програм з веб-сервером. Це різновид інтерфейсу Common Gateway; FastCGI був розроблений, щоб зменшити накладні витрати, пов'язані з взаємодією з веб-сервером та програмами CGI, дозволяючи серверу обробляти більше запитів одночасно.

Коли PHP працює через FastCGI, Apache не обробляє дочірні процеси PHP, FastCGI це робить. Це також відокремлює обробку додатків PHP від обслуговування статичного вмісту; отже, Apache стає дещо запитом проксі-сервера та перенаправляє запити динамічних сторінок на FastCGI.

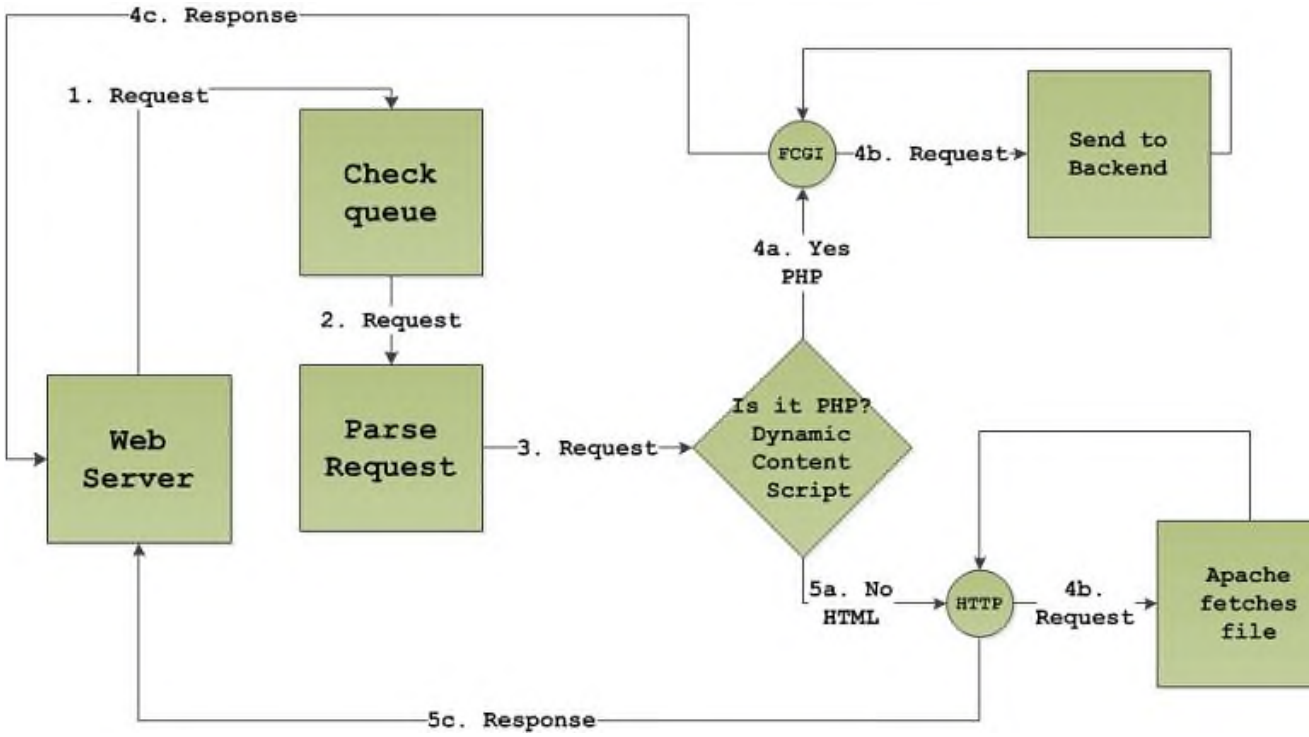


Рис. 0.9 Обробка запитів через FastCGI

FastCGI дозволяє єдиному тривалому процесу обробляти багато запитів, зберігаючи наближення до моделі програмування CGI, зберігаючи простоту, виключаючи накладні витрати на створення нового процесу для кожного запиту. На відміну від перетворення програми на плагін веб-сервера, програми FastCGI залишаються незалежними від веб-сервера.

Для обслуговування вхідного запиту веб-сервер надсилає інформацію про середовище та сам запит сторінки до процесу FastCGI через сокет (у випадку локальних процесів FastCGI на веб-сервері) або TCP-з'єднання (для віддалених процесів FastCGI у фермі серверів) . Відповіді повертаються з процесу на веб-сервер за тим же підключенням, і веб-сервер надає цю відповідь кінцевому користувачеві. З'єднання може бути розірвано в кінці відповіді, але як веб-сервер, так і процеси обслуговування FastCGI зберігаються.

Як і CGI, FastCGI не прив'язаний до внутрішньої архітектури веб-сервера, а тому стабільний навіть при зміні серверної технології. API відображає внутрішню архітектуру веб-сервера, отже, коли архітектура змінюється, змінюється і API.

Запуск PHP під FastCGI має ряд переваг перед тим, як запускати його як CGI:

Apache стурбований лише обробкою статичного вмісту;

FastCGI обробляє всі php-запити окремо від веб-сервера, що означає, що збій виконуваного файлу PHP не вплине на роботу веб-сервера;

На роботу Apache не впливають витоки пам'яті або повільна робота PHP;

FastCGI створює та обробляє певну кількість дочірніх процесів PHP і перезапускає їх при збої;

Дочірні процеси ізольовані від основного веб-сервера, що забезпечує більший захист, ніж API;

Рідне навантаження збалансоване;

Ефект балансування навантаження круглої ваги;

Можливість перезапустити PHP без перезапуску сервера Apache;

Зміни конфігурації можна застосовувати на льоту;

Пачка PHP надає виконуваний файл cgi, тому він працює вбудовано.

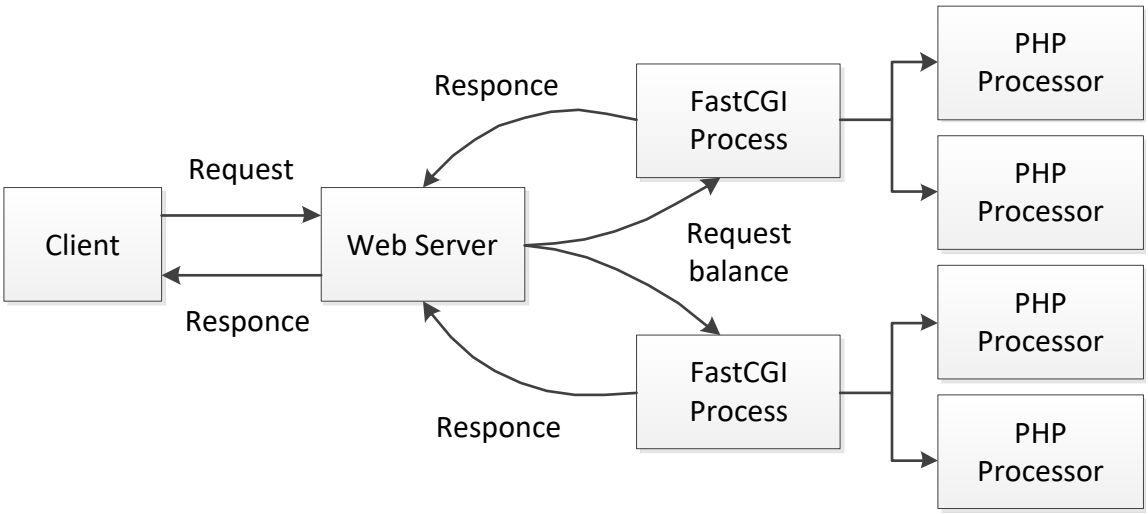


Рис. 0.10 Запитує балансування навантаження за допомогою FastCGI

Кожен окремий процес FastCGI може обробляти багато запитів протягом свого життя, тим самим уникаючи накладних витрат на створення та припинення процесу на запит. Обробка декількох запитів одночасно може бути досягнута кількома способами: за допомогою одного з'єднання з внутрішнім мультиплексуванням (тобто декількох запитів за одним з'єднанням); за допомогою декількох з'єднань; або комбінацією цих методів. Можна налаштувати кілька серверів FastCGI, що підвищує стабільність і масштабованість.

2.6. Підтримка FastCGI в IIS

Модуль FastCGI у IIS дозволяє розміщувати популярні фреймворки програм, що підтримують протокол FastCGI, на веб-сервері IIS з високою продуктивністю (особливо порівняно з режимом CGI) та надійним способом.

Програми CGI запускаються веб-сервером для кожного запиту для обробки запиту та формування динамічних відповідей, які потім надсилаються клієнту. Оскільки багато фреймворків не підтримують багатопотокове виконання, CGI дозволяє їм надійно виконувати роботу в IIS, виконуючи рівно один запит на процес. На жаль, це забезпечує низьку продуктивність через високу вартість запуску та вимкнення процесу для кожного запиту.

FastCGI вирішує проблеми, пов'язані з продуктивністю CGI, забезпечуючи механізм повторного використання одного процесу знову і знову для багатьох запитів. Крім того, FastCGI підтримує сумісність з не безпечними для потоку бінарними файлами та бібліотеками, надаючи пул багаторазових процесів та гарантуючи, що кожен процес обробляє лише один запит одночасно.

Асинхронна модель, керована подіями

У наш час асинхронна модель, керована подіями, стала дуже популярною, і це не дивно, оскільки вона модернізує технологію програмування на новий рівень розвитку та досягнень. Виникають нові фреймворки на основі існуючих спочатку несинхронних середовищ, наприклад Twisted (керований подіями мережевий механізм, написаний на Python і ліцензований під відкритою ліцензією MIT) та Tornado for Python, фреймворк Prado PHP, Perl Object Environment, “EventMachine” для Ruby, фреймворк Mate Flex тощо, а також нові спочатку розроблені для керування подіями технології, такі як Scala (на основі віртуальної машини Java) та NodeJS (на основі механізму V8-javascript).

Причини появи

Зовсім нова лавина експериментів з новими архітектурами була викликана тим фактом, що традиційна архітектура веб-серверів, яка вирішувала всі проблеми в початковій точці зростання Інтернету, не змогла задовольнити розвинуті потреби Інтернету Web 2.0, де все є динамічним і приводиться в рух.

Перевірений багато років пакет пакетів PHP-MySQL-Apache дуже добре обробляв Web 1.0, де для кожного запиту сервер запускав новий крок (а іноді і процес), який пересилав PHP, який, у свою чергу, захоплював деякі дані з бази даних і одна за одною повертали відповідь, остаточно самознищуючись після надсилання відповіді HTTP.

Однак для програм реального часу стабільної та перевіреної роками технології стало недостатньо. Якщо сервер повинен відповісти на 10 000 підключень, він створить безліч потоків, які оброблятиме внутрішній диспетчер завдань операційної системи, який працює подібно до перемикача контексту ЦП. Спочатку запустіть потік, дайте частину часу обробки, потім призупиніть потік, збережіть поточний контекст потоку, а потім обробіть наступний у черзі. Всі ці кроки та те, як часто планувальник завдань перемикає контекст, контролюються операційною системою, а тому є зовнішніми для програми веб-сервера.

Хоча вважається, що перемикання завдань не є дорогим та трудомістким процесом, але воно викликається на всіх синхронних та блокуючих операціях введення / виведення (наприклад, очікування читання БД), навіть найпростіших, і займає близько мікросекунди. Якщо потоки активно читають або записують у різні частини операційної системи, то з ростом таких потоків обсягу кешу L2 процесора буде недостатньо для зберігання всіх контекстів. У цьому випадку контексти зберігатимуться у віртуальній пам'яті, а перемикання завдань буде коштувати набагато більше часу - до 50 мікросекунд. Кеш-пам'ять процесора була розроблена дуже швидко, особливо для вирішення проблеми перемикання завдань, і очевидно, що із використанням великої кількості потоків архітектура стає неефективною.

Інша сторона багатопотокової моделі стосується стеку, пов'язаного з кожним потоком. У стеку містяться дані, поточна адреса, адреса, що повертається, та аргументи, і якщо потік викликає функціональний стек, у свою чергу зберігаються відповідні дані, розміщуючи нові дані поверх попередніх даних, створюючи великий склад даних. Створюючи протектор, операційна система розподіляє стек у віртуальній пам'яті, яка, у свою чергу, складається із частин реальної оперативної пам'яті, так званих сторінок пам'яті. Сторінки

пам'яті виділяються лише у випадку реального використання. Таким чином, на кожному потоці стеків операційна система викликає виняток помилки сторінки, а потім створює сторінку пам'яті у фізичній пам'яті. Для кожного стека операційна система спочатку виділить 1-2 сторінки пам’яті,

Одні потокові сервери не мають проблем із великою витратою пам'яті, оскільки їх складність потоку становить O (1), тому зі збільшенням кількості з'єднань пам'ять не закінчується.

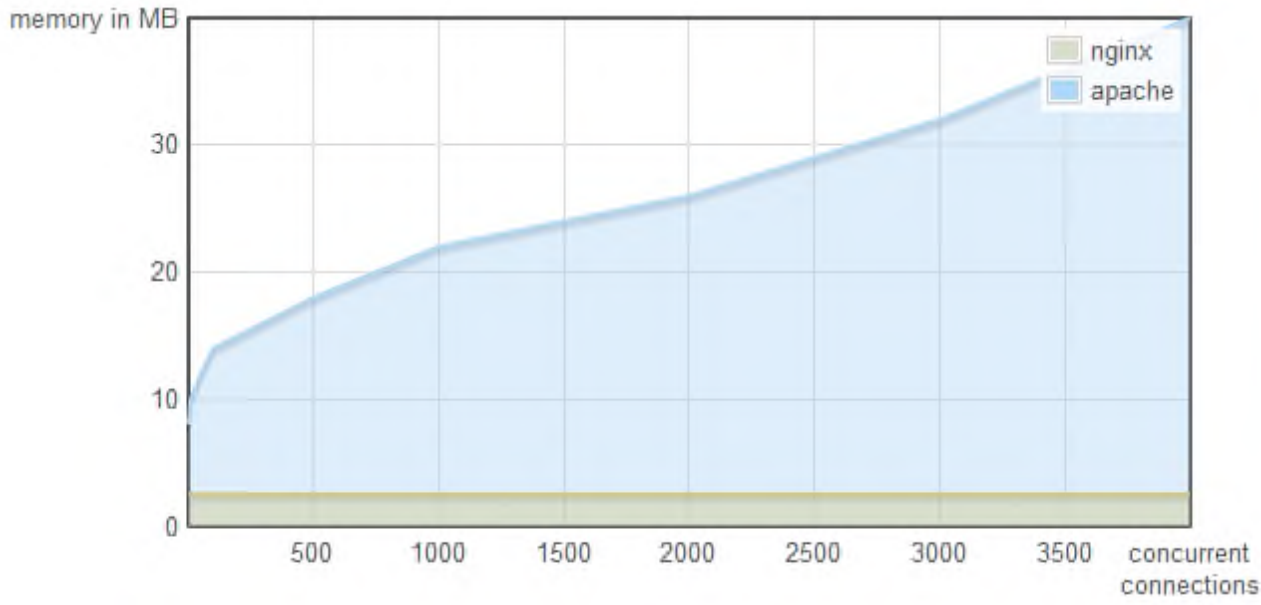


Рис. 0.11 Споживання пам'яті багатопоточних та однопоточних моделей

На ілюстрації показано споживання пам'яті багатопоточних веб-серверів Apache та однопотокових веб-серверів nginx із збільшенням одночасних з'єднань. Звичайно, сучасна реалізація Apache - це оптимізований варіант багатопотокової моделі: є пул підготовлених з'єднань, які обробляються в черзі. Отже, фактична кількість оброблених з'єднань дорівнює розміру опитування, що вирішує проблему надмірного використання пам'яті.

Проблема оптимізації програмного забезпечення веб-сервера для одночасної роботи з великою кількістю клієнтів вивчалась давно, оскільки слід враховувати низку факторів, а отже, отримала узагальнену назву C10k (десять тисяч з'єднань).

Як працює модель, керована подіями

Нова архітектура веб-серверів, керована подіями, була створена для вирішення проблем старої. Він був побудований на основі контуру подій та схеми реактора.

Цикл подій - це нескінченний цикл, який запитує дескриптори (або джерела подій) щодо нових подій, які можна порівняти зі спрощеною версією драйверів апаратного забезпечення операційної системи. Пул виконується за допомогою синхронної бібліотеки вводу / виводу, яка є неблокуючим (O_NONBLOCK) прапором, який повинен передаватися системній функції) вводу / виводу. На кожному циклі він послідовно перебирає всі дескриптори та намагається прочитати події, якщо вони присутні. Події, що повертаються викликом функції читання функції зворотного виклику до системи. У будь-якому випадку, якщо на дескрипторах є або немає нових подій, цикл ітерацій не блокується очікуванням або зчитуванням відповіді [7].

Подія може бути послідовністю пакетів даних у мережевому сокеті або зчитуванням даних буфера з жорсткого диска, будь-якою частиною вводу / виводу даних, яка надходить у систему і чекає подальшої обробки. Наприклад, завантаження зображень надсилається на хостинг-сервер частинами від клієнта через обмежену пропускну здатність клієнта. У цьому випадку дескриптор є вказівником на мережевий буфер даних сокета TCP, за допомогою якого встановлюється з'єднання з хостинговим сервером.

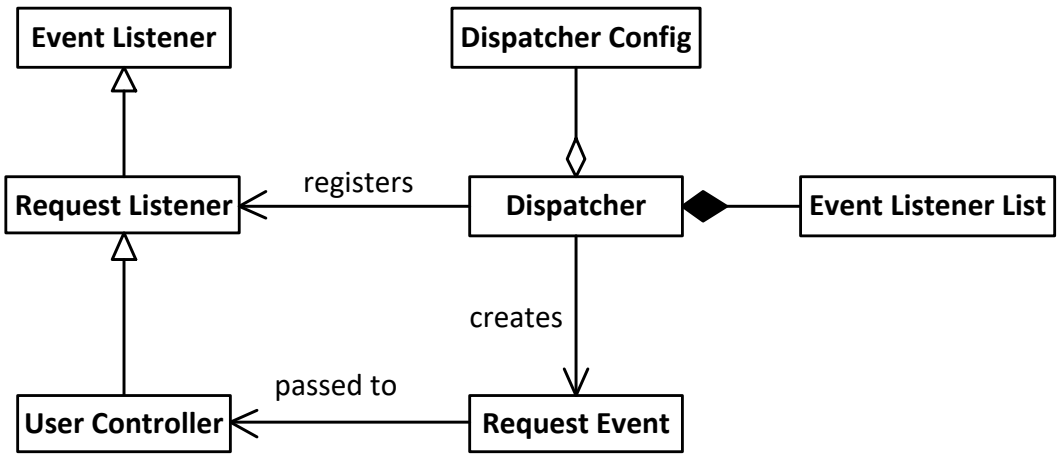


Рис. 0.12 Прив'язка подій у нескінченному циклі повідомлень

Отже, цикл подій (також званий диспетчером повідомлень або цикл повідомлень) - це конструкція програмування, яка очікує та відправляє події (повідомлення) у програму, шляхом опитування зовнішнього диспетчера програми (постачальника подій). Повідомлення закачуються в чергу повідомлень, тому в певному сенсі цикл подій є одним із методів реалізації міжпроцесорного зв'язку. Програми, керовані подіями, можуть бути написані будь-якою мовою, хоча завдання простіше на мовах, що забезпечують абстракції високого рівня, наприклад, закриття. Існує 3 основні кроки для створення програми, керованої подіями: створити серію процедур або методів обробника подій, прив’язати їх до подій sot, щоб при запуску події викликалася правильна функція, та створити основний цикл, який перевірятиме наявність випадків подій, а потім викликати відповідні обробники подій для їх обробки.

Шаблон проекту реактора - це шаблон обробки подій для обробки запитів, одночасно доставлених обробнику одним або кількома входами. Після цього обробник демультиплексує вхідні запити та відправляє їх синхронно до пов’язаних обробників запитів. Код на стороні сервера пишеться не як один великий блок, а як невеликі рутинні блоки, які не виконуються послідовно зверху вниз по всьому списку коду, а частинами, реагуючи на потреби пов'язаної події, що запускається. Отже, код - це сукупність рутинних блоків, які відповідають подіям, які потрібно обробити.

Структура реактора зазвичай складається з ресурсу (забезпечує вхід і вихід у систему), демультиплексора подій (цикл подій, який діє на ресурси і надсилає ресурс диспетчеру), диспетчера (обробляє реєстрацію та скасування реєстрації обробників запитів) та визначеного додатка обробник запитів (обробляє події від диспетчера та пов'язаного з ним ресурсу). Більшість систем конструкцій реакторних конструкцій є однопоточними за визначенням, але можуть існувати в багатопотоковому середовищі.

Величезною перевагою є те, що він повністю відокремлює специфічний для програми код від реалізації схеми реактора, а це означає, що компоненти програми можна розділити на модульні, багаторазові частини. Крім того, завдяки

синхронному виклику обробників запитів, схема реактора забезпечує просту паралельність грубого зерна, не додаючи складності декількох потоків системі.

Але є і мінус: схема реактора важча для налагодження, ніж процедурна, через обернений потік управління.

Для досягнення найкращої можливої продуктивності сервер, керований подіями, повинен уникати блокування будь-якого типу операцій вводу-виводу. Таким чином, на сервері, керованому подіями, використання асинхронного або неблокуючого вводу-виводу є практичною необхідністю.

Асинхронна модель

Асинхронна або неблокуюча модель - це форма обробки, яка дозволяє продовжувати іншу обробку до закінчення попередньої [7].

У моделі синхронного блокування додаток виконує системний виклик, що призводить до блокування програми. Операції введення та виведення можуть бути надзвичайно повільними порівняно з обробкою даних. Наприклад, під час операції з диском, на виконання якої потрібно десять мілісекунд, процесор з тактовою частотою в один гігагерц міг виконати десять мільйонів циклів обробки інструкцій.

Тому загальноприйнятим підходом чекати завершення читання чи запису вводу-виводу та залишати системні ресурси в режимі очікування є неефективне управління ресурсами. Крім того, коли програма виконує багато операцій введення / виведення, це змушує процесор витрачати майже весь час, очікуючи бездіяльності для завершення операцій вводу-виводу. Це означає, що програма блокує до завершення системного виклику (повертає або викликає помилку). Програма, що викликає, перебуває у стані, коли вона не витрачає процесор і просто очікує відповіді, тому є ефективною з точки зору обробки.

Асинхронна модель використовується для поліпшення пропускної здатності, затримки та швидкості реагування. Тут запит на читання негайно повертається, вказуючи на те, що читання було розпочато успішно. Додаток може продовжувати обробку, поки завершується операція фонового читання. Коли надійде відповідь на читання, для завершення транзакції вводу-виводу буде сформовано сигнал або зворотний виклик на основі потоку.

Зворотний виклик - це посилання на виконуваний код або підпрограму, яка передається як аргумент до іншого коду або підпрограми. Це дозволяє програмному рівню нижчого рівня викликати підпрограму (або функцію), визначену на рівні більш високого рівня. Наприклад, програма може не захотіти негайно припинити роботу, коли вона отримає сигнал завершення; щоб переконатися, що за речами подбали та зробили витончений вихід, він повинен передати функцію очищення (або її посилання) як зворотний дзвінок.

Існує два типи зворотних викликів: синхронні (блокуючі) зворотні виклики та асинхронні (відкладені) зворотні виклики [7]. Вони відрізняються тим, як вони керують потоком даних під час виконання, в той час як блокування зворотних викликів викликається до повернення функції, відкладені зворотні виклики можуть бути викликані після повернення функції. Асинхронні зворотні виклики часто використовуються в контексті операцій введення / виведення або обробки подій. Хоча відкладені зворотні виклики передбачають існування декількох потоків, блокування зворотних викликів часто (але не завжди) покладається на один потік. Таким чином, синхронні зворотні виклики не є загальною причиною синхронізації.

Можливість перекриття обчислень та обробки в одному процесі для потенційно декількох запитів вводу-виводу використовує розрив між швидкістю обробки та швидкістю введення / виведення. Поки декілька повільних запитів очікують, процесор може виконувати інші завдання або обробляти вже виконані запити, поки інші запити ініціюються.

Порівняно з іншими моделями, можна сказати, що блокуючі моделі вимагають, щоб ініціююча програма блокувала дескриптор, коли він запускався, а це означає, що неможливо одночасно перекривати подальшу обробку. Синхронна неблокувальна модель вимагає від програми перевірки стану дескриптора на регулярній основі. Хоча асинхронна неблокувальна модель дозволяє перекривати обробку та очікування завершення дескриптора введення / виводу.

Використання асинхронних систем може допомогти побудувати швидші та ефективніші програми вводу-виводу, які ефективніше використовують наявні апаратні ресурси.

Недоліки

Той факт, що додаток працює в асинхронній моделі, керованій подіями, дає два основні недоліки. Перший - це витік пам’яті, другий - обробка винятків та помилок.

Витоки пам'яті необхідні для програмування із зворотними викликами та одним потоком. Якщо у випадку Apache потік має витік пам'яті, то після повернення відповіді потік припиняється, і операційна система обробляє всю виділену пам'ять і можливий витік. У разі зворотних викликів розробник повинен бути обережним, щоб не залишати завислих елементів, які існують між процедурами та не можуть бути видалені збирачем сміття. Інакше процес забиратиме все більше і більше пам’яті з кожним викликом процедури витоку або підключення.

Ще однією великою проблемою є передача помилок. Якщо потік Apache викликає необроблений виняток або просто не вдається, апаш не вплине і відповіді користувачеві з кодом помилки «500 Внутрішня помилка сервера». Хоча модель, керована подіями, є одним великим циклом, який у випадку не обробленого винятку завершується, і користувач взагалі не отримує відповіді. Тому розробник повинен обробляти всі винятки та помилки для витонченого вимкнення сервера або моніторингу та перезапуску сервера вручну.

Крім того, є ще одна неприємна проблема при використанні асинхронної моделі - код спагетті. Код спагетті - це термін для вихідного коду, який має складну та заплутану структуру управління, особливо таку, що використовує безліч зворотних викликів, винятків, потоків або інших неструктурованих конструкцій розгалуження. Не завжди, але часто додаток стає важким для розуміння через численні функції зворотного виклику, які, у свою чергу, містять ще купу функцій зворотного виклику. Одним із рішень для подолання проблеми є використання спеціальної модульної структури додатка, хоча і над основною

процедурою. Іншим рішенням є використання спеціальних бібліотек або фреймворків, призначених для полегшення розуміння коду.

Відповідне використання

Асинхронна модель, керована подіями, буде корисною для тих завдань, де багато користувачів виконують деякі дії одночасно, не роблячи великого навантаження на процесор. Наприклад, датчики температури в режимі реального часу зчитують і обробляють, або відеоспостереження. Обчислення, які не приносять великого навантаження на процесор, спричинені одним циклом подій, який призначений швидким та неблокуючим, тоді як великі обчислення (наприклад, диференціація) змушують цикл чекати його завершення. Це є причиною того, що керовані подіями сервери, такі як NodeJS, підходять лише для задач, що не завантажуються процесором, або як інтерфейс для важкої серверної бази. Вони також хороші для повільних або обмежених з'єднань, де дані надходять як невеликі порції даних; або роль посередника введення / виведення деякого сховища даних.

Якщо програма обробляє декілька одночасних з'єднань і постійно читає та записує в них, то це ідеально підходить для керованої подіями асинхронної архітектури (наприклад, NodeJS, яка підтримує найновіші HTML5 WebSockets). Інші приклади асинхронної моделі, керованої подіями, можуть включати:

система відправлення таксі (яка контролює рух багатьох автомобілів та їх пасажирів, обчислює оптимальні шляхи);

система життєзабезпечення, яка постійно контролює численні дані датчиків і відповідно дає суміші засобів, контролює температуру тощо;

чат або послуга обміну повідомленнями через Інтернет;

масивна багатокористувацька онлайн-рольова гра (MMORPG).

Залежно від завдань, які вирішить проект, архітектор повинен вибрати відповідну архітектуру.

Огляд існуючих засобів тестування навантаження

У цьому розділі будуть розглянуті популярні та найбільш функціональні засоби тестування продуктивності веб-додатків.

Буде розглянуто лише обмежену кількість засобів стрес-тестування, які відповідають наступним критеріям:

Підтримка протоколу HTTP 1.1;

Підтримка запитів GET та POST;

Незалежна від веб-технологій (не спеціалізується на єдиній платформі, як ASP.NET або PHP);

Висока продуктивність (може імітувати і підтримувати сотні активних з'єднань);

Розширюваність функціональності за допомогою плагінів;

Стійкий до несправностей (неправильні відповіді та таймаути підключення не повинні зупиняти запущені тести);

Моніторинг лічильників продуктивності навантаження ОС на тестовій машині;

Вимірювання пропускної здатності та часу відгуку.

Цунг

Tsung (раніше IDX-Tsunami) - це інструмент тестування розподіленого навантаження з відкритим вихідним кодом. Він може бути використаний для наголошення на серверах HTTP, WebDAV, SOAP, PostgreSQL, MySQL, LDAP та Jabber / XMPP. Tsung - це безкоштовне програмне забезпечення, випущене за ліцензією GPLv2. Метою Tsung є імітація користувачів з метою перевірки масштабованості та продуктивності програм клієнт / сервер на основі IP.

Головна сила Tsung - це його здатність імітувати величезну кількість одночасних користувачів на одній машині. При використанні на кластері він може генерувати справді вражаюче навантаження на сервер із скромним кластером, простий у налаштуванні та обслуговуванні. Він може поширюватися на декількох клієнтських машинах і може одночасно імітувати сотні тисяч віртуальних користувачів (або навіть мільйони, якщо апаратне забезпечення може обробляти). Tsung можна використовувати в хмарі, як хмарний сервіс EC2 Amazon.

Він розроблений в Erlang, мові програмування, орієнтованій на паралельну роботу з відкритим кодом, створеній компанією Ericsson для створення надійних

розподілених додатків, стійких до відмов. Tsung базується на Erlang OTP (Open Transaction Platform), що підтримується Process-One.

Основні характеристики Tsung:

Висока продуктивність (може імітувати величезну кількість одночасних користувачів на фізичний комп'ютер);

Генерація розподіленого навантаження (навантаження може розподілятися на кластері клієнтських машин);

Кілька IP-адрес можна використовувати на одній машині, використовуючи основний псевдонім IP ОС;

Моніторинг ОС (процесор, пам'ять і мережевий трафік) за допомогою агентів SNMP або Erlang;

Відмовостійкий (неправильна реакція сервера не призводить до збою всього тесту);

Система конфігурації XML (сценарії користувача написані в XML);

Підтримка складних динамічних сценаріїв, написаних користувачем (наприклад, тестовий цикл сеансу, перезапуск або зупинка, коли рядок або регулярний вираз відповідають відповіді сервера);

Змішана поведінка (кілька сеансів можна використовувати для імітації різних типів користувачів під час одного тесту).

Функції, пов'язані з HTTP:

Підтримка HTTP / 1.0, HTTP / 1.1 та SSL;

ОТРИМАТИ, Опублікувати, ВСТАНОВИТИ, ВИДАЛИТИ та запити HEAD;

Автоматичне та ручне управління файлами cookie;

Базова WWW-автентифікація;

Необмежена модифікація заголовків HTTP та агента користувача;

Режим проксі для запису сеансів за допомогою веб-браузера.

Tsung базується на пакетах і призначений для роботи на Linux та FreeBSD, але повинен працювати на Mac OS та Windows. Він має такі необхідні залежності: Erlang / OTP R12B-5, perl5 та python, бібліотеки модулів (pgsql, mysql, eldap, mochiweb) та bash. Набір інструментів шаблону необхідний для

звітів HTML, для графічного виводу - пакетів gnuplot, mathplotlib та withtsung-plotter. Для розподіленого тестування необхідний доступ ssh до віддалених машин (також підтримується rsh).

Загалом, Tsung - це незалежний від протоколів розподілений інструмент тестування навантаження, який імітує складну поведінку користувача за допомогою файлу опису XML і повідомляє про багато вимірювань у реальному часі. Існує багато інструментів веб-тестування з відкритим кодом, які мають подібну функціональність, але загалом вони не настільки функціональні та ефективні, як на основі орієнтованого на паралельність Erlang Tsung, тому ряд таких інструментів розглядатись не буде.

HP LoadRunner

HP LoadRunner - це автоматизований продукт для тестування продуктивності та тестування від Hewlett-Packard для вивчення поведінки та продуктивності системи під час генерування фактичного навантаження. HP придбала LoadRunner як частину придбання Mercury Interactive у листопаді 2006 року. HP LoadRunner може наслідувати сотні або тисячі одночасних користувачів, щоб застосувати додаток через суворі реальні завантаження користувачів, збираючи інформацію з ключових компонентів інфраструктури (веб-сервери, сервери баз даних тощо). Потім результати можна детально проаналізувати для вивчення причин конкретної поведінки. Це забезпечує ефективні та надійні засоби для перевірки того, що архітектура програми побудована для більш ефективної продуктивності та надійності.

Програмне забезпечення для тестування продуктивності HP LoadRunner можна використовувати для:

запобігати проблемам з продуктивністю, виявляючи вузькі місця перед розгортанням або оновленням системи;

протестувати широкий спектр програм, включаючи новітні RIA, за допомогою технологій Web 2.0, додатків ERP / CRM, а також додатків із застарілими технологіями;

тестувати хмарні та мобільні платформи.

Ненав’язливі монітори продуктивності в режимі реального часу отримують та відображають дані про ефективність з кожного рівня, сервера та компонента системи. Одночасно HP Diagnostics збирає дані рівня додатка та рівня коду. Після завершення перевірки навантаження механізм аналізу HP LoadRunner надає єдиний перегляд даних про продуктивність кінцевого користувача, рівня системи та рівня коду.

Він підтримує лише операційну систему Windows (для запуску повних функцій) і поширюється під власною ліцензією (не безкоштовною), як наслідок, не буде використовуватися як програма для тестування продуктивності.

Інструмент аналізу веб-ємності

Інструмент аналізу веб-ємності (WCAT) - це легкий засіб створення навантаження HTTP, спочатку призначений для вимірювання продуктивності веб-сервера в контрольованому середовищі. WCAT може імітувати тисячі одночасних користувачів, які роблять запити на один веб-сайт або кілька веб-сайтів. Її движок використовує простий скрипт для визначення набору HTTP-запитів, що відтворюються на веб-сервері. Розширюваність забезпечується за допомогою власного коду DLL та стандартного простого API.

Основні особливості WCAT:

HTTP 1.0 та HTTP 1.1;

Підтримка SSL та IPv6;

Розширюваний для обробки будь-якого аспекту запиту або відповіді HTTP;

Багатопотокове тестування;

Підтримує генерацію стрес-тестів з декількох машин;

Розширюється за допомогою плагіна C DLL (динамічні бібліотеки посилань);

Підтримує інтеграцію лічильників продуктивності через IIS та віддалене збирання монітора продуктивності Window;

Вимірює пропускну здатність та час відгуку;

Дуже невелика вага з низькими апаратними вимогами;

Здатний генерувати тисячі одночасних користувачів тестування.

Файли конфігурації та сценарію WCAT мають синтаксис, подібний до C / C ++, із структурою, подібною до XML. Файл конфігурації складається з декількох елементів, які також можуть містити інші елементи або атрибути. Порядок елементів та атрибутів має значення і оцінюється зверху вниз.

Елемент сценарію є кореневим елементом файлу сценарію. Це обов'язковий елемент, який повинен містити принаймні одну транзакцію. Розділ бібліотеки дозволяє вказати розширення DLL та оголосити всі експортовані функції. Заключна частина сценарію - це транзакції, які повинен виконати клієнт WCAT; принаймні один повинен бути визначений. Операції вибираються випадковим чином відповідно до ваги, яка їм надана.

```
scenario
{
    warmup   = 300;
    duration = 300;
    cooldown = 30;

    default {
        setheader {
            name    = "Connection";
            value   = "keep-alive";
        }
        version     = HTTP11;
        statuscode  = 200;
        close       = ka;
    }
    library {…}

    transaction {
        id = "root transaction";
        weight = 100;
        request {
            url = "/";
        }
        sleep {
            delay = 1000;
        }
        close {
            method = reset;
        }
    }
}
```

Рис. 0.13 Структура сценарію WCAT

Хоча WCAT - це інструмент командного рядка, він може бути інтегрований для використання разом із Fiddler (проксі-сервер для налагодження HTTP) через розширення WCAT Fiddler, яке надає графічний інтерфейс для спрощення налаштування сценаріїв сценаріїв для запуску тестів.

WCAT генерує звіти у форматі XML, які можна переглядати за допомогою Internet Explorer та файлу XSLT, що називається "report.xsl", який

розповсюджується разом з WCAT та забезпечує стилістику для спрощення візуального аналізу.

WCAT має тісну інтеграцію з IIS, в результаті працює на Windows NT 5.2 (Windows XP, Windows Server 2003) або новішої версії, і вимагає IIS версії 5.1 або пізнішої.

Apache JMeter

Apache JMeter - це заснований на Java інструмент з відкритим кодом від Apache Software Foundation, призначений для завантаження тестової функціональної поведінки та вимірювання продуктивності. Це багатоплатформна програма, оскільки вона використовує віртуальну машину Java. Спочатку розроблений для тестування веб-додатків, але розширився до інших тестових функцій; може використовуватися для перевірки продуктивності як на динамічних, так і на статичних ресурсах. Він може імітувати велике навантаження на сервер або мережевий об'єкт, щоб перевірити його сильні сторони та проаналізувати загальну продуктивність при різних типах навантаження; може зробити графічний аналіз продуктивності або тестувати поведінку сервера / сценарію / об’єкта при великому одночасному навантаженні.

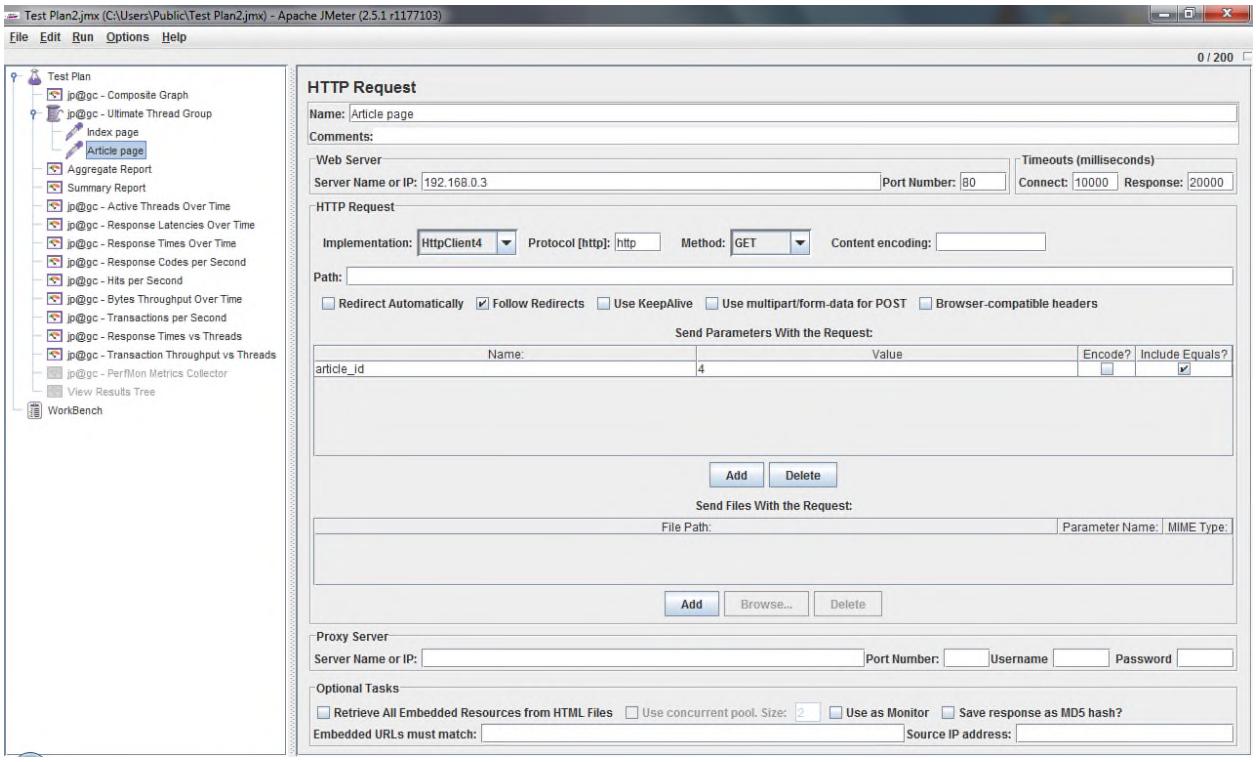


Рис. 0.14 Графічний інтерфейс користувача Apache Jmeter

Це повністю багатопотоковість, яка дозволяє одночасну вибірку для багатьох потоків та одночасну вибірку різних функцій окремими групами потоків.

Особливості Apache JMeter включають:

тестування навантаження та продуктивності різних типів серверів (веб-HTTP та HTTPS, SOAP, база даних через JDBC, пошта POP3 (S) та IMAP (S));

повна портативність завдяки 100% чистоті Java;

ретельно розроблений графічний інтерфейс, щоб забезпечити швидшу роботу та більш точні терміни;

офлайн-аналіз, кешування та відтворення результатів тесту;

висока розширюваність (підключаються пробовідбірники розширюють необмежені можливості тестування і дозволяють вибрати кілька статистичних даних про навантаження за допомогою підключаються таймерів);

плагіни для візуалізації та аналізу даних дозволяють створювати розширені персоналізовані звіти;

скриптові пробники (BeanShell повністю підтримується).

JMeter підтримує параметризацію змінних, твердження (перевірка відповіді), файли cookie та різноманітні звіти. Архітектура базується на плагінах. Більшість чудових “нестандартних” функцій реалізовані за допомогою плагінів. JMeter має лише одну готову опцію для планування потоків (користувачів): просте нарощування, але в його функціональності відсутній більш гнучкий алгоритм планування потоків.

Вибір інструменту для перевірки навантаження

Детальний огляд інструментів тестування показав, що всі вони чудові та забезпечують повний спектр тестових можливостей, тому потрібно зосередитись на основних аспектах, щоб вибрати один інструмент, який буде використовуватися.

Таблиця 2.2

Порівняння засобів тестування

	JMeter	Wcat	Цунг

Написано с	Java	C ++, .Net платформа	Ерланг
Інтерфейс користувача	GUI на основі Java	Командний рядок	Командний рядок
Підтримка ОС	Крос-платформа	Лише для ОС Windows (починаючи з NT 5.1)	Linux, Unix
Залежності	Java Runtime Environment версії 1.5 (і новішої)	Працює з IIS 5.1, IIS 6, IIS 7	Ерланг / OTP perl5 та python баш
Підтримка плагінів	Так	Так; Розширюється за допомогою бібліотек DLL	Так
Управління сценаріями	Через графічний інтерфейс	Зовні як текстовий файл	Зовні як текстовий файл
Створення звіту	Файли CVS та XML	XML-файл	HTML та графічні звіти
Графічне побудова результату	Так	Ні	Так

Apache Jmeter обраний завдяки підтримці декількох ОС, вбудованому графічному інтерфейсу користувача та вдосконаленим інструментам генерації звітів з плагінами Jmeter.

JMeter реалізує архітектуру агента-контролера. У рамках JMeter контролер називається “Master”. Ведучий координує кілька машин агента (він же "підлеглий"). Кожна підпорядкована машина керує кількома робочими потоками. Для того, щоб згенерувати головний та ведений процеси навантаження, запускаються на різних машинах у мережі. Кількість підпорядкованих машин та кількість машин, необхідних для запуску тестів, збільшується із збільшенням навантаження, зазначеного для тестів.

Найфункціональніший набір плагінів “Jmeter plugins” додає Stepping Thread Group до планування потоків JMeter, подібного до LoadRunner. Існує також Ultimate Thread Group, яка забезпечує будь-які гнучкі потреби та забезпечує: початкову затримку, окремий час нарощування, час вимкнення, час польоту для кожного запису розкладу; зменшити навантаження і збільшити навантаження порційно; нескінченна кількість записів розкладу та графік попереднього перегляду завантаження. Починаючи з версії 0.5.0, плагіни Jmeter постачають ServerAgent, який може надавати численні показники від сервера тестування (процесор, пам’ять, навантаження на диск та мережу, інші користувацькі метрики).

2.7. Висновки до розділу

Теорія черг - це математичне вивчення ліній очікування, яке дозволяє проводити математичний аналіз прибуваючих з'єднань у черзі, очікування в черзі та обслуговування. Теорія дозволяє вивести і розрахувати кілька показників продуктивності, включаючи середній час очікування в черзі або системі. За допомогою теореми Літтла можна пояснити міри системи масового обслуговування, як вони впливають на продуктивність системи.

Веб-платформа може оцінюватися за кількома характеристиками, найважливішими серед яких є весь час, час відгуку та доступність.

На сьогоднішній день два архітектурних підходи (керований потоками та подіями) обробки запитів вважаються найбільш ефективними. Перший виділяє внутрішні або зовнішні потоки для обслуговування запитів; останній використовує 1 потік, але прив'язує події ядра системи до виконання асинхронних підпрограм.

РОЗДІЛ 3
ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАСОБУ ВИЗНАЧЕННЯ ВПОДОБАНЬ КОРИСТУВАЧІВ

3.1. Огляд веб-платформи NodeJS

NodeJS - це платформа, побудована на середовищі виконання V8 (механізм JavaScript з відкритим кодом, розроблений Google) для легкої побудови швидких, масштабованих мережевих додатків. Node.js використовує керовану подіями модель, що не блокує введення-виведення, що робить її легкою та ефективною, ідеально підходить для додатків у режимі реального часу, що працюють на розподілених пристроях.

Він був створений Райаном Далом у 2009 році, спонсором якого є Joyent, випущений під ліцензією MIT. Написаний на C / C ++, містить ефективний цикл подій “libuv” та швидкі асинхронні бібліотеки вводу-виводу “libeio”. Він переноситься на багато операційних систем і використовує швидкі внутрішні системні дзвінки. Наприклад, у Windows команда Microsoft допомагала переносити системні дзвінки api низького рівня Windows.

На відміну від більшості програм JavaScript, він не виконується у веб-браузері, а є натомість серверним додатком JavaScript. NodeJS реалізує деякі специфікації CommonJS і забезпечує середовище циклу читання-оцінки-друку для інтерактивного тестування. Подібні середовища, написані іншими мовами програмування, включають Twisted для Python, Perl Object Environment для Perl, libevent для C та EventMachine для Ruby вплинули на розвиток.

Node.js використовує асинхронну модель на основі подій, яка повідомляє операційну систему (через системні виклики дуже низького рівня, такі як kqueue, epoll, / dev / poll або select)

що його слід повідомити, коли з’явиться нове з’єднання, а потім перейде в режим сну (очікування). Якщо встановлено нове підключення, він виконує зворотний виклик з невеликим розподілом купи для кожного з'єднання.

Модель події йде трохи далі - представляючи цикл подій як мовну конструкцію, а не як бібліотеку. В інших системах є блокуючий виклик для запуску циклу подій; зазвичай визначає поведінку через зворотні виклики на початку сценарію та в кінці запуску сервера через блокуючий виклик. У Node немає виклику “start-the-event-loop”, оскільки він просто входить у цикл подій після виконання вхідного сценарію. Вузол виходить із циклу подій, коли більше немає зворотних викликів для виконання. Така поведінка схожа на JavaScript браузера, де цикл подій прихований від користувача, що не дивно, як Node на основі того самого механізму JavaScript.

Двигун V8 реалізує ECMAScript, як зазначено у ECMA-262, 5-е видання. Хоча це не браузерний ECMAScript, але модифікована версія з низкою специфічних функцій, вона має подібні до загальнопризначених мовних бібліотек високого рівня, які називаються модулями. Його висока продуктивність пояснюється компіляцією вихідного коду JavaScript до власного машинного коду перед його виконанням, а не виконанням байт-коду або його інтерпретацією. Подальше підвищення продуктивності досягається за допомогою різних методів оптимізації, таких як вбудоване кешування, написання спеціальних кодів збірки оптимізацій.

Обробка виділення пам’яті для об’єктів та збір сміттєвих об’єктів здійснюється автоматично для зменшення можливих витоків пам’яті.

Є три ключові сфери роботи V8:

Швидкий доступ до властивостей (більшість механізмів JavaScript використовують структуру даних, схожу на словник, як пам’ять для властивостей об’єкта, що є повільним, оскільки кожен доступ до властивостей вимагає динамічного пошуку для визначення розташування властивості в пам’яті. Щоб зменшити час доступу, V8 не використовує динамічного пошуку , він динамічно

створює приховані класи за лаштунками, що дозволяє використовувати класичну оптимізацію на основі класів та вбудований кеш);

Динамічне формування машинного коду (вихідний код компілюється безпосередньо в машинний код при його першому виконанні, без використання проміжного байтового коду або інтерпретатора);

Ефективний збір сміття (для забезпечення швидкого розподілу об’єктів, спочатку збір сміття призупиняє виконання програми, потім обробляє частину купи об’єктів у більшості циклів збору сміття, що мінімізує час очікування програми та, нарешті, уникає помилкового ідентифікації об’єктів як покажчиків, що може призвести до пам’яті витоки).

У V8 купа об’єктів сегментована на дві частини: простір для нещодавно створених об’єктів та простір для об’єктів, що переживають цикл збору сміття. Коли об’єкт переміщується в циклі збирання сміття, V8 оновлює всі вказівники на об’єкт. Його точний збирач сміття є одним із ключів до продуктивності. Таким чином, він підходить для швидкого виконання великих програм JavaScript.

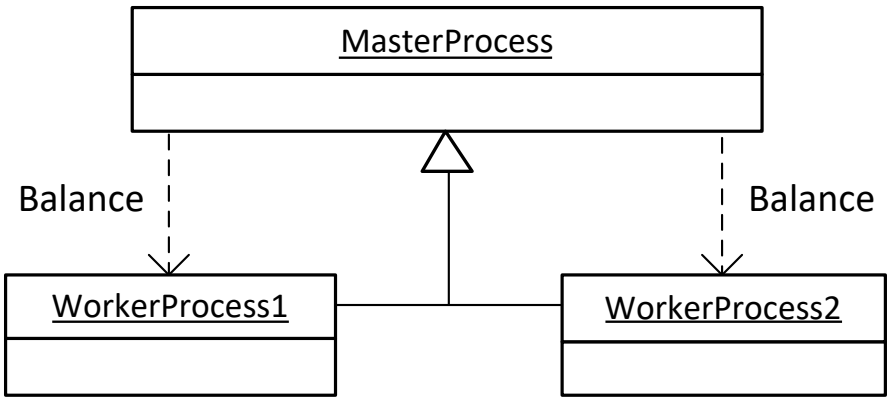


Рис. 0.1 NodeJS балансування навантаження

Тут балансування навантаження є властивістю робочих процесів, тому вони одночасно є і робочими, і самобалансуючими процесами. Вони встановлюються не конфігурацією, а самим додатком. Процес головного (батьківського) сервера після розгалуження робітників вільний від подальших обчислень.

Node.js демонструє набагато кращу ефективність пам'яті при великих навантаженнях, ніж системи, які виділяють кілька мегабайтових стеків потоків для кожного з'єднання. Крім того, у ньому відсутні блокування процесів -

блокування немає, і майже жодна функція в Node.js безпосередньо не виконує введення / виведення, тому процес ніколи не блокується.

Нарешті, Node має вбудовані багатопроцесорні функції одночасності та масштабованості, а також балансування навантаження між різними процесами. Ця функція називається кластеризацією, коли єдиний порт tcp спільно використовується між кількома процесами (і відповідно циклами подій), кластери яких ідентичні, але це дозволяє скористатися перевагами багатопроцесорних систем. Балансування навантаження працює у фоновому режимі і приховане від розробника.

3.2. Налаштування середовища NodeJS

Набори NodeJS доступні для більшості популярних платформ, а також можуть бути скомпільовані для інших. Для встановлення вручну слід використовувати команду bash “make install”, у попередньо розпакованому каталозі або використовувати інтегрований менеджер пакетів Linux.

Для ОС Windows існує інсталятор, який автоматично пов'язує виконуваний файл, щоб бути видимим у будь-якому каталозі в командному рядку. Оскільки NodeJS представляє один виконуваний файл, він не має файлу конфігурації і не записує жодних журналів на диск, якщо це не вказано програмою або включеним модулем.

Щоб перевірити правильність встановлення, можна перевірити версію NodeJS, виконавши такі дії:

Відкрийте командний рядок cmd.exe або Windows PowerShell;

Введіть “node –v”, який повинен відображати використовувану версію NodeJS (як показано на малюнку 0.2).

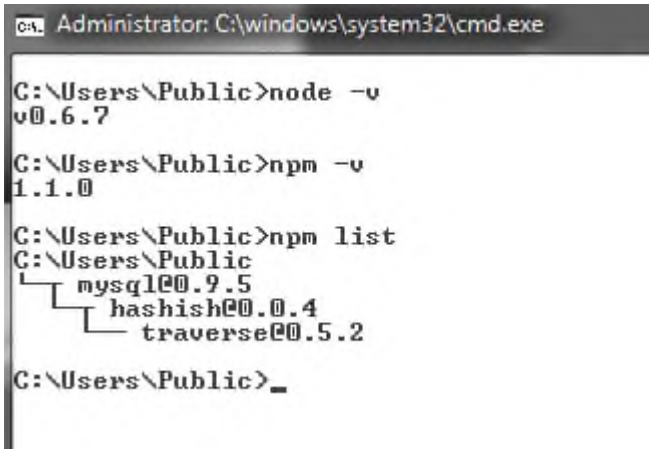


Рис. 0.2 Правильні відповіді правильно працюючих NodeJS та npm

На скріншоті відображаються результати робочого середовища NodeJS та правильно встановленого npm, хоча номери версій можуть бути різними.

Щоб запустити програму з іменем файлу “app.js” (js зазвичай використовується розширенням файлу для NodeJS, але не обов’язково), слід ввести таку команду: “node app.js” або “node app”.

За замовчуванням він працює як єдиний процес, але може створювати дочірні процеси. Помилки програми або NodeJS відображаються лише в консолі, з якої її було запущено.

Порт для прослуховування повинен бути вказаний у файлі програми.

Хоча для більшості потреб існує ряд внутрішніх модулів, Node також підтримує сторонні модулі. Найзручніший спосіб управління сторонніми модулями - це використання менеджерів пакетів, що походять від простоти та великої зручності менеджерів пакетів Linux. Переважним менеджером пакетів для NodeJS є "Диспетчер пакетів вузлів" (npm).

NPM - це менеджер пакетів для Node.js, який проходить через командний рядок та управляє періодичними залежностями для модулів сторонніх програм. Починаючи з версії Node 0.6.3, npm розгортається та встановлюється автоматично із середовищем, але вимагає версії 0.6.x або новішої. Він побудований на базі NodeJS, тому має зручну пряму залежність.

Для встановлення npm в будь-якій операційній системі слід завантажити вихідний код і зв'язати основний файл програми "cli.js" з виконуваним вузлом.

Іншим рішенням є використання “curl http://npmjs.org/install.sh | команда shh bash у відповідному каталозі.

Пакети в реєстрі npm не є частиною самого npm, а надаються співавторами або авторами модулів.

Драйвер MySQL для програми слід встановити за допомогою наступної команди, але заздалегідь перейшовши до потрібної папки, оскільки вона буде встановлена до поточного каталогу:

npm встановити mysql

Потім використовуйте команду “npm ls” або “npm list”, щоб перевірити успішність встановлення, як показано на малюнку0.3. Існує кілька драйверів MySQL для NodeJS, але згаданий вважається найбільш стабільним, оскільки він працює через сокети tcp.

Хоча балансування навантаження виконується автоматично основним процесом, воно програмується та налаштовується з додатка.

3.3. Налаштування IIS

IIS встановлюється та вмикається з коробки лише в операційних системах Windows Server, для встановлення в інших системах слід дотримуватися офіційного посібника з встановлення. Етапи включають відкриття «Панелі управління», вибір «Програми», а потім «Увімкнення та вимкнення функцій Windows» (як показано на малюнку0.3). Детальний посібник з встановлення можна знайти на офіційному веб-сайті http://learn.iis.net.

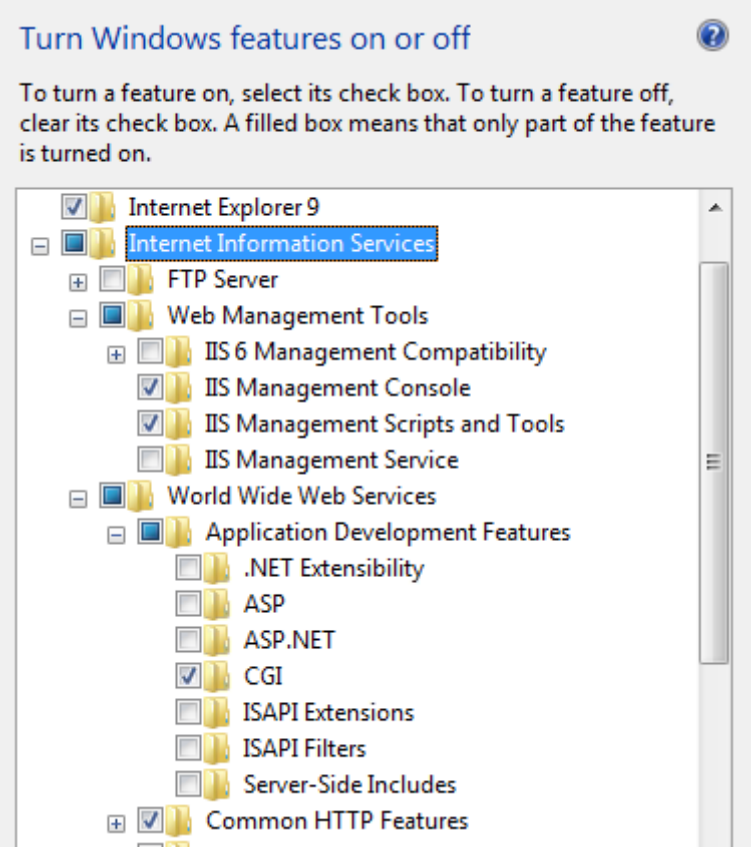


Рис. 0.3 Налаштування, які включають ISS і дозволяють запускати сценарії CGI

На скріншоті показані необхідні налаштування для ввімкнення ISS з підтримкою PHP. Основними функціями є “Консоль управління ISS”, “Загальні функції HTTP” та “CGI”, за допомогою яких виконуватиметься виконуваний файл PHP. За замовчуванням модуль FastCGI відключений, тому, якщо IIS вже встановлений, слід увімкнути прапорець “CGI”, який включає як послуги CGI, так і FastCGI.

Не потрібно перезапускати операційну систему після закінчення інсталяції, оскільки вона працює негайно. Щоб перевірити стан, відкрийте браузер і введіть “localhost”, який повинен показати сторінку привітання.

Наступним кроком є встановлення та налаштування PHP. Рекомендується використовувати непотокову безпечну збірку PHP із IIS FastCGI. Непотокова безпечна збірка PHP забезпечує значний приріст продуктивності, не виконуючи жодних перевірок безпеки потоків, які не є необхідними, оскільки IIS FastCGI забезпечує однопотокове середовище виконання.

Спочатку завантажте найновіший непотоковий безпечний пакет з бінарними файлами PHP, встановіть або розпакуйте файли. Потім перейменуйте файл «рекомендований php.ini» у «php.ini», відкрийте його. Розкоментуйте та змініть такі налаштування:

fastcgi.impersonate = 1

fastcgi.logging = 0

cgi.fix_pathinfo = 1

cgi.force_redirect = 0

розширення =. / ext / php_mysqli.dll

FastCGI під IIS підтримує можливість видавати себе за маркери безпеки викликаючого клієнта, що дозволяє IIS визначати контекст безпеки, під яким виконується запит. Про реєстрацію CGI піклується IIS; і fix_pathinfo надає * реальну * PATH_INFO / PATH_TRANSLATED підтримку CGI, що змусить PHP CGI фіксувати свої шляхи відповідно до специфікацій. Ці модифікації конфігурації середовища PHP підходять і повинні використовуватися як серверами IIS, так і Apache HTTP.

Щоб увімкнути підтримку mysql для програм PHP, потрібно включити низькорівневу бібліотеку, яка реалізує необхідний протокол у файл конфігурації.

Розширення mysqli було розроблено, щоб скористатися новими функціями, наявними в MySQL 5 та новіших версіях; він поставляється в комплекті з версії PHP 5 і настійно рекомендується використовувати над старими розширеннями через низку переваг: об'єктно-орієнтований інтерфейс, підтримка декількох і підготовлених операторів, підтримка транзакцій та розширені можливості налагодження.

Наступним кроком є налаштування IIS для обробки PHP-запитів: у меню «Пуск» Windows виберіть «Виконати», введіть «inetmgr» та натисніть «Ok». З'явиться головне вікно інтерфейсу користувача менеджера IIS.

Для встановлення виконуваного дескриптора PHP .php-запитів виберіть «Отображення обробників», а потім на правій панелі «Дії» натисніть «Додати відображення модуля».

У діалоговому вікні, що з’явилося, потрібно заповнити всі поля (як показано на малюнку 0.4):

Шлях запиту: * .php

Модуль: FastCgiModule

Виконуваний файл: Шлях до виконуваного файлу “php – cgi.exe”

Назва: FastCGI

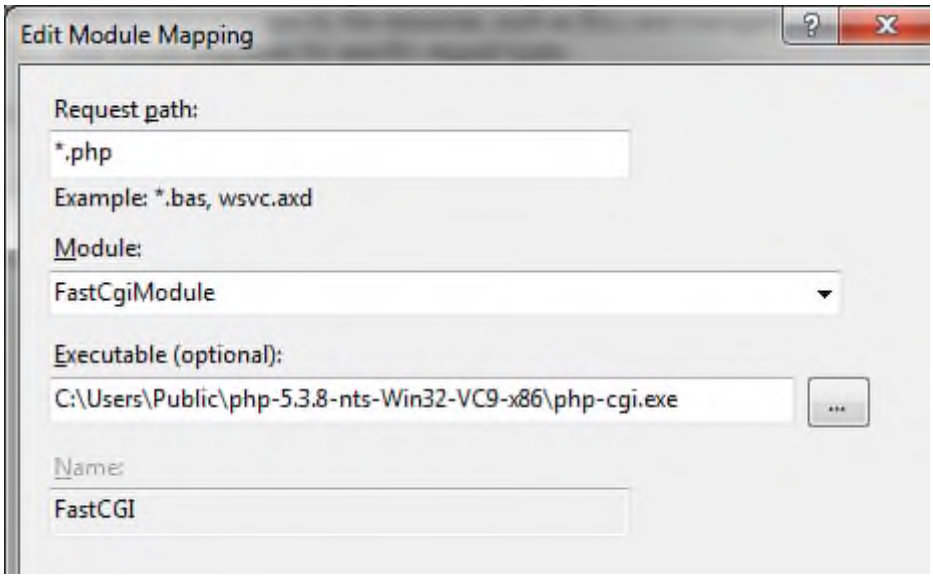


Рис. 0.4 Призначення * .php-файлів до обробника PHP за допомогою модуля FastCGI

У діалоговому вікні підтвердження «Додати відображення модуля» натисніть «Так», щоб створити програму FastCGI для цього виконуваного файлу. Тепер файли з розширенням .php будуть виконуватися виконуваним файлом php-cgi.

Для зручності слід додати файл індексу за замовчуванням PHP до папки. Виберіть “Документ за замовчуванням” і клацніть “Додати” на панелі “Дії”: введіть “index.php” у поле імені.

Для встановлення високої продуктивності та повного використання кількох ядер процесора слід налаштувати «Параметри FastCGI».

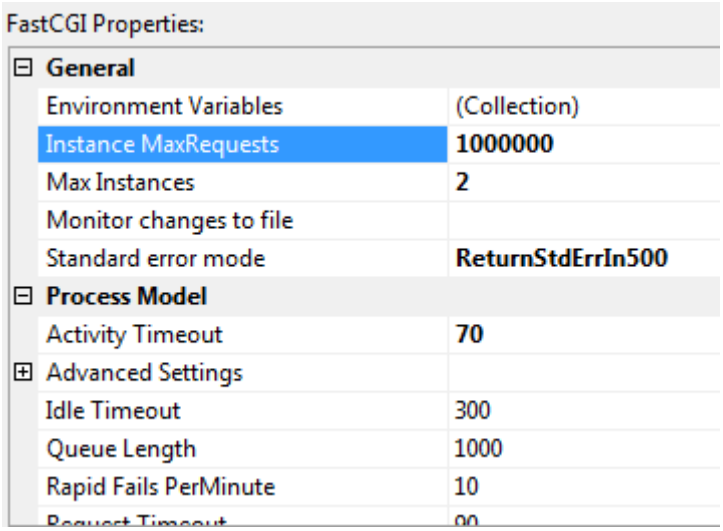


Рис. 0.5 Налаштування властивостей FastCGI

Спочатку потрібно переконатись, що FastCGI переробляє процеси PHP до того, як увімкнеться власне утилізація. Поведінка переробки контролюється властивістю конфігурації instanceMaxRequests, яка визначає, скільки запитів буде оброблено перед переробкою. PHP має подібну функціональність переробки процесів, яка контролюється змінною середовища PHP_FCGI_MAX_REQUESTS.

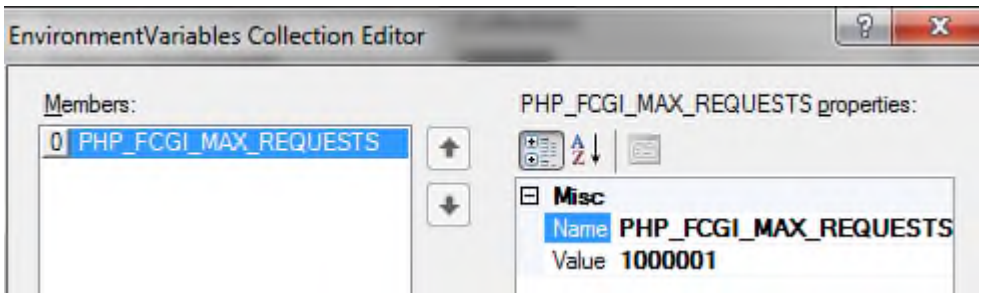


Рис. 0.6 Змінні середовища для виконуваного PHP

Встановивши instanceMaxRequests меншим або рівним PHP_FCGI_MAX_REQUESTS, ви гарантуєте, що вбудована логіка переробки PHP-процесів ніколи не з’явиться.

MaxInsistancевстановлює максимальну кількість процесів FastCGI для дозволу в пулі процесів програми для обраної програми FastCGI. Це число також представляє максимальну кількість одночасних запитів, які може обробляти програма FastCGI.

Щоб встановити поведінку та спосіб обслуговування додатка робочим процесом або набором робочих процесів, слід змінити пул додатків. Пул додатків встановлює межі для програм, які вони містять; ці межі перешкоджають

застосуванню програм в одному пулі додатків впливати на програми в іншому пулі програм.

На лівій панелі виберіть «Пули програм», а потім «DefaultAppPool». “Керований режим ipeline” має бути встановлений на “Інтегрувати”, додатково “Версію .NET Framework” можна встановити на версію 4.

Останній крок - встановити шлях як корінь для каталогу веб-сайту. Виберіть "Сайти" на лівій панелі та "Основні налаштування" на правій панелі. У діалоговому вікні «Редагувати сайт» виберіть «DefaultAppPool» як пул програм і оберіть необхідний фізичний шлях до файлів сайту (наприклад, «C: \ Users \ Public \ site»).

Конфігурація IIS для обробки файлів .php через FastCGI завершена і повинна відображати вміст вибраної папки, коли запитується http: // localhost / address.

3.4. Налаштування HTTP-сервера Apache

Бінарні файли Apache доступні для більшості популярних платформ, і оскільки це продукт з відкритим кодом, його можна скомпілювати для будь-якої конкретної платформи.

Інсталяційний пакет ОС Windows встановлює Apache HTTP Server як системну службу та забезпечує програму моніторингу служб. У Unix він працює як демон, який постійно виконується у фоновому режимі для обробки запитів.

Програма монітора служб Apache відображає поточний стан сервера (і відповідно сервісу), його версію та платформу, запущені модулі (на малюнку 0.7 - mod_fcgid), і дозволяє запускати, зупиняти та перезапускати службу. Є додаткова прихована функція - перевірка правильності конфігураційного файлу. Якщо файл конфігурації містить помилки конфігурації, служба не запускається, але замість цього відображається опис помилок у консолі. Помилки також будуть записані у файл журналу помилок.

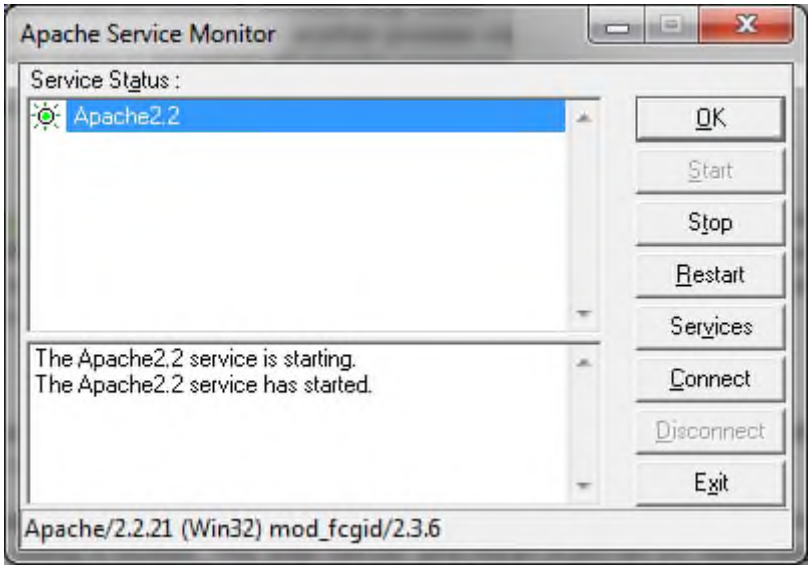


Рис. 0.7 Apache Service Monitor

Apache починається з читання конфігураційного файлу, який знаходиться в каталозі програми “conf” і називається “httpd.conf”. Після того, як сервер запустив та виконав попередні дії (наприклад, відкриття файлів журналів), він запускає кілька дочірніх процесів, які виконують роботу з прослуховування та відповіді на запити клієнтів. Це контролюється вибраним багатопроцесорним модулем. Модулем багатопроцесорної обробки операційних систем Windows NT за замовчуванням є “mpm_winnt_module”; він використовує єдиний процес управління, який запускає дочірні процеси, які, в свою чергу, створюють потоки для обробки запитів.

Конфігурація за замовчуванням визначає порт прив'язки Apache 80 та обробляє запити на адресу домену “localhost” на локальній машині.

Сервер Apache HTTP налаштовується шляхом розміщення директив у текстових файлах конфігурації.

Директива - це слово або поєднання слів, яке розпізнається Apache або його модулями і призначене для представлення унікальної властивості чи аргументу. Директиви не враховують регістр, але аргументи, що їх дотримуються, часто чутливі до регістру.

Основний файл конфігурації “httpd.conf” встановлюється під час компіляції, але додаткові файли конфігурації можуть бути додані за допомогою директиви Include. Директиви, розміщені в цьому файлі, застосовуються до всього сервера. Файл конфігурації Apache повинен містити одну директиву на

рядок. Рядки, які починаються з символу хешу "#", є коментарями та ігноруються.

Сервер Apache HTTP - це модульний сервер, що означає, що під час компіляції на основний сервер включається лише найосновніша функціональність; розширені функції доступні за допомогою модулів. Якщо сервер компілюється для використання динамічно завантажених модулів, тоді модулі можуть бути скомпільовані окремо та додані в будь-який час за допомогою директиви LoadModule. В іншому випадку Apache потрібно перекомпілювати для додавання або видалення модулів.

На першому кроці модуль FastCGI “mod_fcgid” слід завантажити з офіційного сайту Apache, а потім помістити в каталог “модулі”. Для завантаження модуля у файлі конфігурації слід вказати наступний рядок:

LoadModule fcgid_module модулі / mod_fcgid.so

Наступним кроком є встановлення відображення обробника FastCGI у файли .php.

Обробник - це внутрішнє представлення Apache дії, що виконується на основі типу файлу неявним або явним модулем.

Картування модуля fast-cgi для обробки файлів із розширенням .php

AddHandler fcgid-script .php

Дана команда використовується для створення процесів сервера FCGI.

FCGIWrapper "C: /Users/Public/php-5.3.8-Win32-VC9-x86/php-cgi.exe" .php

Файли з розширенням .php тепер будуть виконуватися обгорткою PHP FastCGI.

Наступним кроком є налаштування специфічних директив модуля FastCGI для обслуговування під великим навантаженням.

Директива FcgidMaxProcesses встановлює максимальну кількість процесів програми FastCGI, які можуть бути активними одночасно.

FcgidMaxProcesses 2

Управління дочірніми процесами PHP (PHP_FCGI_CHILDREN) завжди слід вимикати за допомогою mod_fcgid

FcgidInitialEnv PHP_FCGI_CHILDREN 0

FcgidMaxRequestsPerProcess 0

Визначення змінної середовища PHP

FcgidInitialEnv PHP_FCGI_MAX_REQUESTS 0

За замовчуванням процеси PHP FastCGI виходять після обробки 500 запитів, і вони можуть вийти після того, як модуль вже підключився до програми та надіслав наступний запит. Якщо встановити значення 0, PHP не обмежує кількість запитів.

Останніми кроками є прив’язка віртуального шляху до реального каталогу та встановлення властивостей обробки.

Псевдонім / "C: / Users / Public / site /"

<Розташування />

Додайте індексний файл за замовчуванням для читання з папки

DirectoryIndex index.php

Увімкнути виконання cgi

Параметри ExecCGI

AllowOverride контролює, які директиви можна розміщувати у файлах .htaccess

AllowOverride None

</Location>

Директива Псевдонім дозволяє зберігати документи в локальній файловій системі, крім кореневої частини документа. Тут він використовується для обробки кореневої URL-адреси “localhost” до каталогу сайту на диску.

Директива <Location> визначає набір параметрів, застосованих до вказаного шляху URL-адреси та всіх його під-шляхів.

Така конфігурація запускає два PHP-процеси, балансує навантаження між ними і дозволяє обробляти тисячі запитів.

Повний текст конфігураційного файлу міститься в додатку А.

Розробка тестових додатків

Основний акцент робиться на порівнянні різних архітектурних моделей: асинхронних, що базуються на подіях та загальновживаних потоків. Тому

функціональність буде подібною до тієї, яка використовується на більшості популярних сайтів: в блогах та на сторінках новин.

Інтерфейс (розмітка HTML, дизайн та всі файли на стороні клієнта) не є важливим, тому він скопійований з новинного веб-сайту чемпіонату Євро-2012. Адаптований вигляд дизайну сайту наведено в додатку.

Оскільки тест додатків не повинен охоплювати всі функції та функціональні можливості сайту, відтворюється та програмується лише основна функціональність. Зокрема, короткий огляд усіх статей на першій сторінці та перегляд окремих статей новин на спеціальних сторінках.

Перша сторінка - це головна або коренева сторінки, яка не має ніякого зв'язку із жодною конкретною статтею.

Зміст статей буде міститися в базі даних.

База даних Логічний та фізичний дизайн

Логічний дизайн бази даних - це розробка логічної структури бази даних без будь-якої проекції на конкретну фізичну систему баз даних, методи або реалізацію. Це розробка без будь-яких обмежень для певної системи управління базами даних, структур зберігання, методів доступу тощо за допомогою сутностей, відповідних атрибутів, відносин та їх правил.

Компоненти логічного аналізу проектування:

Сутність - це реальний або уявний об'єкт предметної області, який чітко ідентифікує предмет, що цікавить, з точки зору інформаційної моделі;

Атрибути - це деякі особливі властивості сутності, які виділяють серед усіх атрибутів сутності набір атрибутів, що однозначно ідентифікують суть. Існують факультативні та обов’язкові атрибути;

Відносини - це названа асоціація двох сутностей;

Правила - це обмеження, що застосовуються до основних компонентів інформаційної моделі (сутності, атрибути та відносини).

У заявці потрібна лише одна сутність - сутність статей. Ця організація містить усі статті інформаційного сайту.

Атрибути сутності статей:

Id (обов’язково);

Заголовок (обов’язковий);

Зміст (обов’язково).

Правила: Id - це первинний ключ, який однозначно ідентифікує цю сутність.

Далі, перетворення концептуального логічного проекту в реляційну модель, де сутність перетворюється в окрему таблицю. Кожен атрибут перетворюється на стовпець. Факультативні атрибути стають стовпцями NULL, обов’язковими - NOT NULL. Компоненти унікального ідентифікатора сутності стають первинними ключами. Для більш адекватного перетворення даних із логічної моделі бази даних у фізичну визначається спеціальний стовпець з обмеженнями цілісності.

Таблиця 3.1

Реляційна модель сутності статей

Назва стовпця	Тип (довжина)	Опис	Обмеження цілісності
ідентифікатор	ціле число (10)	Унікальний ідентифікатор	Первинний ключ
заголовок	varchar (255)	Заголовок	Обов’язково
зміст	варчар (16536)	Зміст	Обов’язково
Обмеження цілісності таблиці		Усі стовпці є обов’язковими	

Фізичний дизайн бази даних - це реалізація логічного проектування бази даних до конкретної фізичної системи бази даних щодо її методів та властивостей реалізації. На цьому етапі логічні сутності перетворюються у фізичні сутності (таблиці) з конкретною реалізацією методів та властивостей залежно від системи фізичних баз даних.

База даних планується зберігати в реляційній системі управління базами даних MariaDB, яка є безкоштовною форкою системи управління базами даних MySQL з деякими вдосконаленнями, але підтримуючи високу точність і оновлені оновлення з останнім випуском MySQL, забезпечуючи можливість заміни бінарною еквівалентністю бібліотеки і ідентичне узгодження з API, протоколами, структурами та командами MySQL. Таким чином, пакет клієнтських з'єднувачів (драйверів) MySQL працює з сервером MariaDB, а також файли даних та визначення таблиць (.frm-файли) є бінарними.

Операція з базою даних виконується за допомогою реляційних запитів мовою SQL.

Модель логічної бази даних легко трансформується у реляційну фізичну, оскільки логічна модель базувалася на реляційній структурі даних. Всі відношення в логічній моделі є окремими таблицями у фізичній, оскільки логічна модель була доведена до третьої нормальної форми. Отже, в реляційній базі даних є одна таблиця.

Сценарій створення:

```
СТВОРИТИ ТАБЛИЦЯ `статті` (
`id` INT (10) НЕ ПІДПИСАНО НЕ НУЛЕВО AUTO_INCREMENT,
`title` VARCHAR (255) NOT NULL COLLATE 'latin1_general_ci',
`content` VARCHAR (16536) NOT NULL COLLATE 'latin1_general_ci',
ПЕРВИННИЙ КЛЮЧ (`id`)
)
ЗБІРАТИ= 'latin1_general_ci',
ДВИГУН = ПАМ'ЯТЬ;
```

Механізм зберігання MEMORY використовується, оскільки він створює спеціальні таблиці із вмістом, який зберігається в пам'яті, маючи таким чином найкращу продуктивність. Зберігання в пам'яті має ряд обмежень, порівняно з іншими механізмами зберігання, але забезпечує швидкий доступ до даних та низьку затримку. Сервер БД гарантує, що обсяг даних повністю поміститься в пам’яті, не змушуючи операційну систему міняти сторінки віртуальної пам’яті.

Оскільки дані вразливі до збоїв, відключень електроенергії або будь-яких інших апаратних проблем, буде створена дзеркальна таблиця резервного копіювання на основі повільного енергонезалежного механізму зберігання з єдиною метою постійного збереження даних.

Для забезпечення масштабованості, одночасності та доступності слід використовувати кластер MySQL, який пропонує ті самі функції, що і движок MEMORY з вищими рівнями продуктивності, а також надає додаткові функції, недоступні в MEMORY: автоматичне розподіл даних між вузлами, додаткова підтримка диска та підтримка тексту -орієнтовані типи даних (включаючи BLOB та TEXT).

Реалізація за допомогою PHP

PHP не управляє обробкою запитів HTTP, прослуховуванням портів та обслуговуванням статичних файлів, він обслуговує лише вміст HTML, набагато простіший у програмуванні та підтримці коду. Процесор PHP виконує код у підході зверху вниз.

Першим кроком є підключення до бази даних. Додаток викликається і виконується для кожного запиту, що означає, що одночасні підключення до бази даних будуть ініціалізовані при паралельних запитах. При великому навантаженні це може спричинити помилку “Забагато підключень”, тому для ефективної роботи програма використовує постійне (постійне) з’єднання з базою даних. Такий підхід повторно використовує одне підключення для декількох запитів і ініціює одне, якщо з’єднань немає. Кількість постійних одночасних з'єднань та час їх очікування в режимі очікування контролюються параметрами БД.

```php
<?php

$db = new mysqli('p:localhost', 'root', '', 'test');

if(isset($_GET['article_id']))
{...}
else
{...}


require_once "content_generator.php";

printHeader($title);

isset($article_id) ?
    show_article($title, $content)
    :
    show_index($content) ;

printFooter();
```

Рис. 0.8 Спрощений код PHP

Наступним кроком є перевірка, яку сторінку запитують. Якщо присутній параметр “article_id”, витягніть один рядок із бази даних з кореспондентським ідентифікатором; ще - короткий опис усіх статей витягується з бази даних.

Окрім основного файлу, є додатковий файл генерації вмісту, основне призначення якого - вмістити шаблон HTML-коду та правильно об'єднати дані з бази даних. Верхній та нижній колонтитули є загальними частинами всіх сторінок.

Рис. 0.9 Схема послідовності обробки запитів PHP

Нарешті, якщо запит стосується статті, вміст статті додається до відповідного HTML-макета і повертається. В іншому випадку виводиться коренева сторінка з відповідним вмістом і макетом.

Будь-який вихідний результат, що генерується PHP, повертається на веб-сервер, який встановлює відповідні заголовки відповідей, надсилає вміст і закриває з'єднання без участі розробника PHP.

Повний вихідний код доступний у додатку В.

Реалізація за допомогою NodeJS

Програмування NodeJS базується на зворотних викликах та закриттях, стандартних техніках для асинхронної моделі, керованої подіями.

Діаграма послідовності (див. Рис. 3.10) показує, як протікає потік, що обслуговує запити.

Перший крок - запуск сервера з основного файлу, який вказує, які модулі або додаткові файли слід завантажити та оцінити.

Перед початком основного циклу підключення до бази даних ініціалізується, а потім повторно використовується усіма запитами, які обробляються робочим процесом NodeJS. Отже, буде однакова кількість незалежних з’єднань БД, як робочі процеси, і якщо робочий процес вийде з ладу або завершиться, це не вплине на всю програму.

Далі слід визначити основний цикл подій. Основними завданнями яких є прив'язка порту прослуховування та виклику заданої процедури зворотного виклику при кожному запиті.

http.Server (функція (запит, відповідь) {

...

}). слухати (80);

На відміну від PHP, NodeJS повинен не тільки обслуговувати вміст HTML, але також обробляти http-з'єднання та обслуговувати статичні файли. Функція зворотного виклику отримує два аргументи: запит (містить всю інформацію про запит клієнта) та відповідь. Останній - це дескриптор відкритого з'єднання (тут, потік TCP), який повинен бути закритий програмою.

Рис. 0.10 Схема послідовності обробки запитів NodeJS

Далі програма обробляє заголовки запитів HTTP, аналізує URL-адресу і відповідно викликає відповідний метод. Якщо було запитано HTML-сторінку, тоді дані з БД отримуються та передаються в процедуру управління вмістом. Генерація вмісту виконана аналогічно PHP і має кілька функцій для генерації верхнього та нижнього колонтитулів та злиття даних з БД. Якщо запитується файл ресурсу (не інтерпретується), він зчитується з файлової системи у двійковому режимі до потоку відповідей.

Кожен раз перед надсиланням даних відповіді програма встановлює заголовки відповідей, а потім надсилає відповідь HTTP і замикає з'єднання.

Балансування навантаження здійснюється автоматично, але робочі потоки повинні визначатися додатком. Кластерний модуль допомагає скористатися перевагами багатоядерних систем:

```
if (cluster.isMaster) {
 // Вилкові робітники
 for (var i = 0; i <num_of_CPUs; i ++) {
 cluster.fork ();
 }
} ще {
 // Робочі процеси запускають цей блок
 ...
}
```

Повний вихідний код доступний у додатку С.

Налаштування Jmeter

Спочатку слід створити план тестування. План тесту - це місце, де вказані загальні налаштування тесту.

Потім, додавши Ultimate Thread Group до плану тестування, визначте очікуване завантаження активних користувачів, тривалість та поведінку. Були встановлені такі параметри:

Кількість ниток (кількість користувачів для імітації): 200

Початкова затримка, сек: 0

Час запуску (час створення загальної кількості потоків), сек: 200

Утримуйте навантаження протягом, сек: 30

Час відключення, сек: 20

Дія, вжита після помилки семплера: продовжуйте

Після встановлення параметрів відображається графік очікуваного навантаження щодо тривалості випробування.

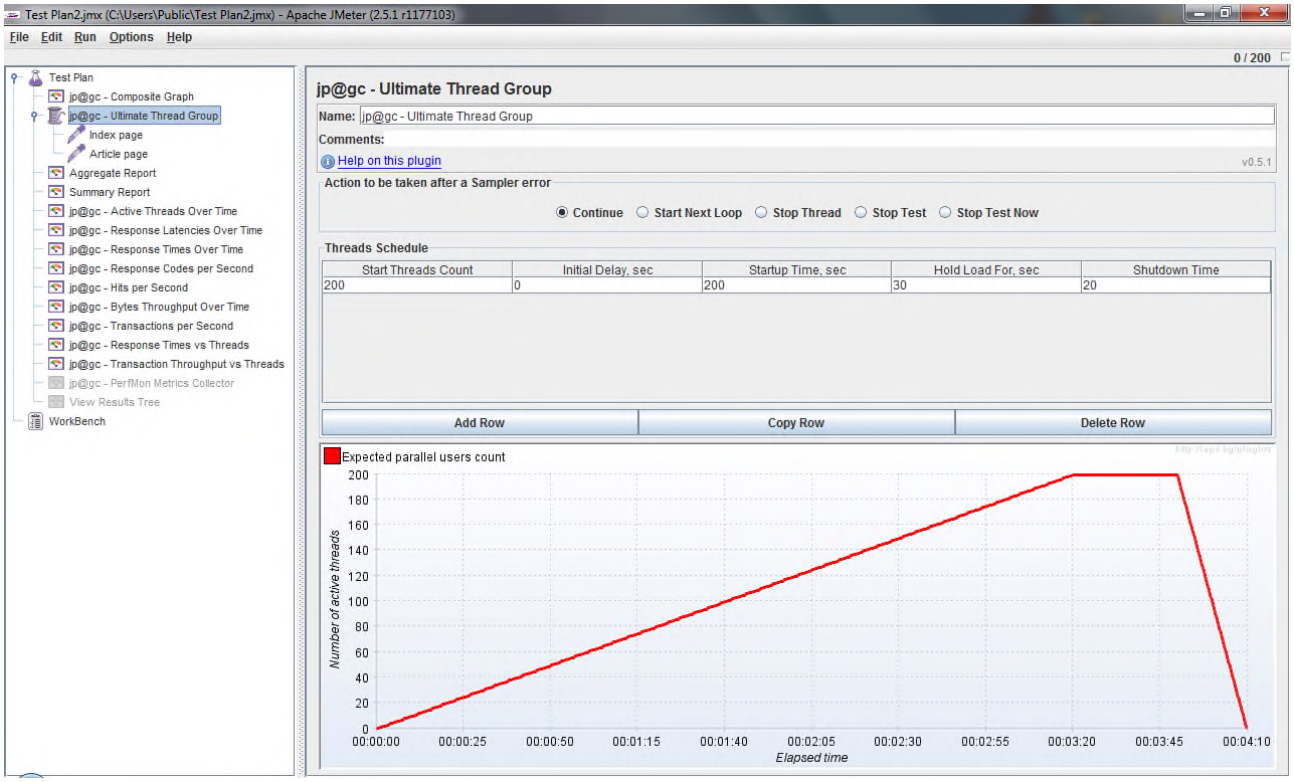


Рис. 0.11 Кінцева група ниток

Групи потоків виконують план тестування незалежно від інших потоків, тому їх можна комбінувати для моделювання великого навантаження.

Наступним кроком є додавання семплера - HTTP-запит. Пробовідбірник визначає параметри запитів, надісланих на веб-сервер. Були встановлені такі параметри:

Ім'я або IP-адреса сервера: 192.168.0.3

Номер порту: 80

Час очікування (кількість мілісекунд очікування, перш ніж викликати виняток)

Підключення: 10000

Відповідь: 20000

Впровадження HTTP:

Java (використовує реалізацію, надану JVM);

HTTPClient4 (частина Apache HttpComponents)

Реалізація Java має кілька обмежень, тому використовується HTTPClient4.

Протокол: HTTP (за замовчуванням)

Метод: GET

Використовувати Keep-Live (встановлює заголовок Keep-Live-з'єднання): не позначено (воно не працює належним чином із реалізацією HTTP або Jakarta httpClient за замовчуванням, оскільки повторне використання з'єднання не контролюється користувачем)

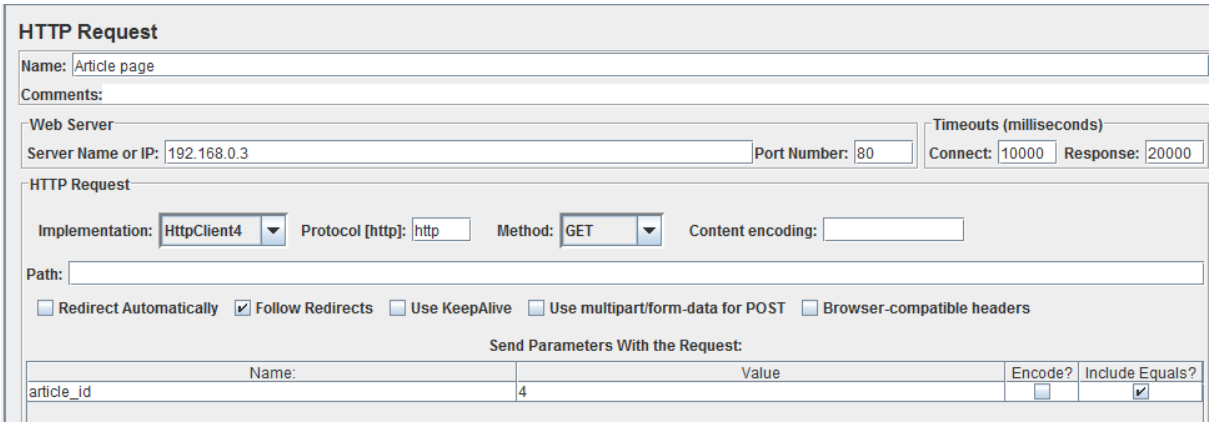


Рис. 0.12 Налаштований запит HTTP

Конфігурація для тестування сторінки індексу завершена, але для доступу до сторінки статті додатковий параметр повинен бути надісланий із запитом. Додано параметр article_id зі значенням 4.

Рядок запиту URL-адреси буде сформовано зі списку параметрів, правильно; залежно від вибору “Методу” (якщо GET або DELETE, рядок запиту буде доданий до URL-адреси, якщо POST або PUT, то він буде надісланий окремо). Крім того, можна вказати кодування URL-адреси кожного параметра.

Тестування

Бенчмаркинг (тестування) веб-серверів - це процес оцінки продуктивності веб-сервера для того, щоб з’ясувати, чи може сервер обслуговувати досить

велике навантаження. Він призначений для визначення швидкості реагування, пропускної здатності та надійності системи за певного навантаження.

Результативність вимірюватиметься за такими ключовими параметрами:

кількість поданих запитів за секунду (звернень за секунду);

час відгуку в мілісекундах кожного запиту;

пропускна здатність в байтах за секунду.

В експерименті порівнюються характеристики (метрики) двох різних реалізацій комбінованого фреймворку. Проведено експеримент із трьома різними конфігураціями робочого навантаження, який буде проаналізовано.

Першим кроком є визначення фізичного тестового середовища та виробничого середовища. Фізичне середовище включає апаратне, програмне забезпечення та конфігурації мережі. Поглиблене розуміння всього тестового середовища з самого початку забезпечує більш ефективне тестування.

Конфігурація тестової машини:

Процесор: Intel Pentium IV 3,2 ГГц

Фізична пам’ять: 2048 МБ

Операційна система: Windows 7 Ultimate SP1, 32-розрядна операційна система

Мережевий адаптер: Гігабітний контролер Marvell Yukon 88E8001

Зв'язок між тестом і ведучим: пряме з'єднання, 100 Мбіт / сек

Головний принцип - змінити одне (тут веб-платформа), залишити все те саме, спостерігати за тим, що відбувається в кожному конкретному випадку, та інтерпретувати спостереження, щоб відповісти на запитання.

Виконання тесту продуктивності передбачає багаторазове виконання нарощування (без кроку) від 1 до 200 віртуальних користувачів (паралельні з'єднання). Така реалізація покаже, як параметри тестування змінюються на різний рахунок паралельних користувачів.

Віртуальний користувацький трафік для завантаження програми буде максимально наближений до реального трафіку. На сайт потрібно завантажувати віртуальних користувачів, які виконують завдання, які виконують реальні

користувачі. Друга, головна машина, генерує навантаження, записує дані відповіді та підтримує всі нові та очікувані з'єднання з тестовою системою.

Однією з головних цілей стрес-тестування є визначення кількості користувачів, з якими може працювати веб-сервер, перш ніж видавати повідомлення про помилки або час відгуку, що перевищує допустимі рівні.

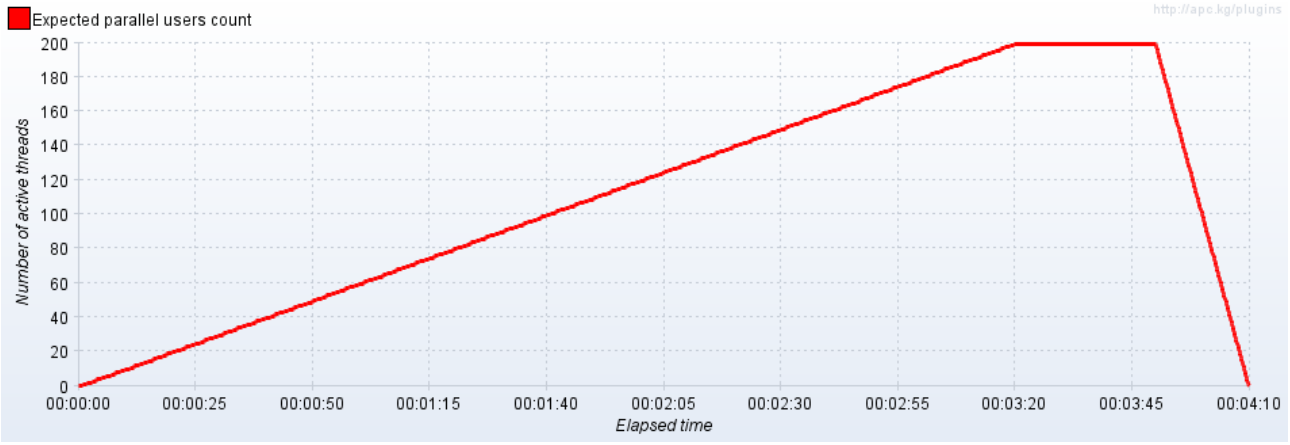


Рис. 0.13 Очікуване (сформоване) завантаження активних користувачів

Графік показує очікувану кількість активних одночасних віртуальних користувачів під час пробного запуску. Фактична кількість активних користувачів повинна відповідати налаштованому шаблону робочого навантаження, але необов'язково повинна бути однаковою.

Інтервал групування встановлений як 1 сек.

Паралельно з точки зору інструменту тестування продуктивності - це кількість активних користувачів, створених програмним забезпеченням, яка часто сильно відрізняється від кількості віртуальних користувачів, які фактично отримують доступ до тестованої програми.

За тестовим навантаженням максимальна кількість одночасних з'єднань становила 200, що більше, ніж швидкість пропускання будь-якої веб-платформи. Підключення, які не обслуговувались у поточному таймфреймі, чекали 10 секунд, щоб викликати помилку очікування підключення. Якщо така ситуація трапляється, це вважається станом перевантаження в середовищі додатків. Припускаючи прийнятне базове значення = 10, величина відхилення від поточного часу очікування при цільовій пропускній здатності визначатиме успіх чи невдачу.

Результати експерименту

Основними характеристиками, які демонструють ефективність та стабільність веб-платформи під навантаженням, є час відгуку та загальна швидкість.

Час відповіді - це час, пройдений клієнтом від підключення до тестового сервера додатків до отримання останнього байта відповіді. Іншими словами, це тривалість від початку підключення до кінця відповіді.

Швидкість пропускання показує кількість з'єднань (звернень), які сервер здатний обробляти щосекунди з поточним навантаженням. Ця кількість залежить від завантаження сервера (кількості підключень) та типу користувальницької діяльності, яку виконує користувач. Важливо вивчити, яку пропускну здатність сервер зміг витримати при зростаючому навантаженні.

Складений графік ємності (див. Рис. 3.14) перекриває час відгуку, відсоток помилок та кількість активних користувачів на кожному інтервалі часу тесту. Потужність системи з точки зору одночасних користувачів - це точка на графіку, після якої ні відсоток помилок, ні час відгуку не виходять за межі допустимих рівнів.

За цим графіком можна ідентифікувати максимум одночасних користувачів, які додаток може обслуговувати сервер швидко і правильно, без помилок. Точка підрахунку активних користувачів на графіку, при якій час відгуку або відсоток помилок перевищує встановлену межу, є максимальною пропускною здатністю системи.

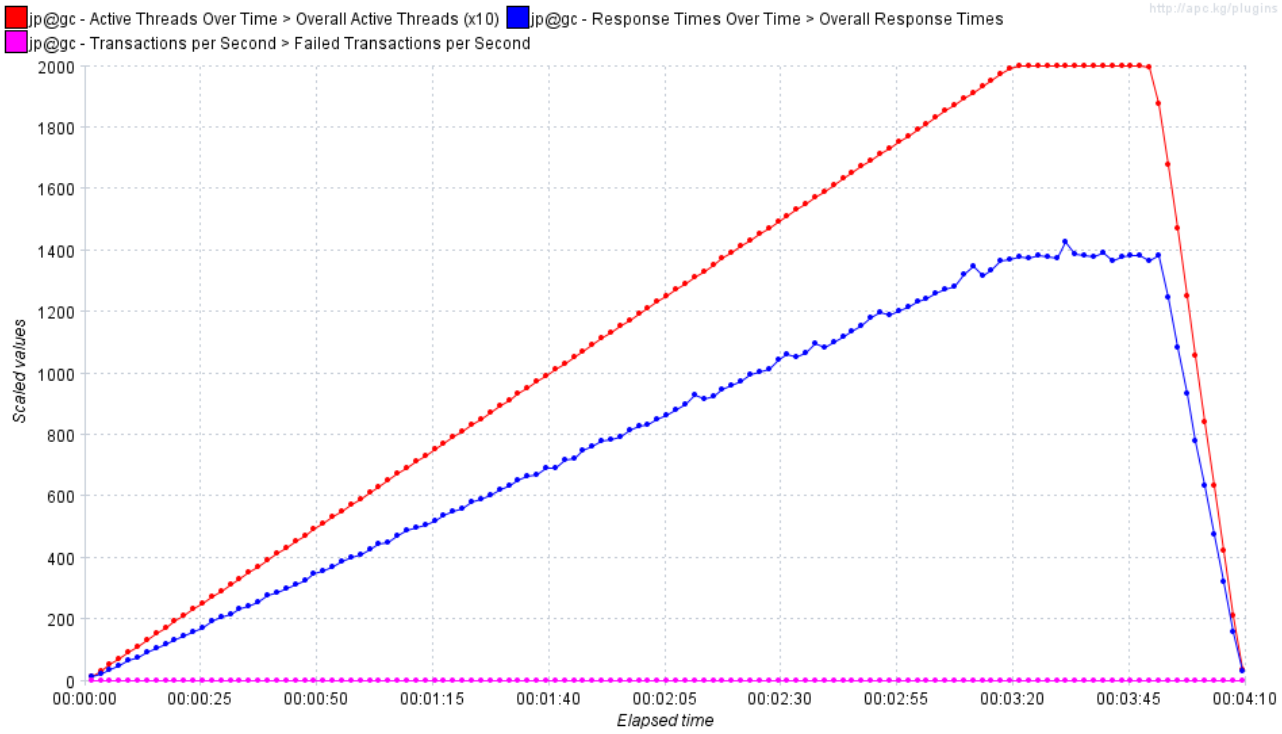


Рис. 0.14 Складений графік NodeJS (активні потоки та час відгуку)

NodeJS продемонстрував чудову продуктивність та ідеальний рівень помилок; - жоден запит не відмовляв у обслуговуванні. Графік часу відгуку повністю відповідає застосованому навантаженню (активні паралельні з'єднання).

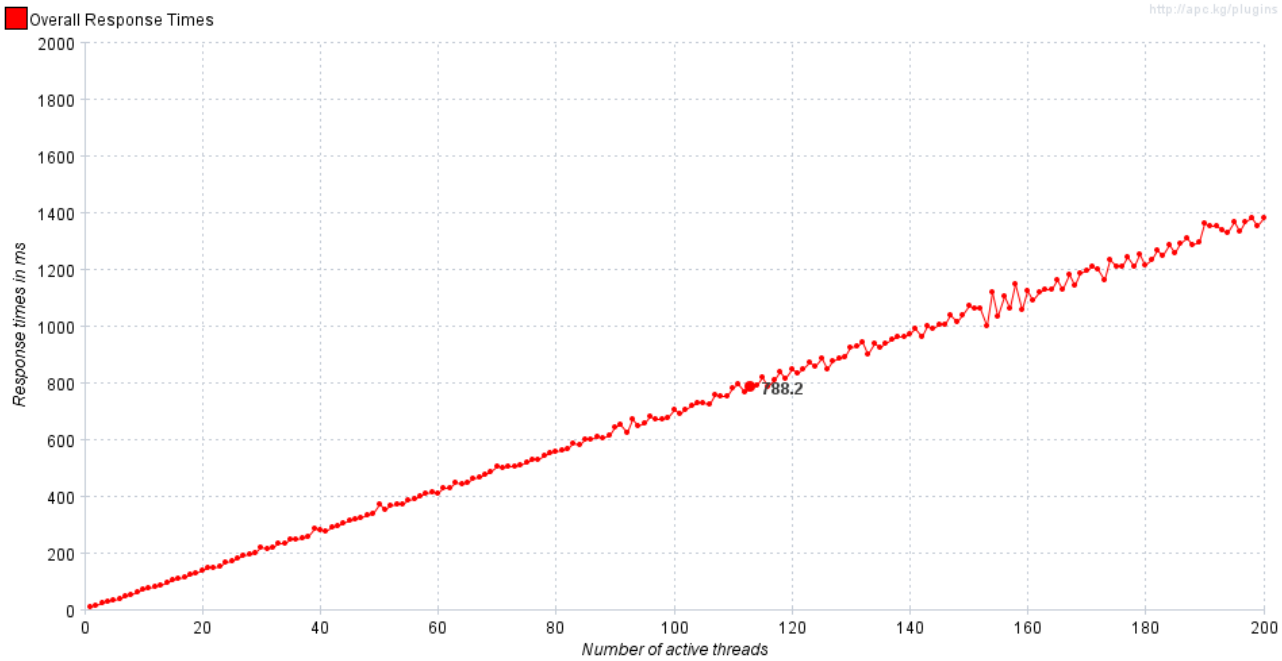


Рис. 0.15 Час відгуку NodeJS проти потоків

Час відгуку на активні потоки NodeJS показав майже ідеальну пряму лінію - лінійну залежність часу відгуку від кількості активних потоків.

Час відгуку змінюється зі зміною кількості паралельних потоків. Серверу потрібно більше часу, щоб відповісти, коли багато потоків підключення запитують це одночасно.

Отже, продуктивність системи прямо пропорційна прикладеному навантаженню, а це означає, що проектна парадигма керованої подіями моделі з AIO масштабується ідеально і придатна для роботи під великим навантаженням.

МКС продемонструвала подібну тенденцію показників ефективності (див. Рисунок 3.16), але з короткочасними сплесками сплесків часу відгуку та помилок. За цим графіком можна ідентифікувати піки часу відгуку при збільшенні навантаження. Допомагає виміряти ємність з точки зору кількості користувачів, які сервер може обслуговувати без особливого погіршення часу відгуку.

У певної кількості одночасних віртуальних користувачів під час перевірки продуктивності графік демонструє різку тенденцію до зростання, відому як коліно [16], у відповідь на всі транзакції. Це вказує на те, що досягнуто певного обмеження ємності в межах програми та почало впливати на час відгуку програми.

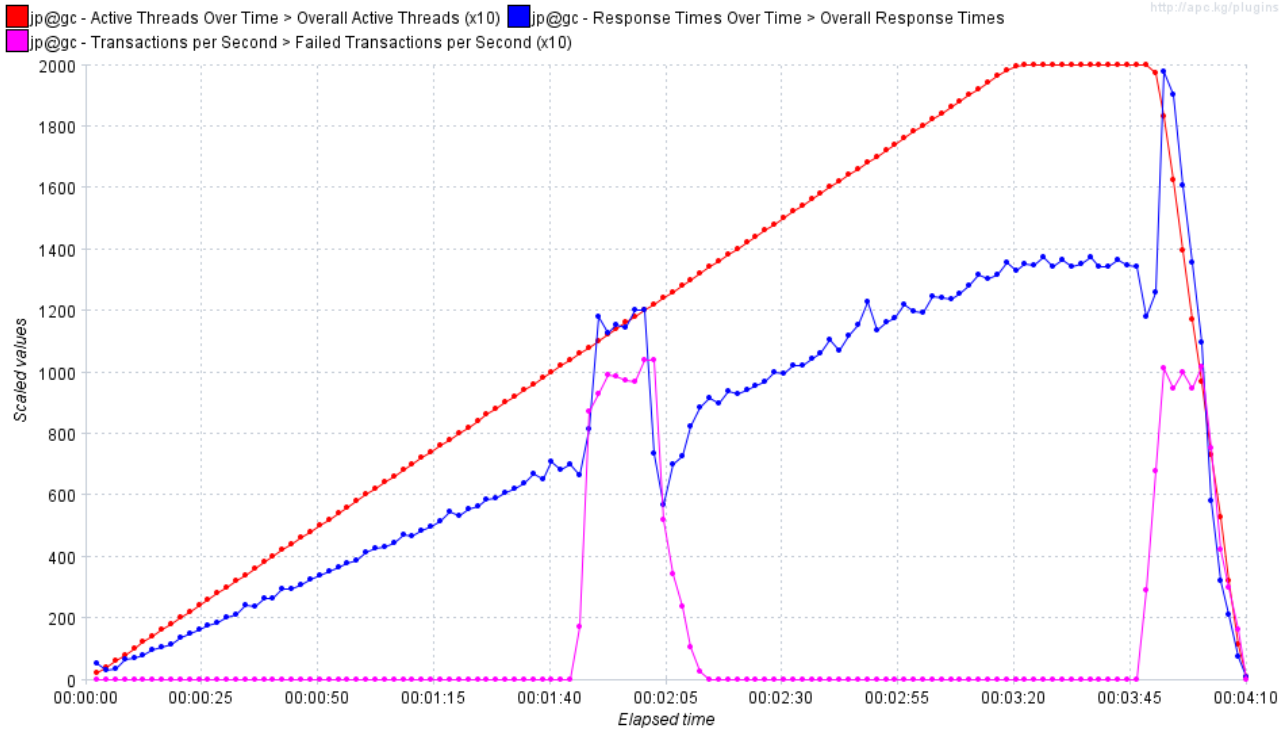


Рис. 0.16 Складений графік IIS (активні потоки, час відгуку та помилка)

Піки з помилками вказують на те, що сервер досяг своєї здатності обслуговувати дані і не може масштабуватись далі.

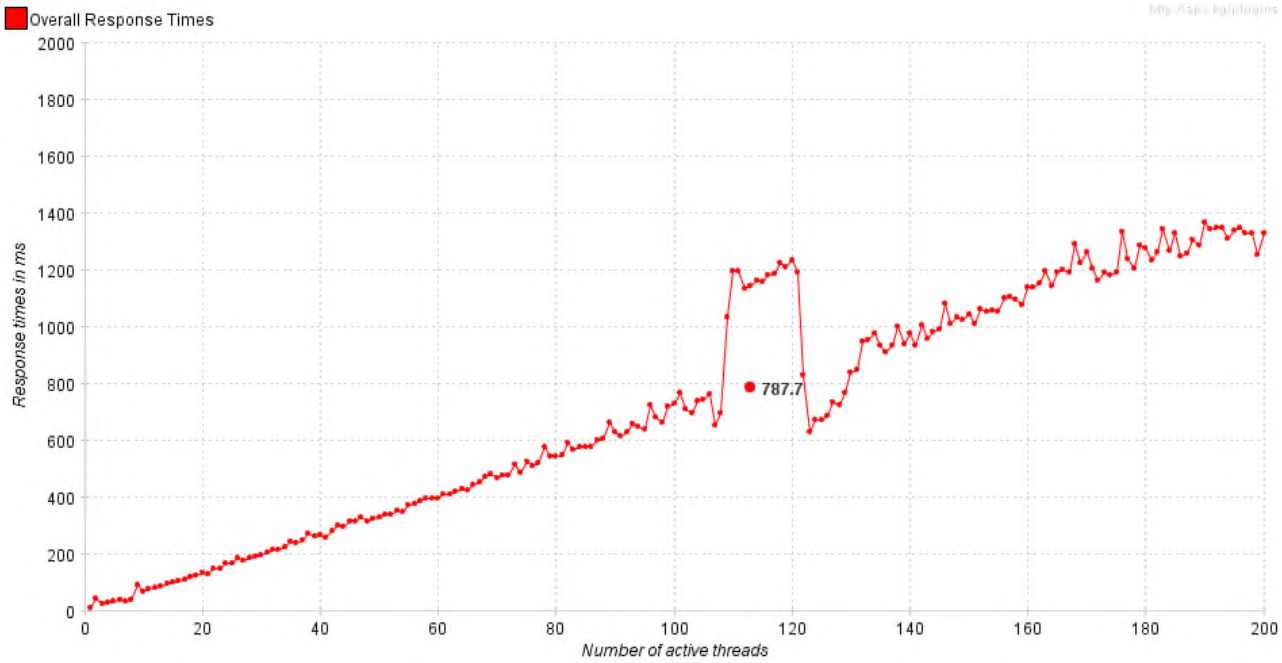


Рис. 0.17 Час відгуку IIS проти активних потоків

Графік пропускної здатності показує кількість активних одночасних HTTP-запитів, що надходять на веб-сервер протягом кожної секунди запуску. Цей графік допомагає оцінити кількість навантаження, яке створюють віртуальні користувачі, з точки зору кількості звернень.

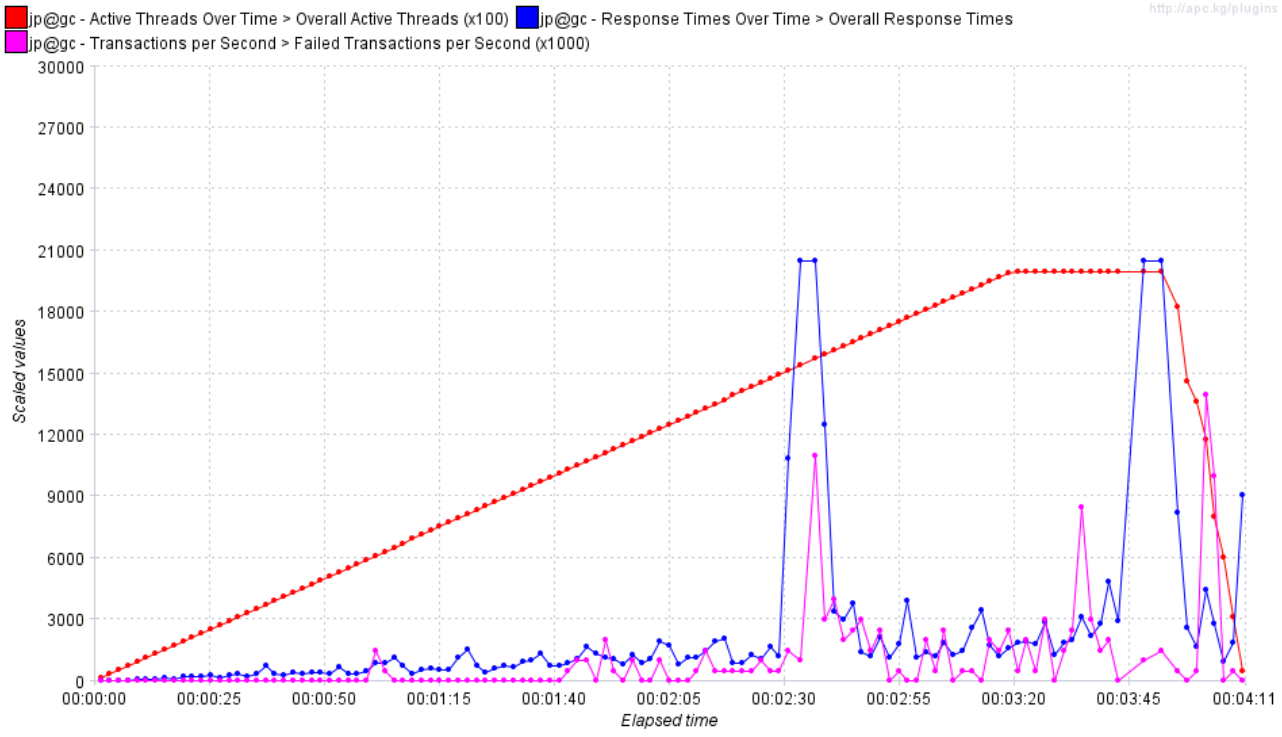


Рис. 0.18 Складений графік Apache (активні потоки та час відгуку)

Графік часу відгуку показує сплеск часу, який сервер прийняв для відповіді, коли навантаження збільшується.

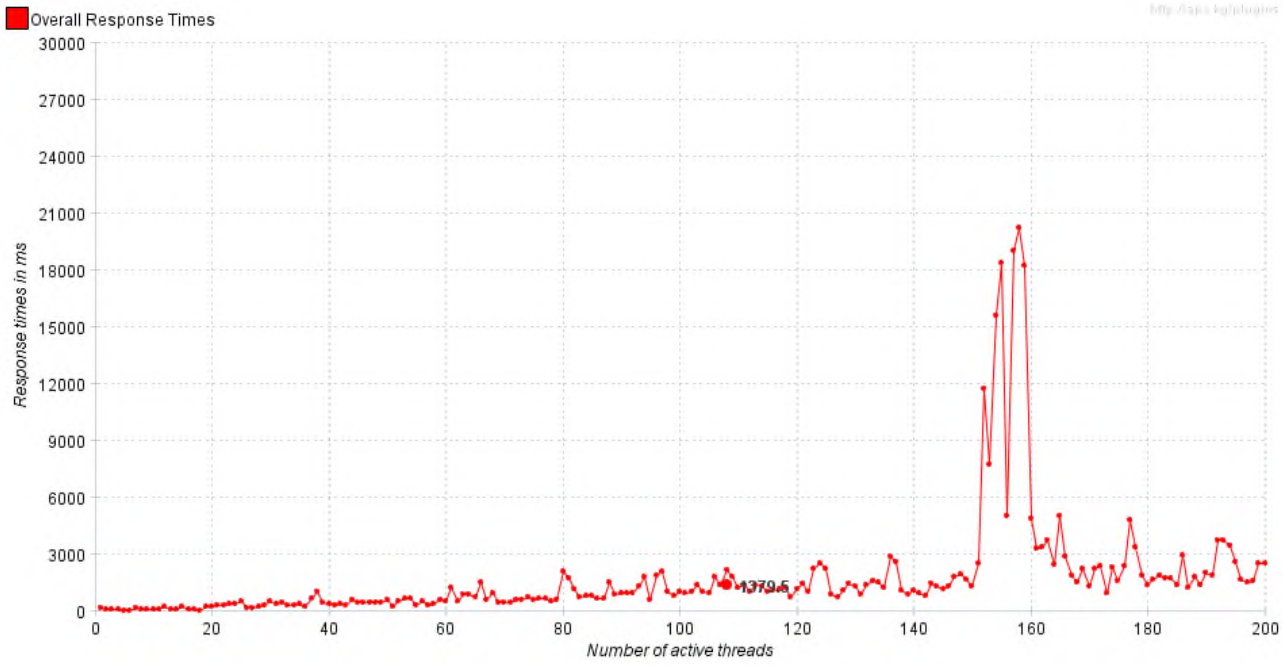


Рис. 0.19 Порушень Apache за секунду на графіку

Раптове зменшення пропускної здатності транзакцій вказує на проблеми і може збігатися з помилками, з якими стикається віртуальний користувач. Це відбувається, коли рівень веб-сервера досягає точки насичення для вхідних запитів.

Віртуальні користувачі починають зупинятися, чекаючи відповіді веб-серверів, що призводить до супутнього зниження пропускної здатності транзакцій. Врешті-решт користувачі почнуть тайм-аут і не зможуть; однак пізніше пропускна здатність знову стабілізується (хоча і на нижчому рівні), коли кількість активних користувачів перестає зростати.

Таблиця 3.2

Підсумок результатів тесту

	NodeJS	IIS	Апачі
Кількість зразків	35481	35543	20712
Середня відповідь, мс	788	787	1380
Медіана відповіді, мс	806	813	1830 рік
Рядок відгуку 90%, мс	1369	1344	4009

Мінімум відповіді, мс	7	8	9
Максимальна відповідь, мс	17	4041	20527
Відповідь std. розробник, мс	439,65	466	3405,6
Помилка,%	0	0,0947	0,0109
Швидкість проходження, хітів / сек	142,6	142,83	82,83
Пропускна здатність, КБ / с	9760,6	3376	2129,58

Час відгуку - одна з найважливіших характеристик навантажувального тестування. Звіти про час відгуку та графік вимірюють досвід роботи в Інтернеті, оскільки вказують, як довго користувач чекає, поки сервер відповість на його запит. Це час, необхідний у секундах, щоб отримати повну відповідь від сервера. Це еквівалентно часу, витраченому клієнтом на підключення до сервера та отримання відповіді, включаючи зображення, сценарій та таблицю стилів.

Хороша модель масштабованості та часу відгуку демонструє помірне, але прийнятне збільшення середнього (середнього) часу відгуку у міру збільшення навантаження віртуального користувача та пропускної здатності транзакцій. Погана модель демонструє зовсім іншу поведінку: із збільшенням навантаження віртуального користувача час відгуку збільшується в кроці, або або не згладжується, або починає стати нестабільним, демонструючи високі стандартні відхилення від середнього значення.

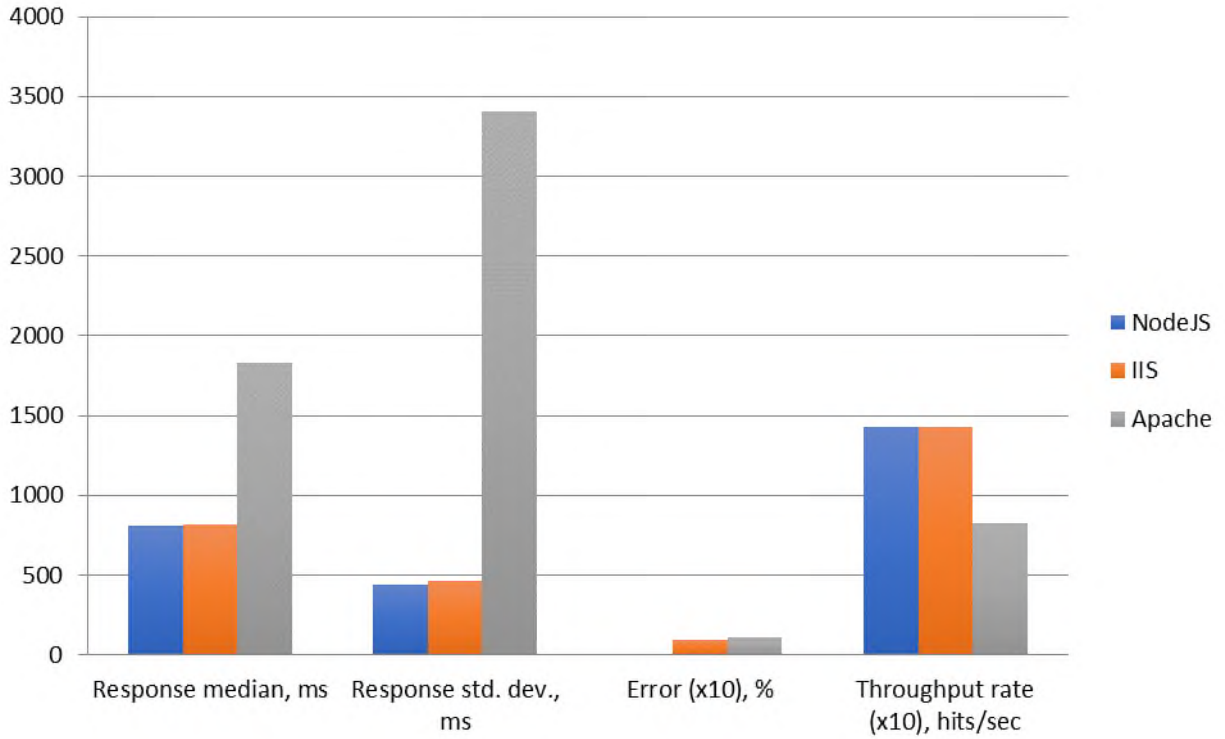


Рис. 0.20 Порівняльна гістограма отриманих результатів

При порівнянні часу відгуку, оскільки пропускна здатність зменшувалась, час відгуку також зменшувався. Зі збільшенням навантаження пропускна здатність починає нівелювання. Вирівнювання пропускної здатності вказує на те, що сервер досяг своєї здатності обслуговувати дані і не може масштабуватись далі.

NodeJS та ISS продемонстрували порівняно однаковий рівень продуктивності, але з різницею в стабільності обробки запитів та кількості повернених помилок.

Apache продемонстрував меншу швидкість пропускання та більший розподіл часу відгуку, порівняно з іншими. Стек черг з'єднань не працює ефективно для підтримки величезної кількості одночасних з'єднань; тому якась частина з'єднань не оброблялася. Таким чином можна зробити висновок, що Apache є найменш ефективною веб-платформою тестованих зразків.

Хоча деякі веб-платформи демонстрували нестабільність та сплески помилок, жоден з тестових зразків не зазнав аварії під високим тиском одночасних користувачів. Очікувані результати, оскільки їхня архітектура через великі накладні витрати заспокоює подальшу стабільну роботу після збоїв або помилок на вузлах.

Звідси висновок полягає в тому, що парадигма проектування асинхронної моделі, керованої подіями, з асинхронними неблокуючими масштабами вводу-виводу чудово масштабується і є ефективною для практичних потреб з великим навантаженням.

3.5. Порядок агрегації даних про вподобання з соціальних мереж

Агрегатор соціальних мереж базується на збиранні та аналізі відкритих у соціальних мережах даних користувачів та складається з трьох основних модулів:

– модуль збору та зберігання інформації;

– модуль аналізу зібраної інформації;

– модуль, що надає користувачу системи графічний інтерфейс.

На рис. 3.1 представлена загальна структура агрегатора соціальних мереж.

При розробці і проектуванні програми піднімаються питання, пов'язані зі змінами політик конфіденційності даних і поступової модифікації структури та функціоналу багатьох соціальних мереж.

Для розробки використовувався об'єктно-орієнтований підхід і структурна методологія побудови програмних систем.

В програмі використовувся фреймворк *Vue.js* з відкритим вихідним кодом для створення користувацьких інтерфейсів.

Легко інтегрується в проекти. Може функціонувати як веб-фреймворк для розробки односторінкових додатків в реактивному стилі. Використання фреймворку спрощує рощробку ПЗ та дозволяє розділити програму на модулі і розробляти її паралельно.

Сучасні фреймворки передбачають використання технології «товстого» клієнта, коли вимоги до комп’ютерів користувача більш високі.

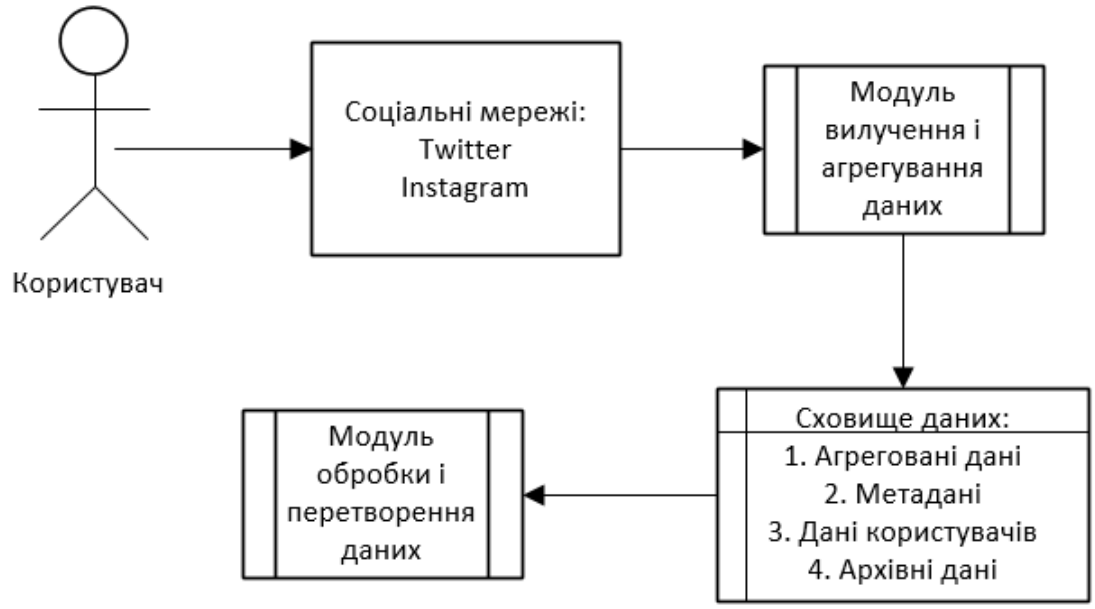


Рис. 3.1. Загальна структура агрегатора соціальних мереж для визначення вподобань користувачів

3.6. Висновок

NodeJS - це платформа з відкритим кодом, побудована на середовищі виконання движка V8 JavaScript, заснована на асинхронній керованій подіями неблокуючій моделі вводу-виводу, що робить її легкою та ефективною, ідеально підходить для програм, що працюють в режимі реального часу, що працюють під великим навантаженням. Він пов'язує передплату подій на системні дзвінки низького рівня, а потім переходить у режим очікування. ISS і Apache використовують потік для моделі підключення і проходять через міжпроцесний комунікаційний інтерфейс FasCGI, який показав нестабільність при перевантаженні. Конфігурація кожної тестової платформи була налаштована на придатність для передачі багатьох з'єднань.

Тестовий додаток був розроблений на двох мовах програмування для тестових цілей, але підтримуючи однакову функціональність на різних платформах. Логічний дизайн бази даних був зроблений для підтримки всіх

функцій середнього веб-додатку, але для зменшення його впливу на тестові характеристики він зберігався в пам'яті.

Тестування проводили на двох машинах: спостережуваній (де було створено 3 веб-платформи) та хост-системі (яка реєструвала результати та підтримувала навантаження). Навантаження поступово збільшувалось до 200 активних з'єднань в секунду і утримувалося протягом 30 секунд.

Результати показали схожу продуктивність ISS та NodeJS, але з різною стабільністю обробки; Результати Apache показали меншу обробну потужність та бризки помилок, - неефективну якість поведінки при перевантаженні.

Час відгуку NodeJS повністю відповідає застосованому навантаженню з нульовим коефіцієнтом помилок. Таким чином, конструктивна парадигма асинхронної моделі, керованої подіями, з асинхронним неблокуючим вхідним висновком чудово масштабується, і тому підходить для практичних потреб при великих навантаженнях.

ВИСНОВКИ

Розуміння сегментів споживачів є ключовим фактором будь-якого успішного бізнесу. Аналітично, сегментація передбачає кластеризацію набору даних для пошуку груп подібних клієнтів. Що означає "подібне", визначається даними, що потрапляють у кластеризацію - це можуть бути демографічні, позиційні чи інші характеристики. А дані, що надходять у кластеризацію, часто обмежуються самими алгоритмами кластеризації - більшість вимагає певної табличної структури даних, а загальноприйняті методи, такі як k-Means, вимагають строго числового введення.Порушення цих обмеженьбуло одним із наших пріоритетів з моменту створення компанії.

То що ви робите, коли хочете знайти сегменти клієнтів, які є «подібними», оскільки вони поводяться однаково - їхній досвід роботи з вами, їхнім брендом, був подібним. Як би ви це визначили? Все частіше компанії збирають дані про послідовність, причому кожен запис є взаємодією із клієнтом - будь то покупка, читання електронного листа, відвідування веб-сайту тощо. Враховуючи популярність методів глибокого навчання для вирішення завдань, пов’язаних із послідовністю, ми думали Застосування нейронних мереж до сегментації споживачів було природним підходом.

Цей пост будується нанаш попередній пост сегментації подорожі клієнтаі демонструє прототип підходу глибокого навчання до сегментації послідовності поведінки. Ми хотіли дослідити, чи зможемо ми використати внутрішній стан рекуррентної нейронної мережі (RNN) на складних послідовностях даних, щоб ідентифікувати відмінні сегменти клієнтів.

Ввідповідно до спроектованої архітектурою була написана клієнтська частина програми з використанням стандартних мов HTML5 / CSS, JavaScript, *PHP*. Були реалізовані рядок пошуку з кнопкою

«Пошук» для передачі користувальницького запиту, поле відображення тексту пісні з кнопкою «Показати / Приховати». Аудіо-програвач був реалізований з використанням технології MediaElement.js і бібліотеки jQuery. Аудіо-програвач підтримує необхідний функціонал управління, а саме: Play /

Pause, Skip, Mute / Unmute, регулювання гучності, відображення тимчасової смуги без можливості прокрутки.

Далі послідувала настройка взаємодії між сайтом і медіатекою для прослуховування музичних записів відповідно до визначення музичного аудіо-стрімінга, поставленими вимогами і спроектованої архітектурою.

Були написані методи на стороні клієнта для відправки запитів до сервера і сам сервер для обробки запитів клієнта і взаємодії з базою даних. Сервер отримує в якості вхідних даних рядок призначеного для користувача запиту або параметри поточної програється композиції в залежності від того, як був викликаний запит: пошуковий запит передається при натисканні на кнопку «Пошук»; параметри композиції передаються по закінченню поточного програється композиції або після натискання на кнопку «Skip».

При обробці пошукового запиту на сервері відбувається підключення до бази даних і пошук у ній схожих найменувань композицій, після чого повертається знайдена запис, або повідомлення про відсутність такої.

При обробці даних грає композиції враховуються тільки частотні параметри і id композиції в базі даних, щоб виключити її з пошуку, так як вона, найімовірніше, виявиться найбільш підходящою. Повернено буде запис про композиції, у якій буде найменша різниця з поточної. Обробка даних відбувається під час програвання

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ