МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Литвиненко О.Є.

«___» _____2021 р.

ДИПЛОМНИЙ ПРОЄКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
"БАКАЛАВР"

Тема: Портал-агрегатор для замовлення послуг з магнітно-резонансних обстежень

Виконавець: Ярмоленко Р.Ю.

Керівник: Артамонов Є.Б..

Нормоконтролер: Тупота Є.В.

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління
Спеціальність 123 "Комп'ютерна інженерія"
(шифр, найменування)

Освітньо професійна програма «Системне програмування»
Форма навчання заочна

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____Литвиненко О. Є.

«_____»_____2020 р.

ЗАВДАННЯ
на виконання дипломного проєкту

Ярмоленка Романа Юрійовича
(прізвище, ім'я, по батькові)

1.Тема роботи: "Портал-агрегатор для замовлення послуг з магнітно-резонансних обстежень "

затверджена наказом ректора від "21" грудня 2020 року № 2523 /ст.

2. Термін виконання роботи: з 11.01.2021 до 28.02.2021

3. Вихідні дані до роботи: 1) вимоги до змісту агрегатора;
2) основні функції агрегатора

4.Зміст пояснювальної записки (перелік питань, що підлягають розробці):
1) аналіз програмних систем-агрегаторів;
2) архітектура клієнт-серверної системи агрегації даних;
3) розробка, використання та тестування портала-агрегатора для замовлення послуг.

5.Перелік обов'язкового графічного матеріалу:
1) загальна структура портала-агрегегатора;
2) архітектура моделі сервера додатків;
3) інтерфейс портала-агрегатора для замовлення послуг з магнітно-резонансних обстежень;
4) схема алгоритму даних портала-ангрегатора.

6. Календарний план

№ п/п	Етапи виконання дипломного проєкту	Термін виконання етапів	Примітка
1	Провести аналіз літератури за темою дипломної роботи	11.01.21-14.01.21	
2	Провести аналіз існуючих систем для агрегації соціальних мереж та визначити їх основні функціональні можливості	15.01.21-17.01.21	
3	Підготувати перший розділ пояснювальної записки	18.01.21-21.01.21	
4	Розробити структуру клієнт-серверної системи агрегації даних	22.01.21-25.01.21	
5	Підготувати другий розділ пояснювальної записки	26.01.21-30.01.21	
6	Реалізувати програмні модулі системи агрегації даних, провести налаштування та тестування системи	31.01.21-07.02.21	
7	Підготувати третій розділ пояснювальної записки	08.02.21-10.02.21	
8	Оформити пояснювальну записку згідно діючих вимог до оформлення дипломних робіт випускників НАУ	10.02.21-17.02.21	
9	Підготувати презентацію та графічні матеріали	18.02.21-20.02.21	

7. Дата видачі завдання «_11_» січня 2021 р.

Керівник дипломного проєкту_____Артамонов Є.Б.
(підпис)

Завдання прийняв до виконання_____Ярмоленко Р.Ю.
(підпис студента)

РЕФЕРАТ

Пояснювальна записка до дипломного проєкту «Портал-агрегатор для замовлення послуг з магнітно-резонансних обстежень»: 54 с., 15 рис. 15 літературних джерел.

МЕДИЧНІ ОБСТЕЖЕННЯ, АГРЕГАТОР, *API,* КЛІЄНТ-СЕРВЕР

Мета дослідження – розробити програмний модуль портала-агрегатора для замовлення послуг з магнітно-резонансних обстежень.

Об’єкт дослідження – зв’язування *web*-додатку з сторонніми сервісами через *API*.

Предмет дослідження – портал-агрегатор для замовлення послуг з магнітно-резонансних обстежень.

Метод дослідження – алгоритмічна і програмна реалізація і проведення збору інформації з застосуванням агрегатора.

Встановлено, що розроблений програмний модуль дозволяє за короткий проміжок часу проаналізувати та отримати інформацію від сторонніх сервісів.

Результати дипломного проєкту рекомендується використовувати при консолідуванні розрзнених даних в одному місті.

ЗМІСТ

ВСТУП

Інформатика як наукова дисципліна пов’язана з багатьма аспектами розробки додатків: формальним описом мов програмування, пов’язаною з ними математичною теорією тощо.

Одне з таких питань - дослідження організаційних форм процесу розробки програмного забезпечення.

Пошук кращих шляхів для організації розробки програмного забезпечення був важливим питанням, оскільки розробка програмного забезпечення змінилася з чисто наукової дисципліни на більш практичну - інформатику, яка використовувалася в галузі.

Розробка програмного забезпечення як процес споживає велику кількість ресурсів. Такі ресурси не є матеріальними, вони, як правило, представлені людською працею та часом, але все одно вони в значній мірі пояснюються вартістю всього процесу розвитку.

Останнім часом, завдяки досягненням у розробці програмного забезпечення, зростає увага до ефективності процесу розробки програмного забезпечення.

Класичні методології програмного забезпечення все частіше критикують за неефективність. У компаніях, що розробляють програмне забезпечення, популярна тенденція застосовувати інші гнучкіші методології.

Зазвичай їх називають спритною розробкою програмного забезпечення.

Ще однією новою тенденцією в розробці програмного забезпечення є Lean - розробка програмного забезпечення.

Бережливе розроблення програмного забезпечення - це сукупність ідей та практик, що походять із так званої системи ощадливого виробництва. Головною метою цієї методології є також вдосконалення процесу розробки програмного забезпечення.

Отже, пошук кращих шляхів програмного забезпечення є важливим

питанням для розробників програмного забезпечення. З постійним зростанням розробки програмного забезпечення як економічної сфери; дедалі більше зростає увага до предмета.

Інша сучасна тенденція - зростаюче занепокоєння широкої громадськості щодо погіршення екологічної ситуації у світі. Погіршення екологічної ситуації є постійним питанням для дискусій.

Результатом такого зростаючого занепокоєння є поява нових організаційних концепцій.

Наприклад, була створена концепція сталого розвитку, яка використовується для вирішення таких питань.

Метою даної статті є поєднання цих двох тенденцій та використання їх для дослідження процесу розробки програмного забезпечення з використанням деяких екологічних концепцій.

Наприклад, концепція сталого розвитку може бути використана щодо процесу розробки програмного забезпечення.

Аналізується так званий екологічний підхід до розробки та використання програмного забезпечення.

Основна частина роботи присвячена розслідуванню різних втрат, які можуть виникнути в процесі розробки. Ця концепція також відома як концепція відходів, і це відносно нова тенденція в процесі розробки програмного забезпечення.

Концепцію відходів розробки програмного забезпечення створили інші автори, однак при описі відходів автори не надали жодної систематизації таких відходів.

Також мало проведено аналізу. Метою цієї роботи є також забезпечити розробника програмного забезпечення систематичним способом виявлення відходів та розподілити їх на офіційні категорії.

Тож насправді бракує наукових досліджень, які б вивчали процес розробки програмного забезпечення та оцінювали різні методології розробки з цих точок зору.

Однак є деякі наукові праці, що стосуються цієї теми опосередковано.

Ці роботи можуть бути використані для виконання поточного дослідження. Основним методом дослідження, що використовується тут, є емпіричне програмне дослідження.

Таке дослідження проводиться на основі інших існуючих наукових праць.

Ця робота може бути використана для подальшого аналізу проблеми відходів та екологічного підходу до програмної інженерії.

На виробничих майданчиках (у компаніях та командах розробників програмного забезпечення) цей документ може бути використаний для вдосконалення організації розробки програмного забезпечення, скорочення термінів поставки продукції та зменшення витрат.

Більш важливим є те, що певні частини цього дослідження можуть бути використані пізніше для проведення інших досліджень з цього питання.

Наприклад, деяким гнучким і худим IT-практикам не вистачає наукових доказів.

РОЗДІЛ 1

ПІДХІД ДО РОЗРОБКИ СИСТЕМ-АГРЕГАТОРІВ

Міжнародні організації, такі як ООН та Світовий банк, створили спеціальні структурні організації, відповідальні за застосування концепції сталого розвитку.

Концепція сталого розвитку складається з трьох аспектів: соціального, економічного та екологічного.

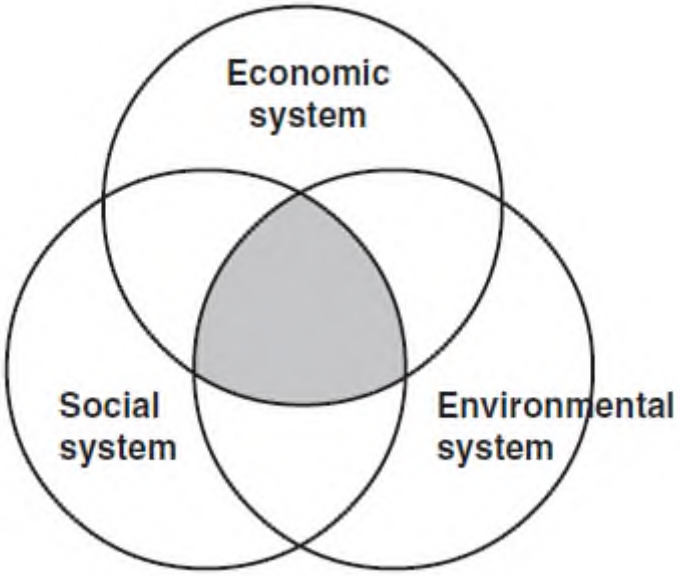


Рис. 1.1 Складові концепції сталого розвитку.

Система, яка є економічно стійкою, повинна мати можливість регулярно виробляти товари та надавати послуги. Він також повинен мати можливість підтримувати певні керовані рівні виробництва.

Іншим питанням є необхідність балансу між різними виробничими секторами, оскільки в деяких випадках надмірне зростання в одній сфері економіки, яке не доповнюється відповідним зростанням в інших сферах, може створити можливу загрозу для економічного добробуту в країні.

Стійке економічне зростання - це наявність позитивної динаміки макроекономічних показників (збільшення постійних та пропорційних темпів), а також відсутність значних коливань їх значень протягом тривалого періоду часу.

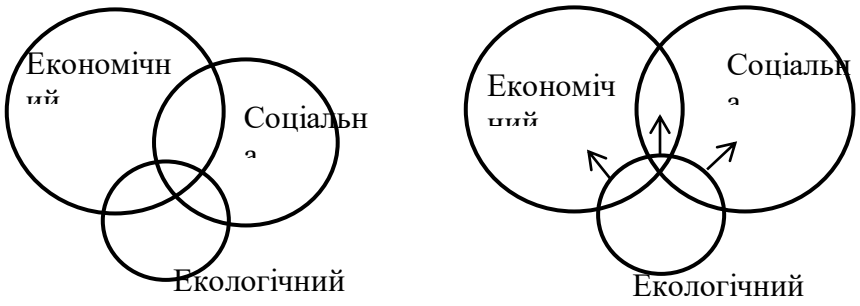


Рис. 1.2 Стермінова оцінка компоненти сталого розвитку та необхідні зміни. [2]

Ще до концепції сталого розвитку уряди переважної більшості країн мали спеціальні законодавчі органи, присвячені забезпеченню сталого розвитку в економіці. Ось чому ця складова сталого розвитку є найбільш розвиненою.[2]

Система вважається екологічно стійкою, коли вона може підтримувати стабільну ресурсну базу та забезпечувати цілісність природних біологічних та фізичних систем. Основна мета тут - зберегти здатність таких систем відновлюватися та динамічно адаптуватися до змін, замість того, щоб зберігати системи в якомусь «ідеальному» статичному стані. Деградація природних ресурсів, забруднення та втрата біорізноманіття зменшують здатність екосистем відновлюватися.

Метою соціально стійкої системи є створення та забезпечення системи рівномірного розподілу ресурсів, включаючи освіту, охорону здоров'я, політичні права тощо. У концепції сталого розвитку людина є не об'єктом, а скоріше суб'єктом розвитку. Розширення людських можливостей, будучи основною цінністю концепції сталого розвитку, передбачає, що люди повинні брати участь у процесах, що формують його середовище, таким чином просуваючи та реалізуючи політичні рішення, контролюючи їх виконання.

Сам економічний ріст не означає автоматично збільшення доходу для кожного сегмента населення. Для того, щоб відчути наслідки економічного

зростання на всі верстви населення, потрібно не лише збільшити виробництво, але й вжити певні політичні дії.

Взаємодія між компонентами сталого розвитку

Важливою особливістю концепції сталого розвитку є те, що вищезазначені аспекти (економічний, екологічний та соціальний) взаємопов’язані та збалансовані.

Ця концепція сталого розвитку відрізняється від так званого "зеленого розвитку" тим, що концепція зеленого розвитку надає пріоритет екологічній стійкості, а економічні та культурні фактори залишає на потім.

Таким чином, концепція стійкості може застосовуватися у таких випадках, коли більш "зелене" рішення є надто дорогим для поточного контексту.

Наприклад, «зелена» електростанція може зажадати занадто великих витрат, тому відкриття такої екостанції у країнах, що розвиваються, може не призвести до позитивних результатів. Через великі витрати електростанцію можуть змусити закрити, що пошкодить економічний аспект сталого розвитку в регіоні.

Дешевша електростанція, хоча і матиме більш згубний вплив на навколишнє середовище, матиме більше переваг для загального рейтингу сталого розвитку.

Співвідношення компонентів сталого розвитку може бути важко оцінити, щоб мати можливість здійснювати практичні дії, що ведуть до сталого розвитку.

Соціальний аспект має сильну позитивну кореляцію з економічним. У той же час протягом останніх багатьох років існує сильна перевернута кореляція між економічними та екологічними аспектами розвитку.

Екологічний аспект має сильний вплив на соціальний аспект. Наприклад, погіршення екосистем погіршує умови життя людей нижчого класу. Половина міського населення в Африці, Азії та Латинській Америці страждає від хвороб, спричинених відсутністю доступу до чистої води та санітарії[2]. Погіршення стану рибного сектору економіки в країнах, що розвиваються, позбавляє людей доступу до відносно недорогого джерела білка - споживання риби на душу населення в більшості цих країн зменшилось з 1985 по 1997 рік[2].

Складна взаємодія між різними компонентами сталого розвитку може унеможливити оцінку успіху проектів, спрямованих на впровадження концепції сталого розвитку в глобальному масштабі, оскільки вплив трьох компонентів один на одного дуже важко стверджувати.

Застосування концепції сталого розвитку для організацій

Незважаючи на те, що оригінальна концепція сталого розвитку була розроблена для опису глобальних світових процесів, її ідеї та концепції можуть бути використані в мікроекономіці. Тоді це може застосовуватися до окремих організацій.

Зі зростанням популярності концепції виникли такі терміни, як стале підприємництво. Це сталося на додаток до вже існуючих неологізмів, таких як соціальне підприємництво та екопідприємництво.

Сталий бізнес - це бізнес, який прагне уникнути негативного впливу на глобальну та місцеву екологічну ситуацію, суспільство та економіку.

Часто такі компанії мають більш інноваційну (порівняно з традиційним бізнесом у цій області) екологічну та кадрову політику.

Інше визначення стійкого бізнесу полягає в наступному. Сталий бізнес - це організація, яка використовує принципи сталого розвитку у всіх процесах, виробленій продукції чи наданих послугах, зберігаючи прибуток.

Сталий бізнес розглядає всі три аспекти сталого розвитку.

Постійний, стабільний і поступовий розвиток підприємства та отримання доходу від його діяльності становить економічну складову концепції.

Екологічна складова складається із зусиль, спрямованих на зменшення шкідливого впливу на навколишнє середовище під час виробництва товарів або надання послуг компанією. Це може включати: оптимізацію процесів зменшення споживання ресурсів, зменшення шкідливих викидів у навколишнє середовище, переробку відходів, що утворюються в процесі виробництва.

Екологічна складова також включає зусилля, спрямовані на те, щоб зробити готовий продукт компанії безпечнішим для навколишнього середовища.

Соціальна складова сталого розвитку включає піклування про вплив діяльності компанії на персонал підприємства та місцеве суспільство. Соціально

"стійкий" бізнес зазвичай бере участь у благодійних програмах та освітніх кампаніях. Наприклад, програми навчальних курсів для підприємств, які готують молодих спеціалістів на матеріальній базі компанії, не тільки покращують освітній рівень місцевої громади, але й створюють джерело персоналу для самої компанії.

Хоча оцінка глобальних проектів сталого розвитку стикається з багатьма проблемами, особливо існує проблема пошуку відповідних показників; окремі компанії можуть порівняно легко виміряти успіх своїх дій на місцевому, мікроекономічному рівні.

Компанії мають кілька доступних показників, таких як прибуток, вартість пошуку нових співробітників, оцінка репутації компанії споживачами та інші.

Ці показники можна описати чисельно, часто у грошовому еквіваленті, що дозволяє компаніям порівнювати ефекти дій, що включають різні аспекти сталого розвитку.

Що ще важливіше, такі показники створюють можливість оцінити позитивні та негативні ефекти різних практик сталого розвитку, що використовуються.

Переваги переходу на стійке підприємництво

Хоча застосування концепції сталого підприємництва має багато позитивних наслідків, ці ефекти є позитивними у світовому масштабі.

Крім того, позитивний вплив у соціальному та особливо екологічному аспектах проявляється лише через значний проміжок часу.

Оскільки концепція сталого підприємництва шукає баланс між соціальними, екологічними та економічними факторами, то в деяких випадках негайний чистий прибуток може бути зменшений після введення цієї концепції.

У той же час певні практики стійкого підприємництва можуть призвести до вищих прибутків. Також можна виявити, що деякі інші практики сталого підприємництва використовувались компанією, не розуміючи, що вони якимось чином пов'язані з концепцією сталого розвитку.

Наприклад, у багатьох компаніях є відкриті курси, які проводять навчання майбутніх працівників перед тим, як їх приймати на роботу. Хоча така діяльність

згадується в соціальному аспекті концепції соціального розвитку, компанія переслідує більш прагматичну мету - сприяти відбору персоналу компанії.

Таким чином, зміни в прибутку компанії під час переходу до сталого розвитку в значній мірі залежать від сектору економіки, в якій працює компанія та практики, що використовуються в компанії.

Використання «зеленої» технології та реалізація різних соціальних програм може вимагати для компанії додаткових витрат. Зниження заробітку в цьому випадку виглядає як конкурентний недолік - тоді як одна компанія, турбуючись про вплив на навколишнє середовище, витрачає гроші на вивезення сміття, інша компанія може витратити гроші на просування своєї продукції на ринок.

Це порушує питання, чи не заважає сталий розвиток бізнесу.

Однак ця точка зору помилкова.

Навіть якщо особисті фактори, такі як турбота керівництва про навколишнє середовище та суспільство, не беруться до уваги, компанії можуть мати інші причини застосовувати концепцію сталого розвитку.

Основним із цих факторів є репутація компанії - а саме ставлення людей до товарів та послуг компанії. Репутація компанії змінюється, якщо люди знають, що компанія дбає про навколишнє середовище, виробляючи продукцію та надаючи послуги.

Соціальна політика компанії також відіграє велику роль у ставленні людей до компанії.

Існує опитування Greendex, проведене в 2010 році National Geographic. Основні питання опитування спрямовані на з’ясування ставлення людей до екологічних проблем.

Згідно з цим опитуванням значна частина населення стурбована проблемами довкілля[3].

Результати опитування показують, що лише 13% респондентів повідомили, що їх не турбують екологічні проблеми.

7% респондентів сказали, що вважають екологічні проблеми головними проблемами в своїй країні, тоді як ще 16% вважають, що основними проблемами в їх країні є ті, які можна віднести до соціальної складової сталого розвитку.

Лише 21% респондентів вважають, що компанії працюють над тим, щоб зробити навколишнє середовище чистим.

55% усіх респондентів готові платити більше за товари, які більш енергоефективні протягом усього життя.

Це означає, що компанія, що використовує ідеї сталого розвитку, отримує конкурентну перевагу в очах покупця у вигляді покращення іміджу компанії та збільшення попиту на товари та послуги компанії.

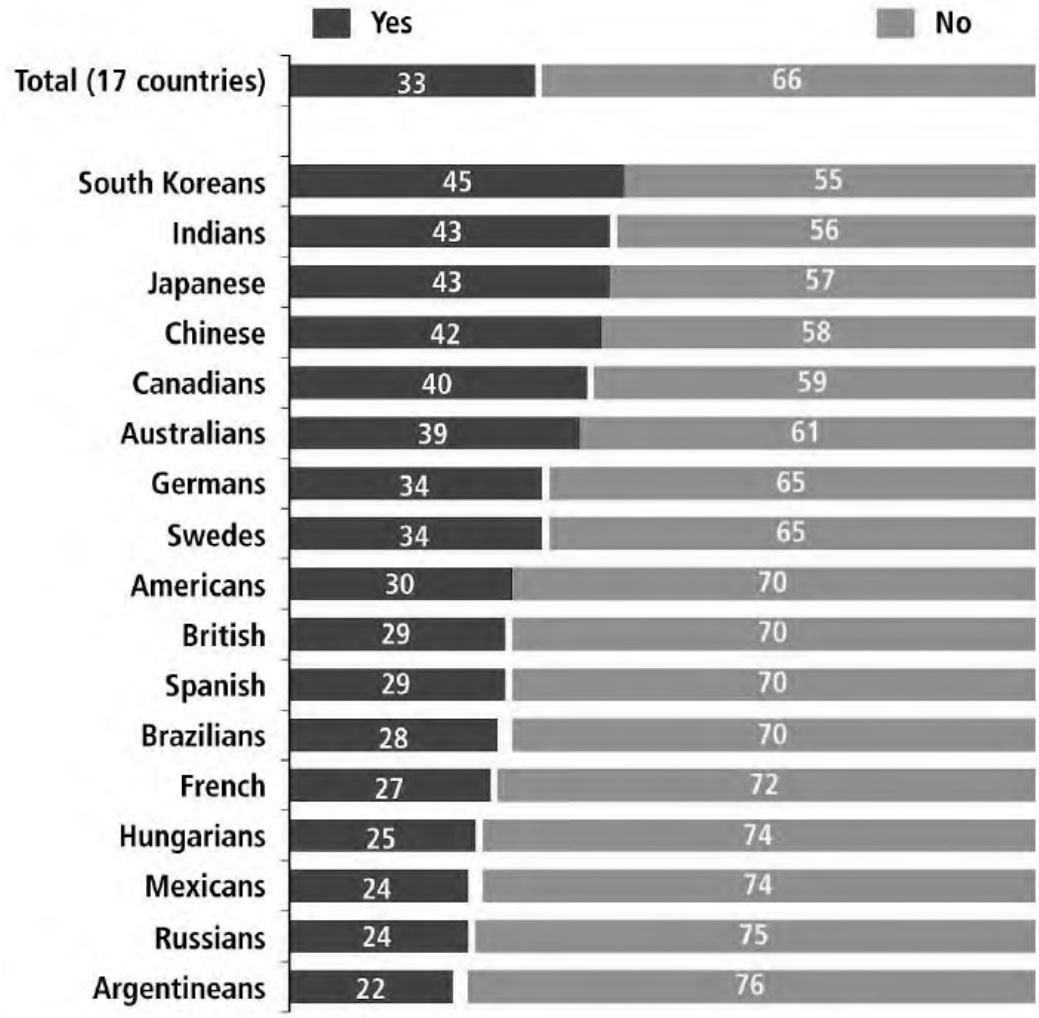


Рис. 1.3 Відсоток людей, що погоджуються із твердженням "Життя в моїй країні є екологічно стійким для майбутніх поколінь". [3].

Добре продумана соціальна та екологічна політика компанії може також мати позитивний вплив на ставлення працівників компанії до самого підприємства, що позитивно впливає на вартість управління людськими ресурсами.

Інформаційні технології

Інформаційні технології в найширшому розумінні - це технології, що використовуються для збору, обробки, зберігання, інтерпретації та використання інформації.

Широкий термін інформаційні технології включає: мови програмування, інформаційні системи, комп’ютерні мережі, бази даних та багато іншого.

Термін "інформаційні технології" та відповідний набір дисциплін утворився в результаті бурхливого розвитку комп'ютерних технологій наприкінці 20 століття, тому, коли згадується фраза "інформаційні технології", це, як правило, означає, що людина говорить про комп'ютер інформаційні технології.

Хоча комп’ютерні технології насправді є інформаційними, загальне визначення інформаційних технологій є ширшим і включає будь-яку технологію, що займається обробкою інформації, незалежно від реалізації такої обробки.

Таким чином біоінформатика та певні бізнес-процеси також можуть бути віднесені до інформаційних технологій.

Широке визначення інформаційних технологій охоплює як апаратні, так і програмні технології, що використовуються для обробки інформації. Ця сфера також включає офіційну діяльність щодо вивчення, організації та формалізації таких технологій.

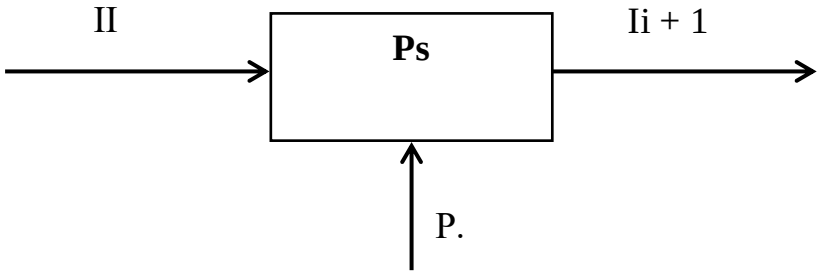
Інформаційні технології вимагають великих інвестицій та певного рівня економічного розвитку. Навчання IT-спеціалістів вимагає часу і вимагає великих витрат.

Більше того, таке навчання стикається з багатьма труднощами - а саме, надзвичайно високий рівень змін в IT-технологіях. Під час навчання IT-спеціаліста технології можуть застаріти. Все це ускладнює підготовку навчальних матеріалів та передачу знань від одного вихователя до тих, хто отримує освіту.

Інформаційні технології - індустрія, що швидко розвивається. У 2007 році, згідно з доповіддю Світового банку, 6,8% світового валового внутрішнього продукту було витрачено на інформаційні технології.[4]

Згідно з цим звітом, Сполучені Штати витратили 1 096 112 мільйонів доларів на інформаційні технології. Це становить середню витрату близько 3690 доларів на душу населення в США. [4]

У загальному вигляді інформаційні технології можуть бути представлені графічно наступним чином:



Фігура 1.4. Графічне зображення інформаційної технології.

Фігура 1.4 містить такі елементи:

Ps - процес використання інформації або обробки інформації.

Це становить фактично найважливішу частину інформаційних технологій.

Ii - вхідна інформація, тобто інформація, яка використовується або обробляється в процесі Ps.

Ii + 1 - вихідна інформація, тобто інформація, яка виникла в результаті роботи процесу Ps. Важливо зауважити, що вихідна інформація, отримана внаслідок діяльності однієї інформаційної технології, може стати вхідною інформацією іншої.

R - ресурси, які використовуються під час виконання процесу Ps. R може стосуватися широкого типу ресурсів, таких як: матеріальні ресурси, людські ресурси, часові ресурси тощо.

Розробка програмного забезпечення

Розробка програмного забезпечення - це систематичний процес проектування, виробництва, обслуговування та утилізації програмних продуктів.

У свою чергу, розробка програмного забезпечення також може розглядатися як інформаційна технологія, яка також може бути виражена якФігура 1.4, де:

Процес Ps представляє фактичний процес програмного забезпечення, що розробляється.

Вимоги до програмного забезпечення є звичайним прикладом вхідної інформації Ii.

R - ресурси, які використовуються під час виробництва програмного забезпечення. Зазвичай це означає використовувані людські ресурси. Але він може також представляти спеціальні програмні продукти та послуги, які використовуються під час розробки.

Результатом всього процесу (Ii + 1) є програмний продукт.

Залежно від організації процесу розробки програмного забезпечення інформація Ii може бути представлена різними структурами.

Для компанії, яка надає розробку програмного забезпечення як послугу, Ii може бути представлена як набір формальних вимог до програмного забезпечення, що розробляється.

Якщо організація виконує використання старого програмного забезпечення та використовує його для створення нового продукту, то старий програмний продукт також може виступати в якості вхідної інформації Ii.

Якщо компанія розробляє програмний продукт за власною ініціативою, виходячи з очікувань, що створюваний програмний продукт принесе прибуток компанії, Ii може бути представлений не набором чітко визначених вимог, а скоріше як емпірична концепція, заснована на оцінки очікувань керівників ринку та компаній.

У цьому випадку розробка більш формалізованих вимог до програмного забезпечення є завданням самого процесу розробки Ps. Залежно від процесу розвитку організації такі вимоги можуть залишатися постійними або зазнавати частих змін під час розвитку.

Життєвий цикл програмного забезпечення

Хоча розробка програмного забезпечення сама по собі передбачає власне процес програмування, тобто написання вихідного коду продукту; цей крок - це не тільки діяльність, що виконується в процесі розробки програмного забезпечення.

Опис етапів існування програмного продукту від початку розробки до кінця служби називається життєвим циклом програмного забезпечення. Життєвий цикл описується різними моделями, кожна з яких визначає свій власний набір етапів життєвого циклу. Тим не менше, певні етапи життєвого циклу є загальними і повторюються в різних моделях:

Створення вимог до програмного забезпечення для майбутнього програмного продукту

Розробка програмного продукту,

Впровадження або виробництво

Тестування або перевірка

Обслуговування програмного продукту

Використання програмного продукту

В основі процесу розробки програмного забезпечення лежить процес програмування, але питання, який із етапів є найбільш важливим, залишається відкритим.

Деякі дослідники вважають, що етапи визначення вимог і проектування важливіші за процес написання коду.

Для ілюстрації цієї аналогії використовується виробництво фільму [5]. Перед початком зйомки пишеться сценарій фільму. Цей сценарій служить планом майбутніх дій. Стадія проектування програмного продукту виконує те саме завдання.

Після цього починається дорогий і трудомісткий процес зйомки, під час якого будь-які зміни сценарію призводять до значного збільшення часових та грошових витрат.

У класичних моделях життєвого циклу стадії проектування та програмування трактуються однаково.

У той же час такий етап, як обслуговування програмного продукту (вирішення різних питань, не виявлених на етапі виробництва та тестування), може, за різними оцінками, становити від 40% до 90% вартості програмного продукту [6] [7].

Етап використання програмного забезпечення відсутній у деяких моделях життєвого циклу, але він також важливий. Про важливість етапу може свідчити такий факт - у 2001 році майже 50% розробників програмного забезпечення були зайняті модифікацією існуючого програмного забезпечення, а не написанням нового з нуля[6].

Класичні методології розробки програмного забезпечення

Найвідомішою та найпоширенішою моделлю моделі розробки програмного забезпечення є "водоспад" або послідовна модель.

Основна ідея, на якій лежить модель водоспаду, полягає в послідовному переході від однієї стадії розвитку до іншої після завершення попередньої.

Базова модель водоспаду передбачає перехід від однієї фази розвитку до іншої лише після завершення попередньої, але існують також так звані модифіковані моделі водоспаду з деяким ступенем зворотного зв’язку між стадіями та припущенням перекриття етапів.

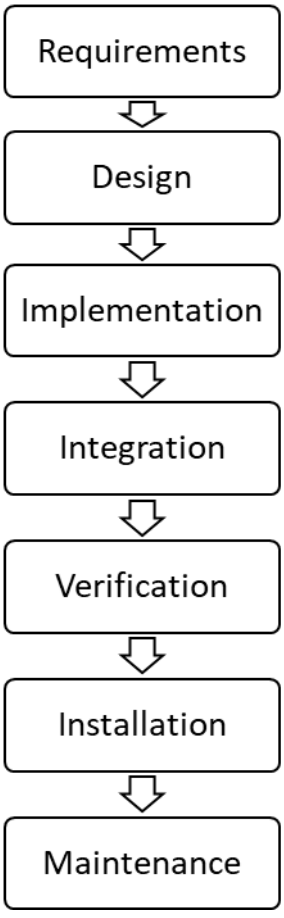


Рис. 1.5 Схема класичної моделі водоспаду.

Додатковими етапами, представленими в цій моделі, про які не йшлося вище, є:

Інтеграція - інтеграція окремих компонентів, розроблених різними групами, в єдиний програмний продукт.

Верифікація - перевірка відповідності готового програмного продукту початковим специфікаціям. Специфікація товару відіграє важливу роль у моделі водоспаду.

Встановлення - прийняття програмного продукту на клієнтському сайті.

Розробка програмного забезпечення як дисципліни на початку свого існування була нерозривно пов'язана з розвитком великих апаратних систем. Програмні продукти протягом тривалого часу не розглядалися як окремий продукт, а як невід'ємна частина комп'ютерної системи.

Таким чином, розробка програмного забезпечення була пов’язана з процесом проектування та виробництва апаратних засобів і часто організовувалася подібним чином.

Більше того, на зорі свого існування розробка програмного забезпечення як діяльність переважно переважала державні установи та великі корпорації.

Ці умови були першопричиною появи моделі водоспаду життєвого циклу програмного забезпечення.

По-перше, у моделі водоспаду вимоги до виробу, що розробляється, формуються до початку розробки. Тісний зв’язок між програмними та апаратними системами на той час ускладнював або навіть неможлив вносити будь-які зміни до специфікації продукту після початку процесу розробки програмного забезпечення.

На розробку програмного забезпечення як дисципліни суттєво впливали традиційні інженерні дисципліни. Зміни після початку виробництва в таких дисциплінах дуже дорогі або навіть неможливі.

По-друге, модель водоспаду дозволяє досягти більшої формалізації процесу розвитку. Етапи розвитку чітко позначені і відокремлені один від одного; розвиток на сцені чітко визначається інформацією, що надходить з попередньої стадії.

Точний процес формалізації є життєво важливим для державних організацій, і модель водоспаду часто зустрічається в офіційних стандартах

державних установ. Наприклад, стандарт Міністерства оборони США DOD-STD-2167A описує використання моделі водоспаду для розробки програмного забезпечення.

Існують і інші класичні моделі розробки програмного забезпечення, такі як V-модель, але вони були менш поширеними, ніж модель водоспаду та її модифікації; і приділяли менше уваги розробників.

Зелений IT

Поки сектор інформаційних технологій розширювався, спостерігалося значне збільшення кількості комп'ютерів та пов'язаних з ними пристроїв. Що ще важливіше з'явилися нові сфери використання таких пристроїв.

В результаті такого бурхливого розвитку зараз комп’ютери відповідають за споживання 2,61% світової електроенергії та 1% усіх викидів CO2 у світі. [8]

Виробництво комп’ютерної техніки - складний і ресурсомісткий процес.

Хімічні речовини, які становлять загрозу для навколишнього середовища, можуть використовуватися для виробництва комп’ютерного обладнання. Зокрема, ЕЛТ-монітори містять велику кількість свинцю, а РК-монітори - ртуті. Це створює проблему належної утилізації старого комп’ютерного обладнання.

У той же час, зростаюча стурбованість щодо погіршення екологічної ситуації, розуміння того, що технології повинні бути екологічно чистими, що виробляється таким чином з метою мінімізації майбутнього впливу на екологічне середовище, проникло в такі високотехнологічні галузі, як IT.

Результатом таких процесів є поява концепції зеленого IT.

Зелені інформаційні технології - це інформаційні технології, які розробляються, виготовляються, використовуються та утилізуються з мінімальним впливом на навколишнє середовище. [9]

У практичному застосуванні зелені IT найчастіше зосереджуються на виробництві та використанні комп’ютерного обладнання таким чином, щоб мінімізувати несприятливий вплив на навколишнє середовище.

Оскільки саме визначення інформаційних технологій є досить широким, термін "зелені" IT також включає багато аспектів.

Інформаційні технології часто базуються на складному поєднанні людей, апаратного та програмного забезпечення; Зелена інформаційна технологія також акцентує увагу на екологічній ефективності кожного з компонентів технології.

Приклади зеленого:

Оптимізація енергоспоживання серверів на серверних платформах.

Віртуалізація комп'ютерних ресурсів, таких як сервери та мейнфрейми, для кращого використання та розподілу ресурсів.

Розробка спеціального програмного забезпечення для енергозбереження, наприклад, програм для ПК, які контролюють споживання енергії в офісі.

Розробка енергоефективних алгоритмів для пристроїв. Сюди також входить енергоефективне вбудоване програмне забезпечення.

Заміна бізнес-процесів комп'ютерними технологіями (наприклад, використання електронних систем ведення діловодства для підприємства та використання відеоконференцій для зменшення обсягу ділових поїздок).

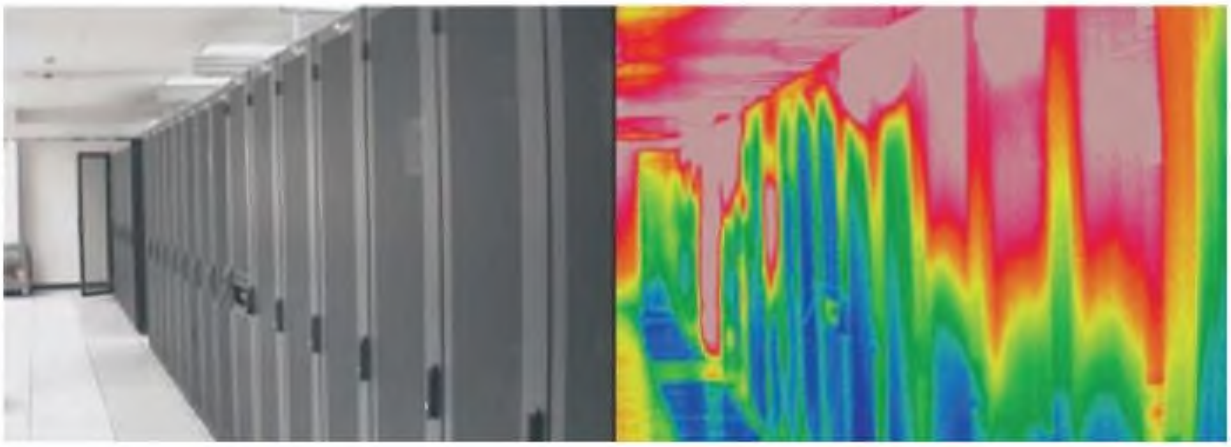


Рис. 1.6 Теплові карти використовуються для проектування серверних приміщень

Протиставляючи ідеї концепції сталого розвитку, концепція зелених IT надає перевагу екологічним аспектам технологій перед усім іншим.

Причини прийняття зеленого IT такі самі, як і причини прийняття сталого розвитку. Дуже часто використання енергоефективних рішень та оптимізація процесів з точки зору енергозбереження призводить до скорочення витрат.

Автоматизоване управління живленням комп'ютерів в офісах може слугувати прикладом ситуації, коли контроль споживання енергії обладнанням може безпосередньо заощадити гроші.

Такі екологічні рішення представлені в декількох програмних продуктах і, в свою чергу, створюють власний невеликий ринок екологічних продуктів.

Процес розробки програмного забезпечення також може бути предметом оптимізації з точки зору зелених ІТ.

Загальні правила в цьому випадку такі самі, як оптимізація звичайної офісної діяльності: безпаперове виробництво, енергоефективні технології, що використовуються в пристроях, оптимізація бізнес-процесів, щоб уникнути зайвої діяльності, як подорожі.

Робота з програмістами на віддаленій основі також є звичайною практикою в компаніях, які займаються розробкою програмного забезпечення. Окрім економії часу та ресурсів, що споживаються щодня, коли працівник подорожує містом, використання віддаленої роботи полегшує збільшення персоналу, дає можливість наймати фахівців з інших міст та країн, знижує витрати на обслуговування офісу та багато іншого.

Іншим важливим питанням є те, що розробка програмного забезпечення як інформаційної технології створила інші інформаційні системи. Програмні продукти, розроблені однією компанією, згодом можуть бути використані на мільйонах комп'ютерів та пристроїв у всьому світі.

Тож від рішень, прийнятих у розробницькій компанії, залежить вплив такого програмного забезпечення на навколишнє середовище. Критерій екологічності в цьому випадку також повинен дотримуватися в кінцевому продукті компанії, а не лише в процесах розробки програмного продукту.

Частіше "зелений" продукт є важливішим, ніж "зелений" процес розробки такого продукту.

Сучасні тенденції розвитку програмного забезпечення

З розвитком комп’ютерної індустрії відбувся поступовий перехід від загальновизнаного розуміння програмного забезпечення як компонента великої комп’ютерної системи до трактування програмного забезпечення як самостійного продукту.

У міру розвитку комп’ютерної індустрії постійно зростає кількість використовуваних комп’ютерів та зростає кількість «розумних» пристроїв, таких як смартфони та планшетні ПК.

Старі методології розробки програмного забезпечення все частіше піддаються критиці.

Поява та популяризація Інтернету забезпечили промисловість технічними можливостями щоденних оновлень програмного забезпечення. Збільшується кількість програм, що використовуються в Інтернеті, які взагалі не потребують інсталяції.

Ринкова конкуренція між розробниками програмного забезпечення, що призвела до ситуації, коли час випуску та час оновлення відіграють важливу роль у успіху продажів програмних продуктів.

Таким чином, програмний продукт перетворений із продукту з тривалим терміном розробки та чітко визначеними вимогами у послугу з постійно мінливими вимогами та необхідністю швидкого виходу на ринок.

Класичні моделі розробки, такі як модель водоспаду, зазнали критики з боку різних експертів у галузі розробки програмного забезпечення.

У своїй книзі "Lean Software Development" Мері та Том Поппендік із іронією говорять про заздалегідь встановлені специфікації програмних продуктів:[7]

«Шановний клієнт, будь ласка, будь ласка, надайте нам список того, що програмне забезпечення має робити. І, будь ласка, підпишіться нижче. Після цього, якщо вам потрібно щось змінити або додати, вам доведеться пройти складний процес, який називається «управління змінами», щоб отримати схвалення для кожної зміни. Тож якщо ви хочете хорошого програмного забезпечення, спробуйте передбачити все заздалегідь, тому що ми повинні знати про все це перед початком процесу розробки.

Не дивно, що наші клієнти включають до цього списку все можливе»

Том і Мері Поппендік є авторами та популяризаторами нової моделі розробки програмного забезпечення, відомої як "бережлива розробка".

Модель базується на так званій бережливій виробничій системі, спочатку розробленій і використовуваній у виробництві автомобілів Toyota. Автори адаптували основні принципи системи ощадливого виробництва для використання в області розробки програмного забезпечення.

У своїй книзі вони описують основні принципи та підходи до ощадливого розвитку програмного забезпечення з метою підвищення ефективності процесу розробки програмного забезпечення. Окрім технологій Toyota, ощадливий розвиток програмного забезпечення спрямований на зменшення втрат у виробництві, вдосконалення команд розробників та більш відкриті відносини з клієнтами.

Том і Мері Поппендік з їх концепцією ощадливого розвитку програмного забезпечення є частиною так званої гнучкої спільноти, спільноти розробників програмного забезпечення, що практикують нові "спритні" підходи до розробки програмного забезпечення.

Швидкі практики розробки програмного забезпечення

Швидка розробка програмного забезпечення - це не єдина модель розробки, а скоріше сукупність загальних принципів та ідей.

Існує багато конкретних і практичних спритних моделей, і кожна з них має свої власні практики та підходи до розвитку. Але всі вони дотримуються цих загальних принципів.

Agile Software було сформовано як альтернативу такій формалізованій моделі розвитку, як модель водоспаду.[10]

Основною ідеєю гнучких моделей розробки є перехід від заздалегідь запланованого послідовного процесу розробки програмного забезпечення до менш формалізованого процесу, що заохочує адаптивні зміни в планах розвитку під час фактичного розвитку.

Спритні практики також вважають, що добре надати командам розробників більше свободи і дозволити їм самостійно організовувати та вдосконалювати процес розвитку. Клієнтам дозволяється вносити зміни в технічні характеристики продукту під час розробки, і команда розробників повинна мати можливість швидко та ефективно реагувати на такі зміни.

Згідно з книгою "Мистецтво ощадливої розробки програмного забезпечення", всі гнучкі методології використовують короткі ітерації з обмеженим часом, причому кожна така ітерація призводить до функціонального програмного продукту. Тривалість ітерацій варіюється залежно від методології.[10]

Термін "спритний розвиток" є відносно новим, він утворився в 2001 році в Маніфесті Agile: [11]

Ми розкриваємо кращі способи розробки програмного забезпечення, роблячи це та допомагаючи іншим робити це. Завдяки цій роботі ми прийшли до оцінки:

Особи та взаємодія над процесами та інструментами

Працююче програмне забезпечення над вичерпною документацією

Співпраця замовника під час переговорів про контракт

Відповідаючи на зміни, дотримуючись плану

Тобто, хоча в елементах праворуч є цінність, ми більше цінуємо елементи ліворуч.

Маніфест підписали представники різних методологій (екстремальне програмування, Scrum, DSDM, адаптивна розробка програмного забезпечення, Crystal Clear, розроблена функціями, прагматичне програмування). Ці методології в подальшому називатимуться «гнучкими».

Рис. 1.7 Деякі спритні компанії надають платні послуги з сертифікації.

Швидкі моделі розробки програмного забезпечення набирають популярності. Згідно з опитуванням "Стан гнучкого розвитку", станом на 2010 рік 90% респондентів працювали в організаціях, які якимось чином використовували гнучкі практики розвитку.[12]

Іноді програмна компанія поєднує у своїй роботі практики різних методологій. 5% опитаних компаній використовували власні версії гнучких методологій.

У цьому ж опитуванні популярність різних спритних практик виглядала наступним чином:

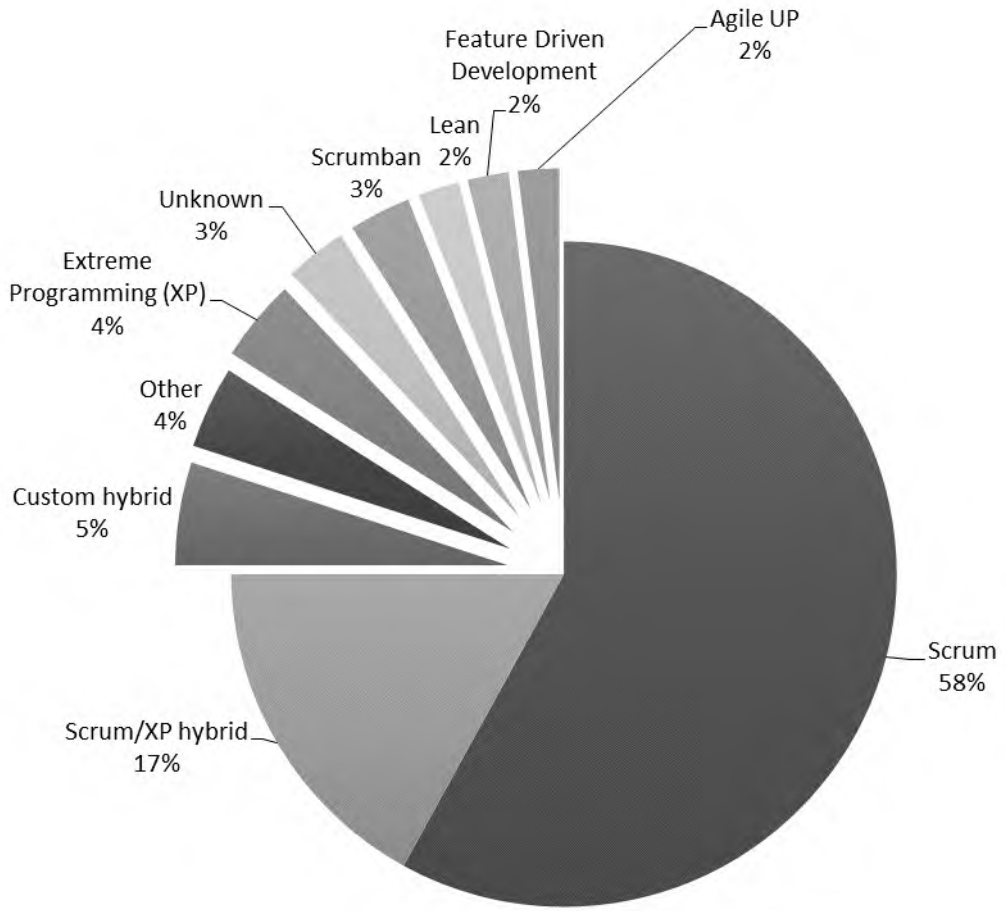


Рис. 1.8. Популярність різних гнучких методологій розробки програмного забезпечення.[12]

Кожна з методологій має власний набір практик розробки програмного забезпечення; багато з них мають власні книги, що описують ці практики; у деяких є шанувальники і навіть власні офіційні програми та організації сертифікації.

Прихильники гнучких методологій вважають, що більша гнучкість у процесі розробки програмного забезпечення зменшує відходи, пов'язані з процесом розробки, а більша свобода, надана командам розробників, покращує якість програмного забезпечення.

Крім того, завдяки більш швидкій реакції на зміни, спритні практики, як вважають, зменшують інвестиційний ризик для нових програмних продуктів.

Висновки

Зростаюча увага до погіршення глобальних екологічних та соціальних проблем призвела до появи нової глобальної концепції людського розвитку.

Така концепція була представлена комісією ООН і отримала назву: концепція сталого розвитку.

Концепція зосереджена на такій організації внутрішньої політики, де потреби поточного покоління задовольняються, не ставлячи під загрозу здатність майбутніх поколінь задовольняти свої потреби.

Концепція зосереджена на трьох аспектах: соціальному, економічному та екологічному. Таким чином, розвиток є стійким, коли він отримав стійкість за всіма трьома компонентами. Ці три концепції повинні бути збалансовані між собою. Це відрізняє цю концепцію від інших "зелених" концепцій, які часто зосереджуються лише на екологічних проблемах.

Створена спочатку для прийняття рішень у макромасштабі концепція сталого розвитку знайшла своє застосування в організації та функціонуванні підприємств.

Отже, концепція сталого розвитку може бути застосована до управління командою розробників програмного забезпечення.

Розробка програмного забезпечення, будучи інформаційною технологією, підпорядковується певним схемам організації розробки програмного забезпечення. Життєвий цикл програмного забезпечення визначає етапи, через які програмне забезпечення проходить протягом свого існування.

Методологія розробки програмного забезпечення визначає, як пов'язані ці кроки та як повинен бути організований процес розробки в рамках цих етапів.

Класичною і, мабуть, найвідомішою методологією розвитку є модель водоспаду - модель розвитку з поступовим переходом від однієї стадії до іншої в міру завершення однієї стадії.

Модель водоспаду розкритикували представники так званої спритної спільноти. Модель водоспаду критикують за занадто формалізовану форму, бюрократичність та відсутність гнучкості.

Для вирішення цих проблем пропонуються інші методології, такі як Scrum, екстремальне програмування тощо. Серед гнучких методологій існує також концепція "Бережливого розвитку програмного забезпечення", яка є спробою перенести концепцію "бережливого виробництва" Toyota у процес розробки програмного забезпечення.

Кінцевою метою "ощадливої розробки програмного забезпечення" є підвищення ефективності процесу розробки програмного забезпечення за рахунок зменшення втрат, мінімізації ризиків, поліпшення обміну інформацією в команді тощо.

Агрегатор для замовлення послуг з магнітно-резонансних обстежень – це програмний продукт або сервіс, який збирає інформацію з різних порталів, блогів та інших ресурсів в одне джерело. Також важливо те, що на разі одним з головних трендів розвитку Інтернету є персоналізація – користувачам пропонується тільки та інформація, що відповідає їх інтересам.

Розглянувши готові рішення агрегаторів, було з’ясовано, що кожен сервіс має різне покриття інформаційного поля і набір специфічних функцій.

РОЗДІЛ 2

АРХІТЕКТУРА КЛІЄНТ-СЕРВЕРНОЇ СИСТЕМИ АГРЕГАЦІЇ ДАНИХ ДЛЯ ІНТЕРНЕТ ПОРТАЛУ

АНАЛІЗ СУЧАСНИХ ВЕБ-ТЕХНОЛОГІЙ І ПРОБЛЕМ ВИСОКОГО НАВАНТАЖЕННЯ

1.1 Сучасні веб-технології

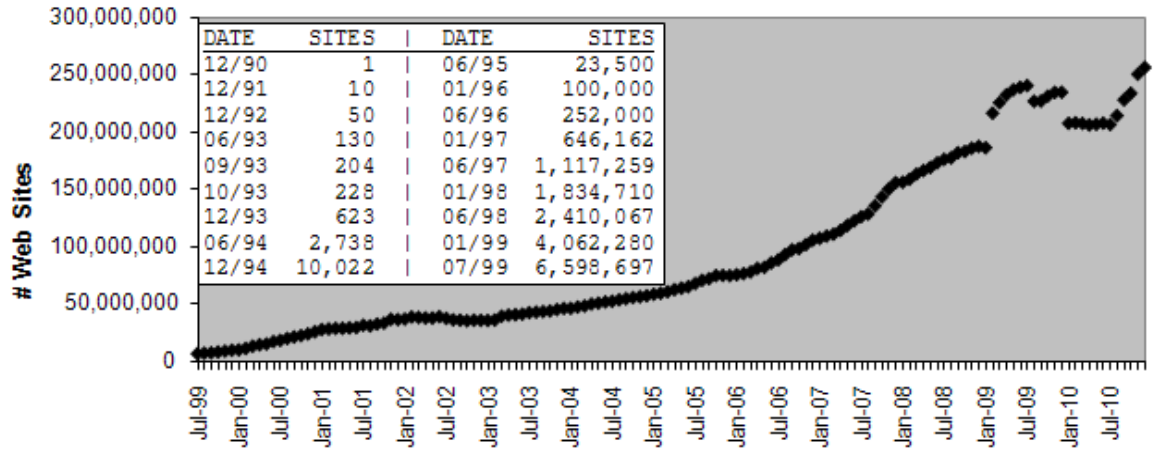
Среда зростання закону Мура породила революцію. Сучасні обчислювальні середовища набагато складніші та хаотичніші, ніж у перші дні вимірюваного підключення, дорогоцінних циклів процесора та обмеженої ємності. Дані та послуги мобільні та широко відтворюються для забезпечення доступності, продуктивності, 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 імен об'єктів, узгодженості, розташування та маршрутизації.

У наш час Інтернет став неминучою частиною нашого життя. Щогодини все більше і більше людей відкривають щось нове та цікаве в Інтернеті та діляться цим з іншими. У період між 2005 і 2010 роками 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 веб-сайт, адміністраторам та розробникам потрібно задуматися про багато речей, і іноді їм доводиться мати справу з науковими проблемами, а не технічними, які можуть виникнути через технічні. 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, не пов'язаними 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 між собою в 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 роботи, паралельність, керовану подіями, 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 продукту 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 керований подіями архітектурний дизайн дає вищі або такі ж придатні показники, 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 зростаючим значенням Інтернету як Інтернет-додатків виникає все більша потреба в точному моделюванні та відтворенні типових робочих навантажень 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 важким, оскільки робочі навантаження в Інтернеті мають ряд незвичних функцій.

По-перше, емпіричні дослідження роботи веб-серверів показали, що вони відчувають надзвичайно мінливі 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 забезпечити це зростання. 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.

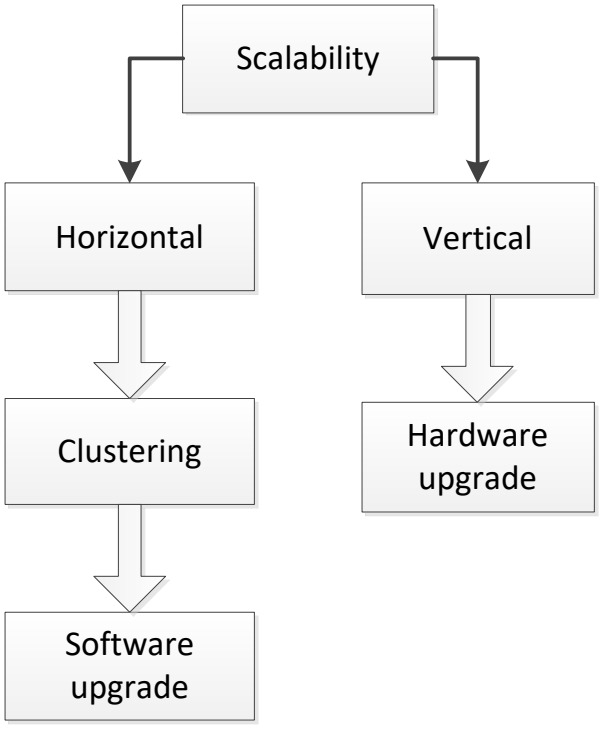


Рис. 2.2. Підходи до масштабованості

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 процесорних блоків 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, оскільки 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 балансування 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 балансувальнику 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 для виконання деяких спеціальних функцій, коли всі серверні сервери недоступні або недоступні. Це може включати переадресацію на резервний сервер або балансування 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 контролю 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 режимі очікування при відмові або ненормальному завершенні роботи раніше активного сервера додатків, 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 обслуговуванні.

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 їх 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 одними і тими ж 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, які можуть спричинити невдачу розповсюдження несправності іншими блоками.

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 вийшов з ладу. Зазвичай це

стосується кластерів, які мають кілька служб, що працюють одночасно; в окремому 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 (вузол очікування відмови після переходу в режим тимчасово стає активним, поки оригінальний вузол не може бути відновлений або повернути в Інтернет) і N-to-N - Комбінація / 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 розкрита проблема прикладної теорії управління зворотним зв'язком до якості кешування проксі-сервісу (QoS), а також проблема автоматизації налаштування контролера. На мережевому рівні теорія управління застосовувалася 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, які допоможуть застосувати теоретично-теоретичні методи проектування до більшого класу систем. Робота продемонструвала, що додавання можливостей теоретичного передбачення черг для управління циклами може підвищити ефективність 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 базувалася на теорії планування в режимі реального часу, яка стверджує, що час відгуку може бути гарантований, якщо використання сервера підтримується нижче заздалегідь 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 навантажень 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 пріоритетів досліджували для диференціації часу відгуку на веб-серверах [3, 4, 7, 8, 9].

У роботі [8] автори розглядали строгі 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 в тому випадку, якщо жодного запиту не існувало в жодному вищому пріоритетному класі. 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 найкращих показників.

Теорія управління зворотним зв'язком нещодавно набула великої популярності як аналітична основа для забезпечення 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, невід’ємних нелінійностей системи та варіацій системних параметрів у 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, швидше за все, призведе до 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.

2.3. Висновки до розділу

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 до управління зворотним зв'язком специфікація заданих точок контуру управління була використана 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.

РОЗДІЛ 3

РОЗРОБКА, ВИКОРИСТАННЯ ТА ТЕСТУВАННЯ АГРЕГАТОРА ДАНИХ ДЛЯ ЗАМОВЛЕННЯ ПОСЛУГ З МАГНІТНО-РЕЗОНАНСНИХ ОБСТЕЖЕНЬ

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 неблокувальну модель вводу-виводу, 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 арі 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:

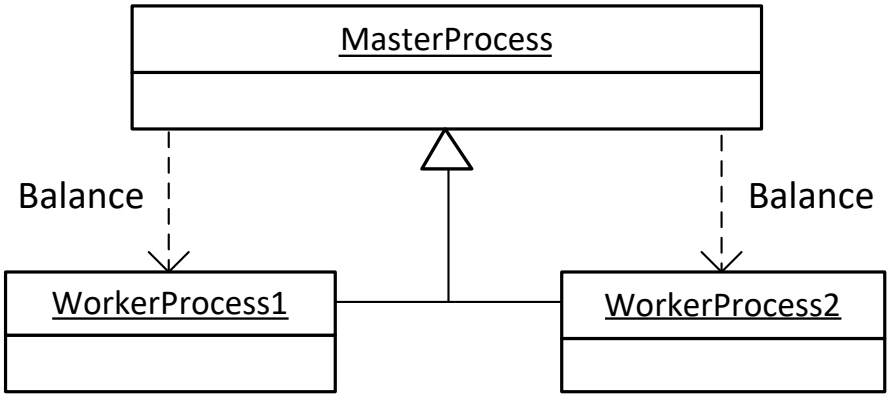
Швидкий доступ до властивостей (більшість механізмів JavaScript використовують структуру даних, схожу на словник, як пам’ять для властивостей об’єкта, що є повільним, оскільки кожен доступ до властивостей вимагає

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.



Малюнок 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 кількома процесами. Ця функція називається кластеризацією, коли єдиний порт tcp спільно використовується між кількома процесами (і відповідно циклами подій), кластерами, які всі однакові, але це дозволяє скористатися перевагами багатопроцесорних систем. Балансування навантаження працює у фоновому режимі і приховане від розробника.

Налаштування середовища NodeJS

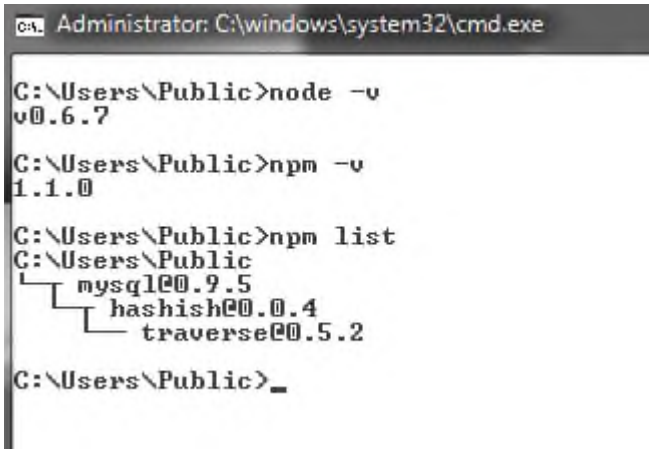
Набори NodeJS доступні для більшості популярних платформ, а також можуть бути скомпільовані для інших. Для встановлення вручну слід використовувати команду bash “make install”, у попередньо розпакованому каталозі або використовувати інтегрований менеджер пакетів Linux.

Для ОС Windows існує інсталятор, який автоматично пов'язує виконуваний файл, щоб бути видимим через будь-який каталог у командному рядку. Оскільки NodeJS представляє один виконуваний файл, він не має файлу конфігурації та 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).



Малюнок 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 - 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 потрібна версія 0.6.x або новіша. Він побудований на базі NodeJS, тому має зручну пряму залежність.

Щоб встановити npm на будь-яку операційну систему, слід завантажити вихідний код і зв'язати основний рутинний файл "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:

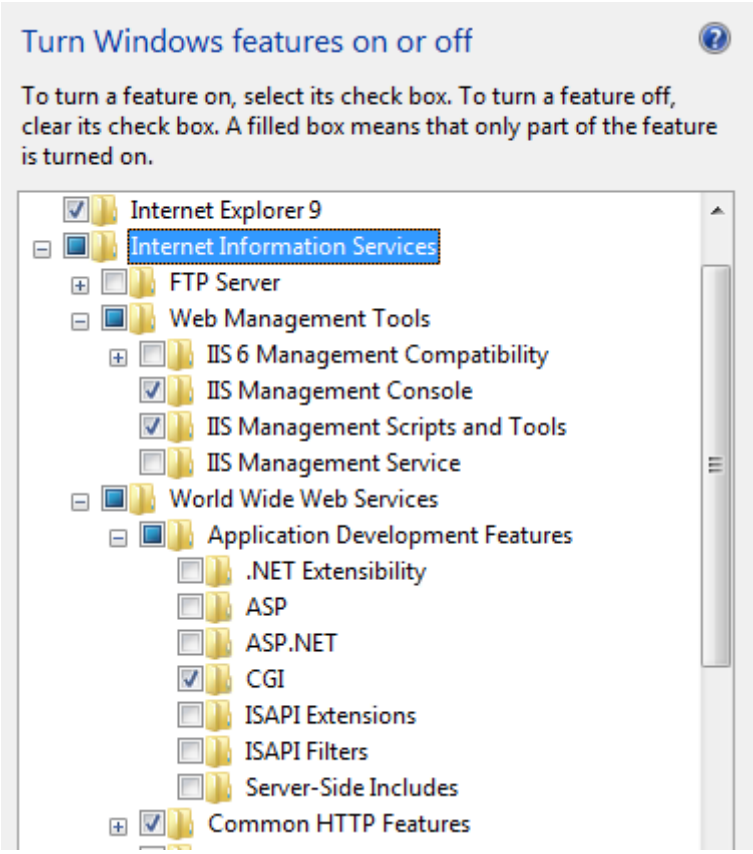
npm встановити mysql

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 всередині програми.

Налаштування IIS

IIS встановлюється та вмикається з коробки лише в операційних системах Windows Server, а 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 із встановлення можна знайти на офіційному веб-сайті http://learn.iis.net.



Малюнок 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 відразу. 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” на “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 виправити 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».

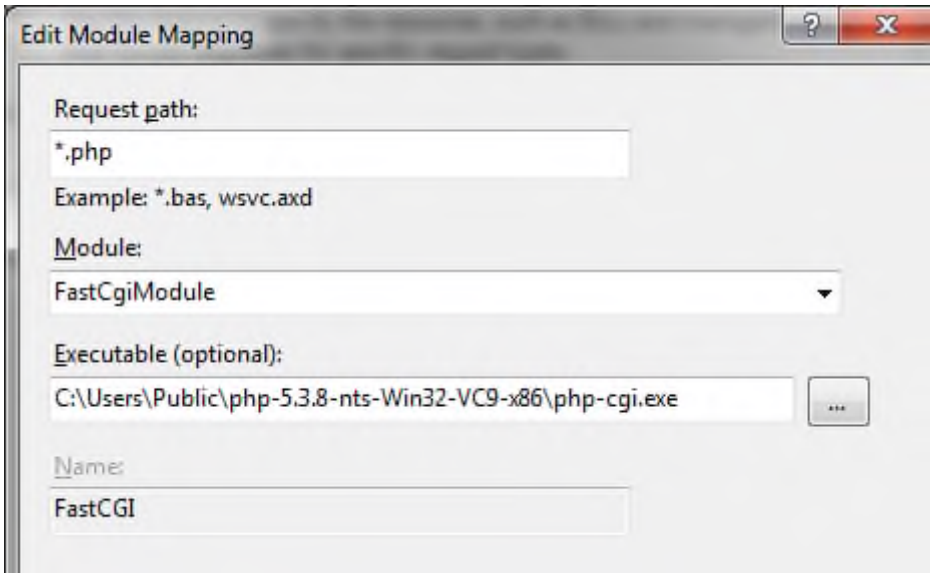
У діалоговому вікні, що з’явилося, потрібно заповнити всі поля (як показано на малюнку 0.4):

Шлях запиту: * .php

Модуль: FastCgiModule

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

Назва: FastCGI

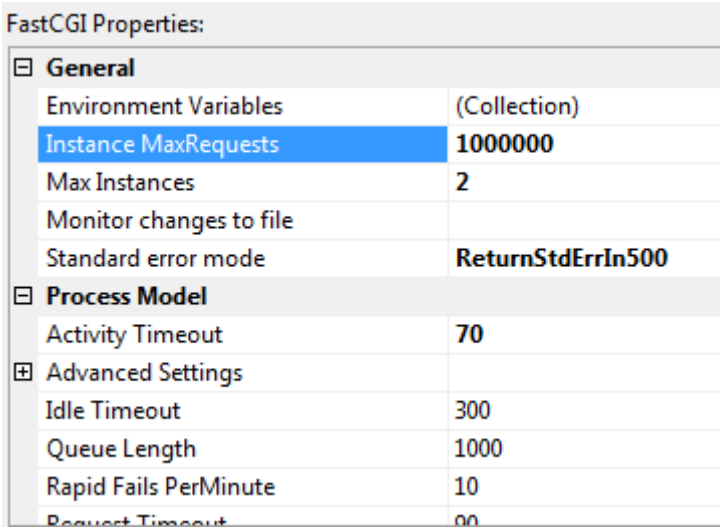


Малюнок 0.4 Призначення * .php-файлів до обробника PHP за допомогою модуля FastCGI

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.

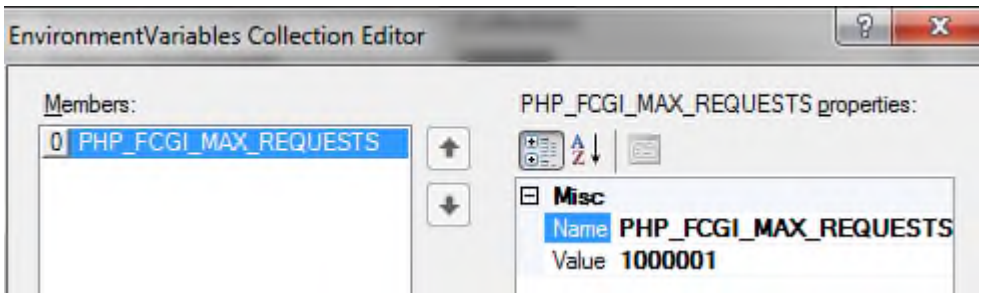
Для зручності слід додати файл індексу за замовчуванням PHP до папки. Виберіть “Документ за замовчуванням” і клацніть “Додати” на панелі “Дії”: введіть “index.php” у поле імені.

Щоб встановити високу продуктивність та повною мірою скористатися декількома ядрами процесора, слід налаштувати “Параметри FastCGI”.



Малюнок 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.



Малюнок 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»).

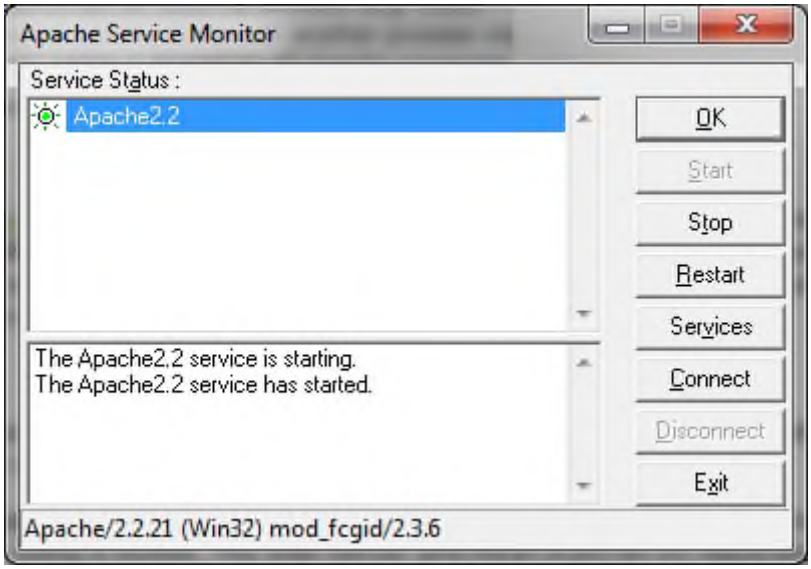
Конфігурація IIS для обробки файлів .php через FastCGI завершена і повинна відображати вміст вибраної папки, коли запитується http: // localhost / address.

Налаштування сервера HTTP 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.



Малюнок 0.7 Apache Service Monitor

Apache починається з читання конфігураційного файлу, який знаходиться в каталозі програми “conf” і називається “httpd.conf”. Після запуску та виконання попередніх дій (наприклад, відкриття файлів журналу) сервер запускає кілька дочірніх процесів, які виконують роботу з прослуховування та відповіді на запити клієнтів. Це контролюється вибраним багатопроцесорним модулем. Модулем багатопроцесорного керування 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 жодного 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 аналізу логічного 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’язковий).

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	varchar (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.

Базу 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 ПІДПИСАНИЙ НЕ НУЛЬНИЙ 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 можливу 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 збереження даних назавжди.

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 додаткові функції, недоступні в MEMORY: автоматичний 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.

```
1   <?php
2
3   $db = new mysqli('p:localhost', 'root', '', 'test');
4
5   if(isset($_GET['article_id']))
6   {...}
16  else
17  {...}
23
24
25  require_once "content_generator.php";
26
27  printHeader($title);
28
29  isset($article_id) ?
30      show_article($title, $content)
31      :
32      show_index($content) ;
33
34  printFooter();
```

Малюнок 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 для всіх сторінок.

Малюнок 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 згенерований PHP вихід 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. Малюнок 3.10) показує, як протікає потік, що обслуговує запити.

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

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.

Малюнок 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 (неінтерпретуючий), 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, щоб визначити приблизне навантаження активних користувачів, тривалість та поведінку. Були встановлені такі параметри:

Кількість ниток (кількість користувачів для імітації): 200

Початкова затримка, сек: 0

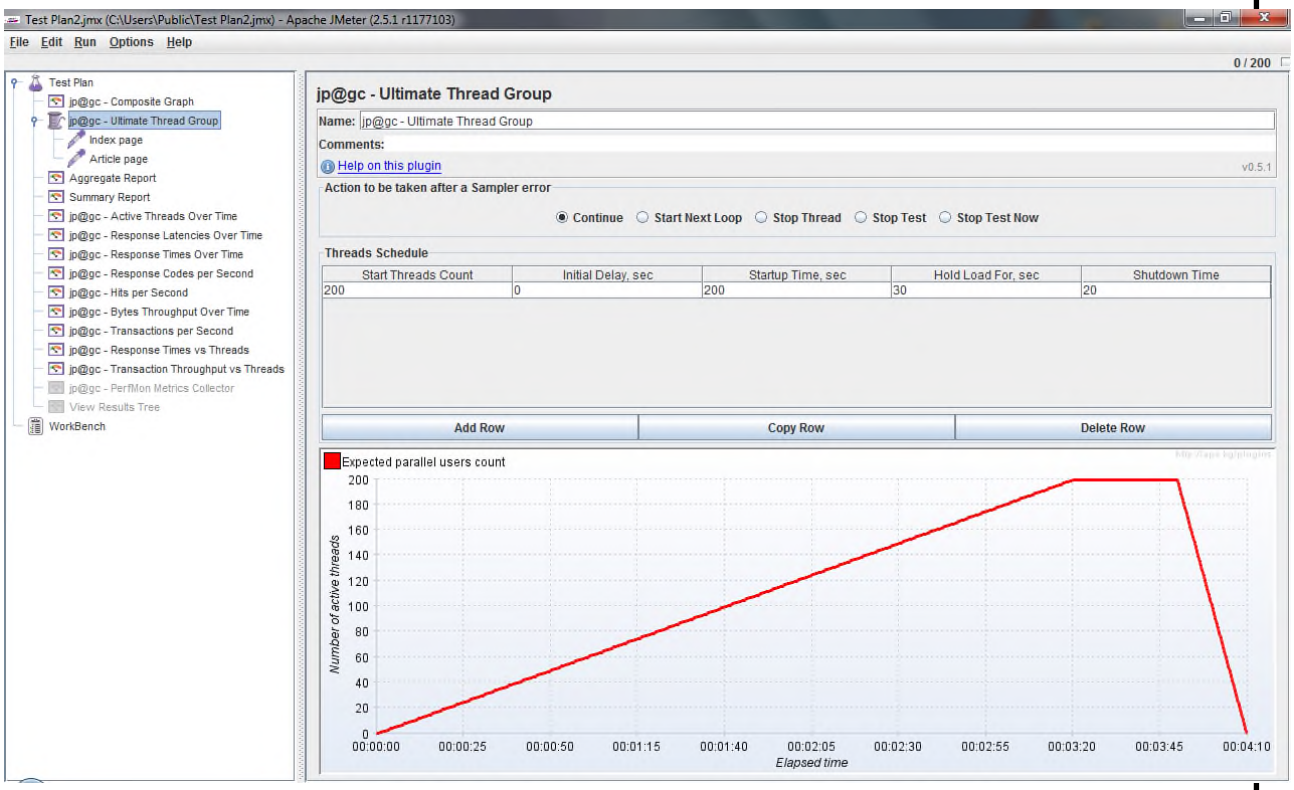
Час запуску (час створення загальної кількості потоків), сек: 200

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.



Малюнок 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);

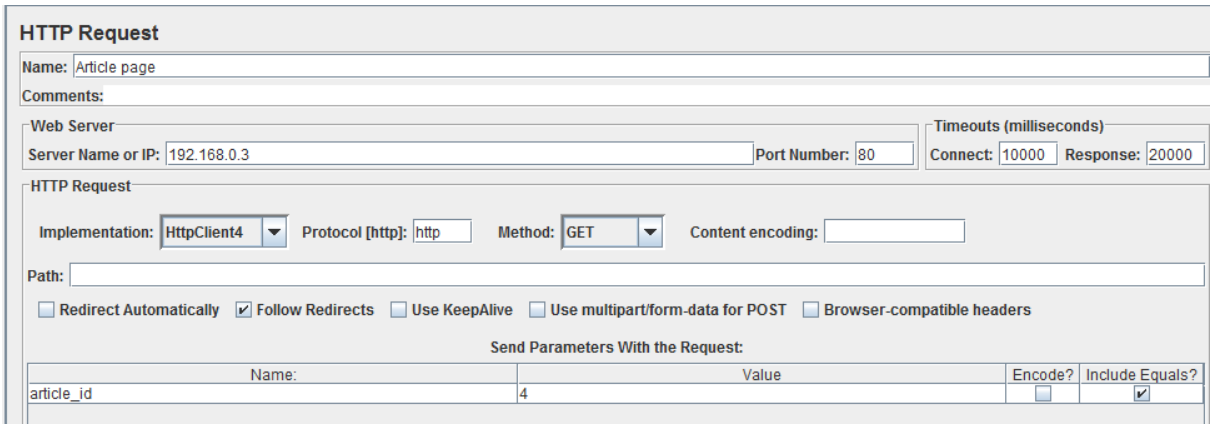
HTTPClient4 (частина Apache HttpComponents)

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 HTTP або Jakarta httpClient за замовчуванням, оскільки повторне використання з'єднання не контролюється користувачем)



Малюнок 0.12 Налаштований запит HTTP

Конфігурація для тестування індексної сторінки 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

Тестування (тестування) 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.

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 обладнання, програмне забезпечення та конфігурації мережі. Поглиблене розуміння всього тестового середовища з самого початку дає змогу зробити 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

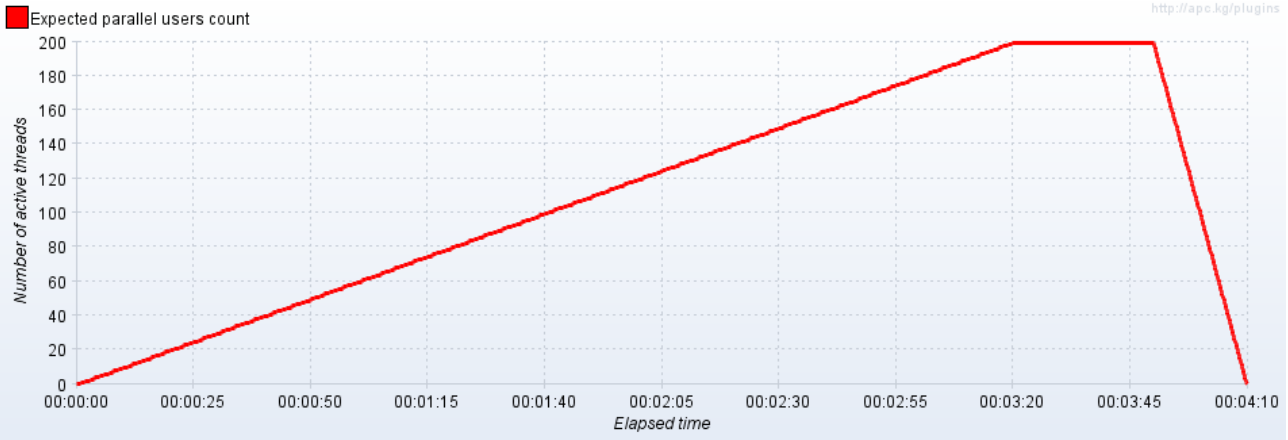
Зв'язок між тестом і ведучим: пряме з'єднання, 100 Мбіт / сек

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 при різному підрахунку 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 відгуку 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.



Малюнок 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, то це розглядається як стан перевантаження, що стався в середовищі додатків. Припускаючи прийнятне базове значення = 10, величина відхилення від поточного часу очікування при цільовій пропускній здатності визначатиме успіх чи невдачу.

Результати експерименту

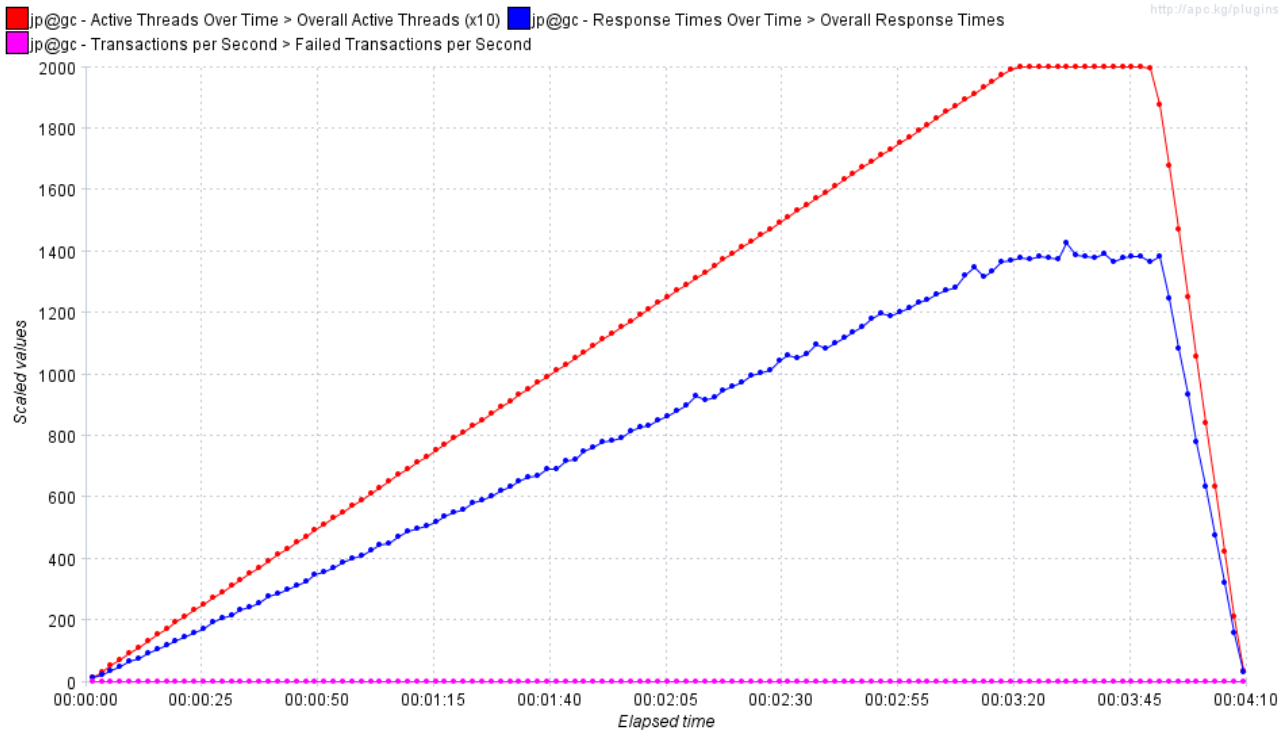
Основними характеристиками, які демонструють ефективність та стабільність веб-платформи під навантаженням, є час відгуку та загальна швидкість.

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 варіюється залежно від завантаження сервера (кількості підключень) та типу користувальницької діяльності, яку виконує користувач. Важливо вивчити, яку пропускну здатність сервер зміг витримати при зростаючому навантаженні.

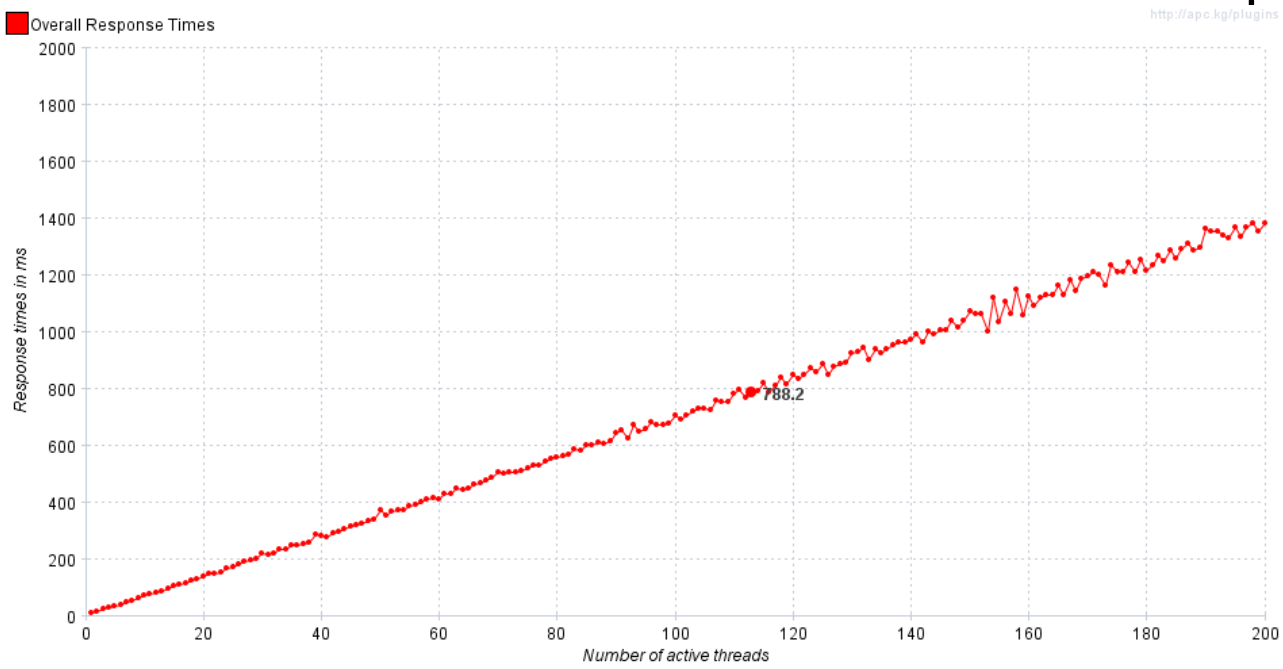
Складений графік ємності (див. Рис. 3.14) перекриває час відгуку, відсоток помилок та кількість активних користувачів на кожному інтервалі часу тесту. Потужність системи з точки зору одночасних користувачів - це точка на графіку, після якої ні відсоток помилок, ні час відгуку не виходять за межі допустимих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



Малюнок 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).



Малюнок 0.15 Час відгуку NodeJS проти потоків

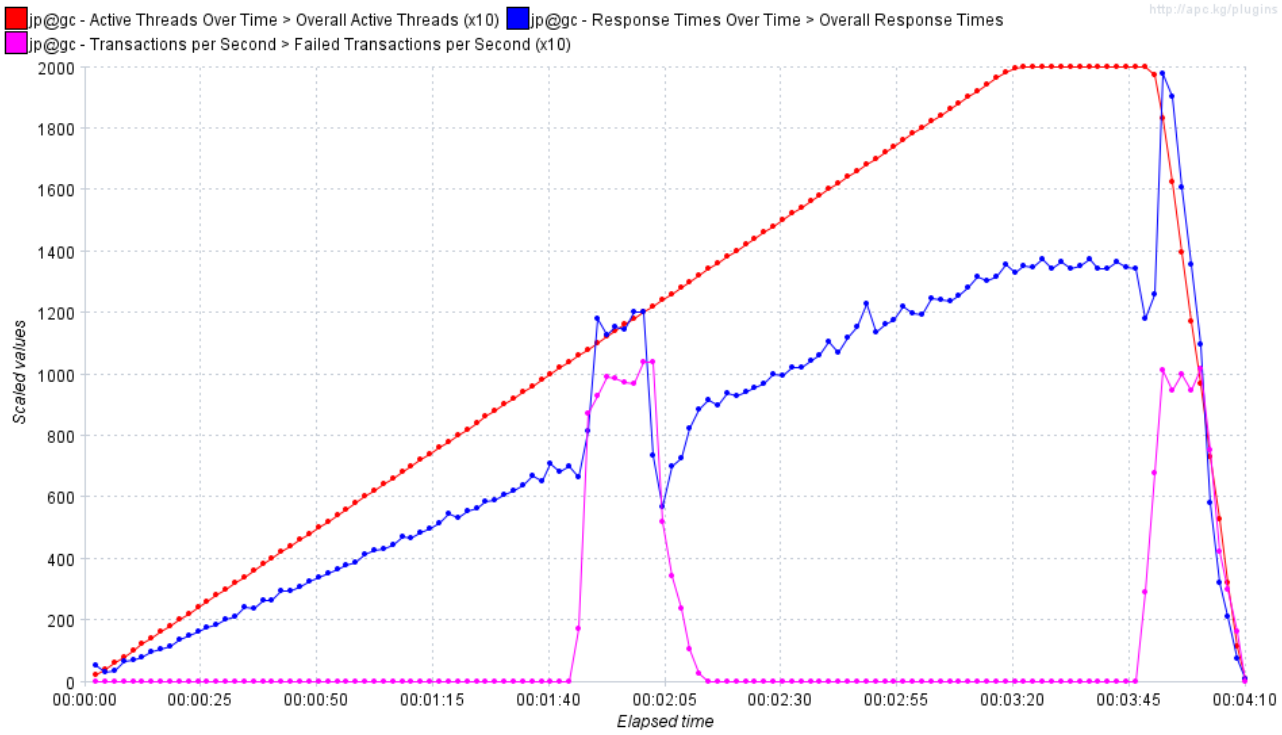
Час відгуку на активні потоки NodeJS показав майже ідеальну пряму лінію, - лінійну залежність часу відгуку від кількості активних потоків.

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.

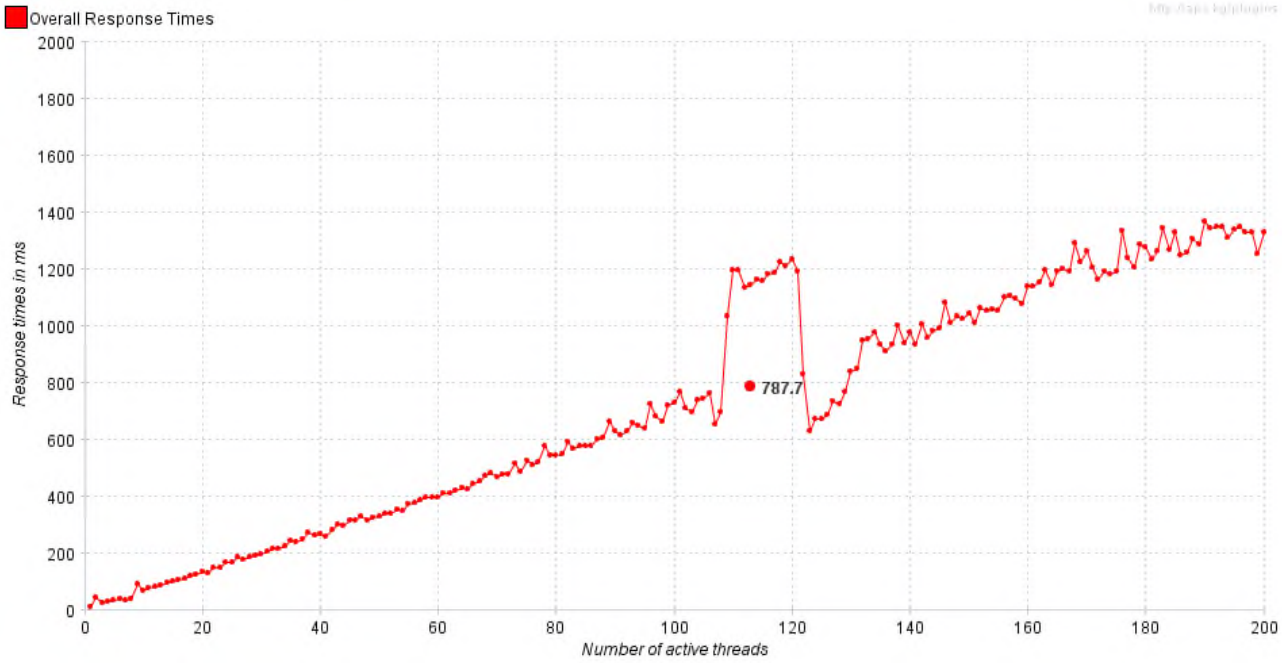
МКС продемонструвала подібну тенденцію показників ефективності (див. Рисунок 3.16), але з короткочасними сплесками сплесків часу відгуку та помилок. За цим графіком можна ідентифікувати піки часу відгуку при збільшенні навантаження. Допомагає виміряти ємність з точки зору кількості користувачів, які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

На певній 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.



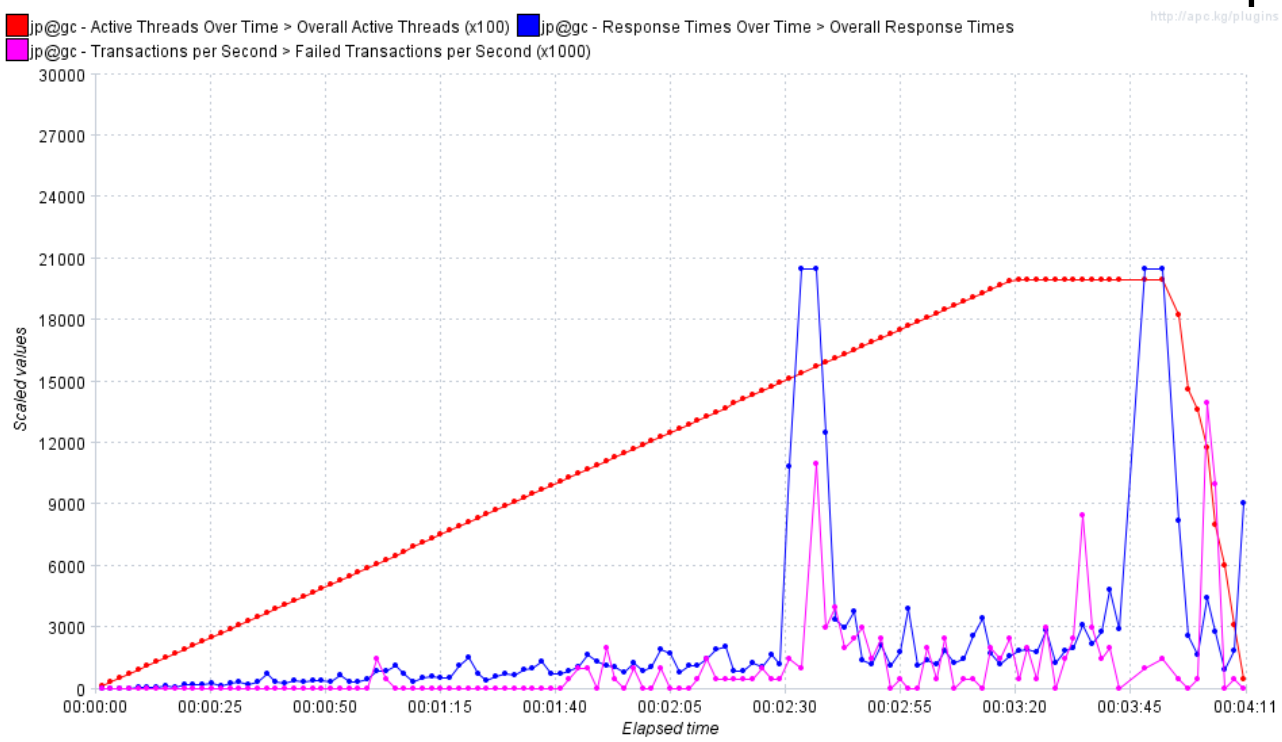
Малюнок 0.16 Складений графік IIS (активні потоки, час відгуку та помилка)

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



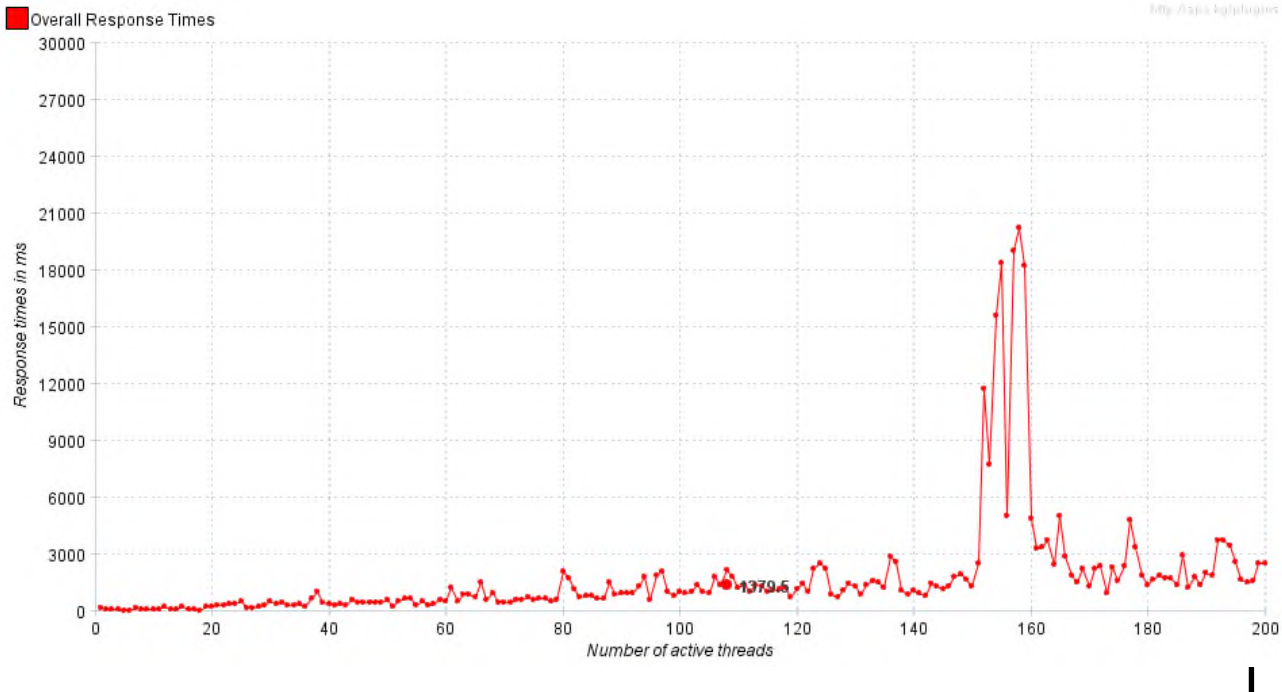
Малюнок 0.17 Час відгуку IIS проти активних потоків

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.



Малюнок 0.18 Складений графік Apache (активні потоки та час відгуку)

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



Малюнок 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

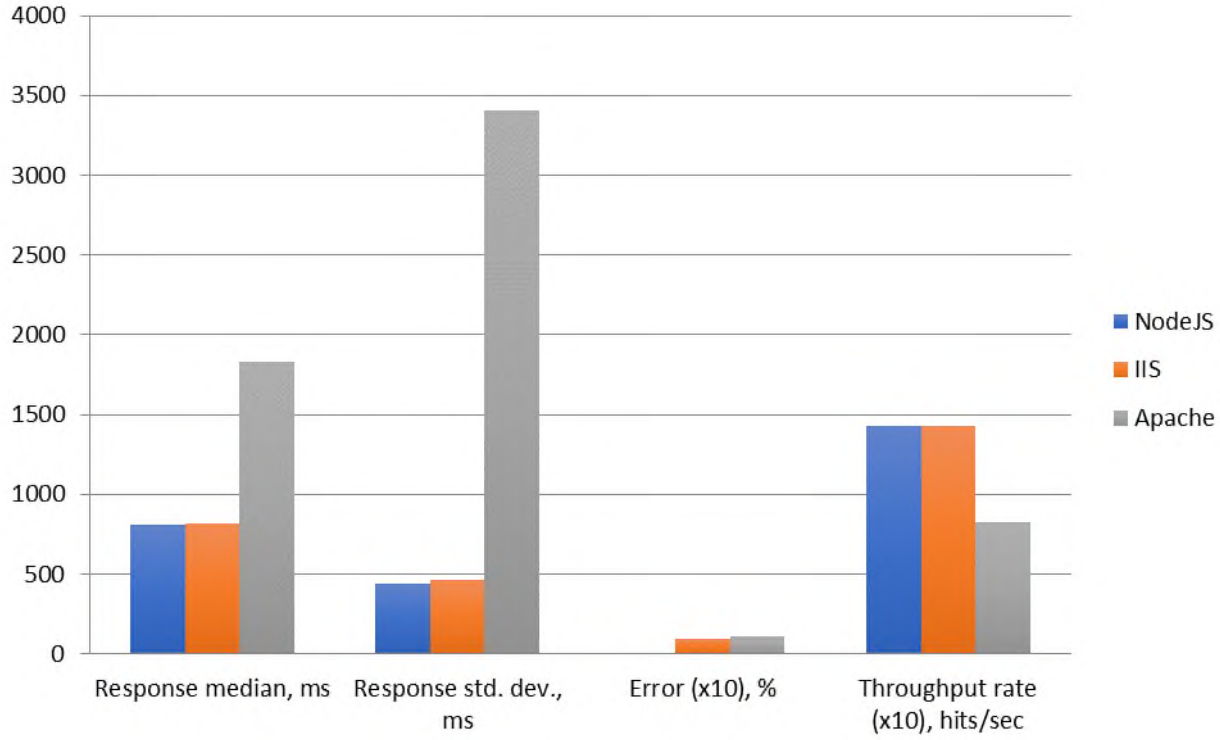
Підсумок результатів тесту

	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	1791 рік	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, який потрібно в секундах для отримання повної відповіді 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 в 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 не згладжується, або починає стати нестабільним, демонструючи високі стандартні відхилення від середнього значення.



Малюнок 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 масштабуватися 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 випливає висновок, що парадигма проектування асинхронної моделі, керованої подіями, з асинхронними неблокуючими масштабами вводу-виводу ідеально підходить і є ефективною для практичних потреб з великим навантаженням.

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 додатків 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.

Відкривши *HTML* файл на сторінці будуть відображені (рис. 3.3):

– кнопка “Додати”

– поле “Нікнейм”

– кнопка “*Twitter*”

– кнопка “*Instagram*”

– поле для відображення сторінок

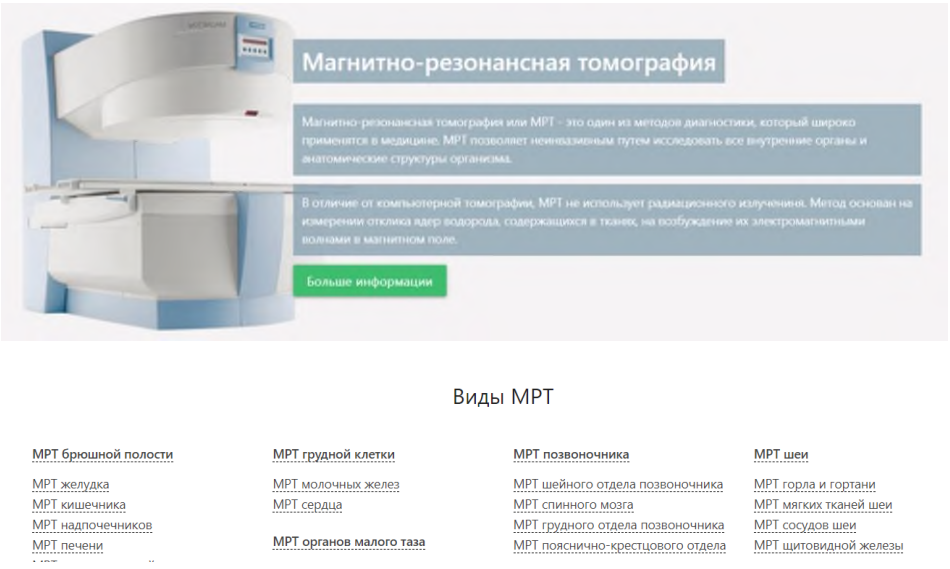


Рис. 3.3 Інтерфейс агрегатора для замовлення послуг з магнітно-резонансних обстежень

Поле “Нікнейм” призначене для вводу імені акаунту, який бажаємо агрегувати. Після заповнення даного поля слід натиснути на кнопку з зображення порталу для завантаження даних, цією дією буде викликана відповідна функція та запуститься робота агрегатора, і справа з'являться записи, які стосуються цього акаунту. На рис. 3.4 наведено приклад вигляду інтерфейсу програми з

доданою сторінкою, на якій буде відображена інформація по запрошеному користувачу. Також кожне таке поле має кнопку “Хрестик”, натискання на яку викликає функцію видалення/очищення даної вставки.

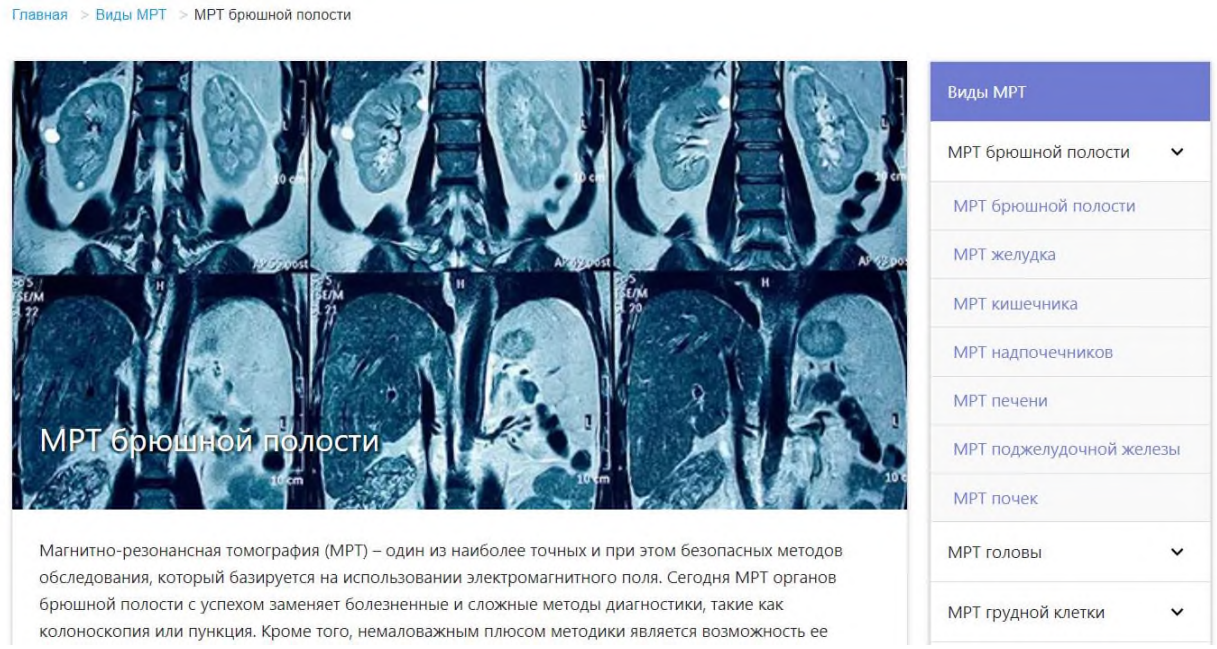


Рис. 3.4. Приклад вигляду інтерфейсу агрегатора

Після додавання декількох порталів сторінка буде мати вигляд, як на рис. 3.5.

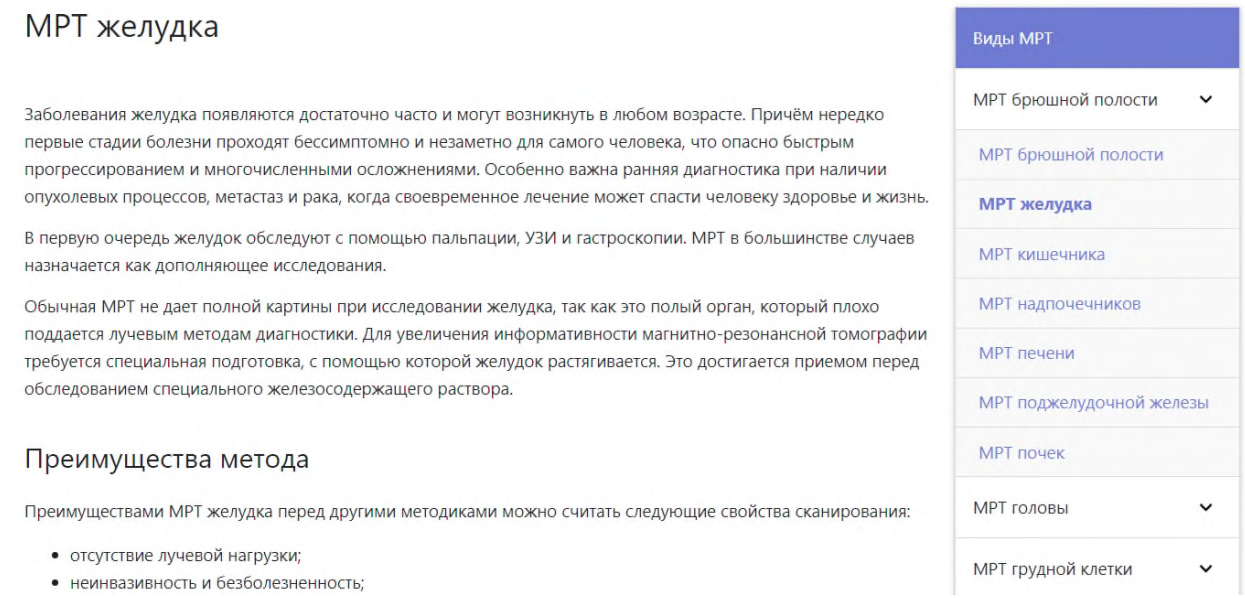


Рис. 3.5 Приклад вигляду інтерфейсу агрегатора

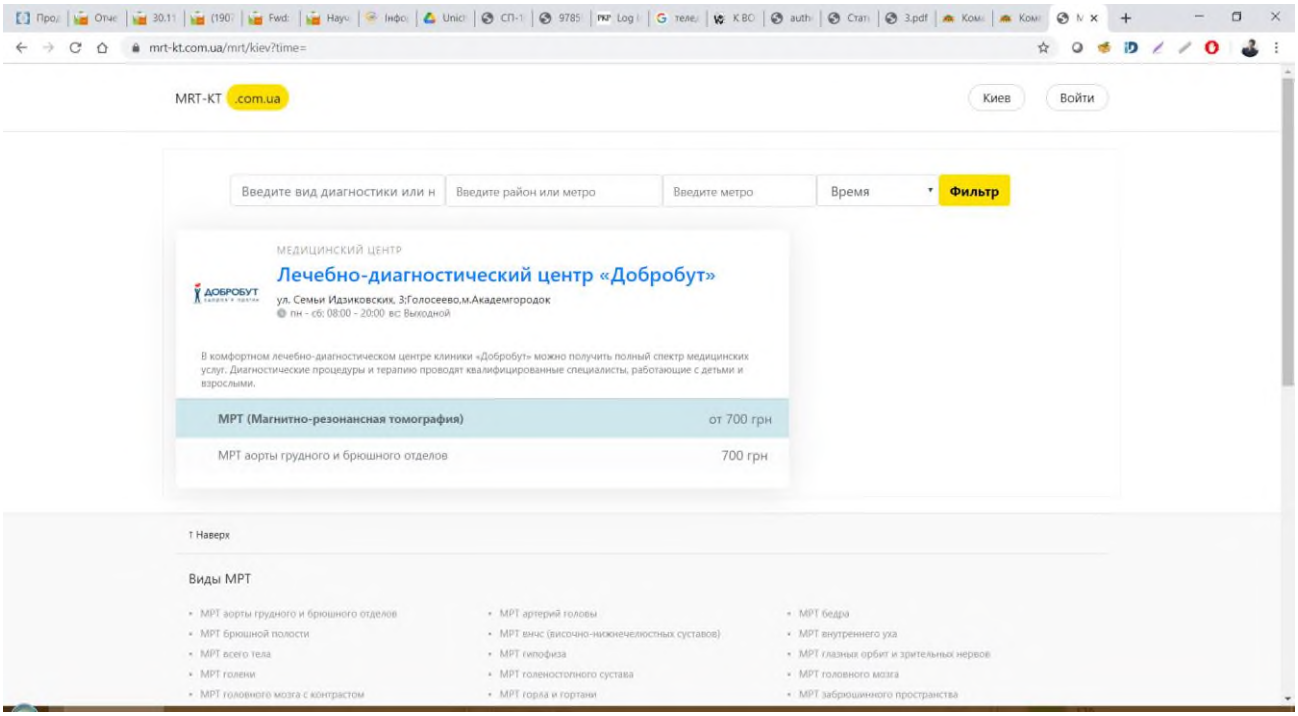


Рис. 3.6 Інтерфейс агрегатора

3.4. Висновки до розділу

Розроблено програму агрегатор соціальних мереж, яка має легкий для освоєння інтерфейс та може працювати як на локальній машині (за умови підключення машини до мережі *Internet*), так і на сервері з можливістю підключення по *http*.

На завершальній стадії розробки програма мала декілька вихідних файлів, які об'єднано в один файл інструментом *Webpack*. Вказавши точку входу в додатку *webpack* проаналізував файли і об'єднав їх в один вихідний *JavaScript*-файл, який містить все необхідне для запуску програми. Таким чином ми зменшили кількість вихідних файлів, не змінюючи код та структуру цих файлів. Отриманий основний файл завжди можна буде розкласти на початкові модулі.

Фреймворк *Vue* сильно допоміг в створенні користувацького інтерфейсу з візуалізацією нової інформації, та зміні *HTML* документу. *Vue.js* використовує віртуальний *DOM*, завдяки чому підвищується продуктивність програми.

Розроблена програма дає можливість агрегувати інформацію з соціальних мереж *Instagram* та *Twitter*.

ВИСНОВКИ

В дипломному проєкті проведено огляд архітектури типу клієнт-сервер. Провівши огляд існуючих агрегаторів для замовлення послуг з магнітно-резонансних обстежень встановлено, що наявні на ринку системи часто мають складний, з точки зору недосвіченого у користуванні персональним комп'ютером користувача, графічний інтерфейс. Також відрізняється і вартість послуг. Кожен сервіс має різні лінгвістичні алгоритми, покриття інформаційного поля і набір специфічних функцій.

Ця дипломна робота містить суттєву та актуальну інформацію про найбільш часто використовувані веб-платформи для високого навантаження з їх аналізом архітектури. Були розглянуті проблеми високого навантаження та фактори, що дозволяють характеризувати поведінку веб-платформ під великим навантаженням. Крім того, дослідження оцінює існуючі підходи до проектування послуг, включаючи паралельність на основі потоків та подій.

У дисертації було розглянуто дві найважливіші проблеми при розробці та розгортанні служб в Інтернеті: масштабування до граничних рівнів одночасності та підтримка надійної роботи під великим навантаженням.

У попередніх роботах зазначалося, що традиційні моделі програмування, засновані на потоковому та керованому подіями паралелі, не відповідають потребам масштабованості та управління перевантаженням для великомасштабних Інтернет-послуг.

Ця робота показує, чому керована подіями архітектура з асинхронним неблокуючим входом / виходом краще підходить для потреб високого навантаження. За допомогою порівняльного експерименту, підкріпленого характеристиками, виведеними з теорії черг (обслуговування масових надходжень в ефективну чергу) та теореми Літтла, результати експерименту підтверджують гіпотезу.

Результати цієї роботи рекомендовані для використання в дослідженнях щодо надійності великої кількості одночасних запитів, що обробляються, та у виробництві, де виникає вибір надійної веб-платформи для великих навантажень.

У цій галузі ця робота може бути використана для відповіді на питання вибору між потоковими та подієвими моделями для використання з великим навантаженням з урахуванням стабільності обробки запитів.

Подальші дослідження можуть зосередитись на тому, як складна та заплутана структура вихідного коду з багаторівневими вкладеними зворотними викликами впливає на сприйняття кодом програмістом.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Социальные сети и виртуальные сетевые сообщества / отв. ред. Верченов Л. Н., Ефременко Д. В., Тищенко В. И. — М.: ИНИОН РАН, 2013. — 360 с

2. Лоусон, Гарольд Путешествие по системному ландшафту: моногр. / Гарольд Лоусон. – М.: ДМК Пресс, 2016. – 368 *с.*

3. Буч, Гради Введение в *UML* от создателей языка / Гради Буч , Джеймс Рамбо , Ивар Якобсон. – М.: ДМК Пресс, 2015. – 496 *с.*

4. *Ryan Boyd. Introduction // Getting Started with OAuth* 2.0. – 1-е изд. – 1005 *Gravenstein Highway North, Sebastopol*: *O'Reilly Media, Inc.,* 2012. – 67 с.

5. *Alex Kyriakidis, Kostas Maniatis. The Majesty of Vue.js. – Packt Publishing Ltd,* 2016. – 230 с.

6. Федоров, А.Г., *JavaScript* для всех. – М: Компьютер-пресс, 2008.– 384 с.

7. Вільямс Г. 23 Дивовижні Статистики в Інтернеті і соціальних медіа в 2019 році. – [електронний ресурс]. Постійне посилання: *https://uk.wizcase.com/blog/*дивовижні-статистики-в-інтернеті/.

8. Фрейен Бен *HTML5* и *CSS*3.Разработка сайтов для любых браузеров и устройств; Питер – Москва, 2014. – 304 *с.*

9. *Drupal*. Создание и управление сайтом; Символ-плюс – М., 2010. – 576 *с.*

10. Гультяев, А. К. Проєктирование и дизайн пользовательского интерфейса / А.К. Гультяев, В.А. Машин. – М.: Корона-Принт, 2010. – 350 *с*

11. *Social Media & eventS RepoRt* 2012 [Електронний ресурс]/ режим доступу: *http://www.s-bc.ru/issueview/27a47da2-e423–406e-be5a*

12. Социальные сети и виртуальные сетевые сообщества / отв. ред. Верченов Л. Н., Ефременко Д. В., Тищенко В. И. — М.: ИНИОН РАН, 2013. — 360 с

13. ДСТУ 3008-95 Документація. Звіти у сфері науки і техніки. Структура і правила оформлення.

14. Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проєкти) випускників Національного авіаційного університету. – К.: НАУ, 2017. – 63 с.

Додаток А

Лістинг коду основного програмного модуля