УДК 681.3

ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ СИСТЕМ ІЗ СЕРВІСНО-ОРІЄНТОВАНОЮ АРХІТЕКТУРОЮ ЗА РАХУНОК ОЦІНКИ І РОЗПОДІЛУ НАВАНТАЖЕННЯ

Є.М. Туліка

Інститут програмних систем НАН України,
03680, Київ, проспект Академіка Глушкова, 40.
Тел.: (066) 709 5155, e-mail: vranen@gmail.com

Розглянуто основні підходи розподілу навантажень між веб-серверами та особливості балансування навантаження в сервісно-орієнтованій системі,. Запропоновано підхід до розподілу навантаженнь в сервісно-орієнтованих системах. Описано умови та правила використання підходу. Проведено дослідження ефективності та наведені результати експериментів.

The main approaches for workloading between web servers and load balancing features in service-oriented system was considered. Proposed an approach to distribute load in service-oriented systems. Analysis of the effectiveness was done and results of experiments is adduced.

Вступ

Із розвитком засобів передачі даних та мереж зв'язку все більшого розвитку набуває використання підходів розподіленого програмування. При побудові розподіленої системи обчислення великої задачі розбивається на кроки, які можуть виконуватись на різних комп'ютерах, об'єднаних спільною мережею, а після виконання результати їх роботи використовуються при обрахунках початкової задачі [1]. Існує багато випадків, де використання одного комп'ютера є, в принципі, можливим, але використання розподіленої системи прагматично більш виправдано. Наприклад, ефективніше, з точки зору вартості, отримати бажаний рівень продуктивності з використанням кластера із декількох недорогих комп'ютерів, порівняно з єдиним високовартісним комп'ютером. Розподілена система може бути більш надійною ніж нерозподілена, оскільки відсутній елемент, збій в якому призведе до краху всієї системи [2]. Крім того, розподілену систему простіше масштабувати ніж монолітну однопроцесорну систему, оскільки її проект від початку розрахований на взаємодію декількох комп'ютерів.

Якщо комп'ютери, на яких виконується розподілена програма, поєднуються мережею Інтернет, то обмін повідомленнями, такими як запит та відповідь, має відбуватись за протоколом HTTP, оскільки більшість проксі та брандмауерів не пропускають обмін даними мережею за іншими портами та а іншому форматі без додаткових налаштувань. Поняття веб-сервіса включило в себе всі ті програми – частини розподіленої системи – які отримують виклик на виконання та передають результати своєї роботи мережею Інтернет з використанням протоколу HTTP.

Типи реалізацій веб-сервісів можна розділити на дві категорії за підходом до формування повідомлень, які підтримує веб-серівс. Перші – використовують засоби протоколу HTTP: запити типу POST, GET – для передачі лише мінімальної кількості даних, необхідних для виклику операцій веб-сервіса. Такий підхід використовує принципи REST. REST – архітектура розподілених систем, яка наголошує на обмеженість різновиду запитів, які надсилаються до ресурса, а разом з тим – на збільшенні кількості ресурсів, яким вони адресуються [3]. Веб-сервіси, які використовують такий підхід належать до RESTful веб-сервісів. Друга категорія веб-сервісів використовує спеціальний формат даних SOAP, відповідно до якого форматуються дані повідомлень, передані у запиті, і який дозволяє розширити множину запитів, що оброблятиметься сервісом, до будь-якої необхідної кількості. При використанні цього формату до основних даних додається опис цих даних, та інформація, необхідна для їх обробки. Формат SOAP засновано на XML і, незважаючи на те, що розмір повідомлення при цьому значно збільшується, сучасні засоби передачі даних дозволяють нехтувати цією різницею. Натомість додається можливість автоматичного читання та формування повідомлень, оскільки завдяки універсальності XML використовуються одні і ті ж засоби роботи із повідомленнями у всіх програмах. Отже використання веб-сервісів стає незалежним від мови та платформи.

Але для реалізації розподіленої програми не достатньо розбити її функціональність на окремі процедури та реалізувати їх у вигляді веб-сервісів. Необхідним також є координація викликів операцій веб-сервісів та опис того, як результати їх роботи впливають на кінцевий результат. Така взаємодія веб-сервісів підпадає під визначення бізнес-процесу – набору пов'язаних, структурованих дій чи завдань, які ведуть до виробництва сервісу або продукту. Для підтримки виконання бізнес-процесів використовується імперативна мова BPEL, діалект XML, за допомогою якої описуються послідовності викликів веб-сервісів та обробка проміжних результатів.

Отже, йдеться про використання не одного чи декількох веб-сервісів, а про множину сервісів які можуть одночасно входити до реалізації різних розподілених програм. Для створення нової програми у подальшому потрібно створювати все менше нових сервісів, а зосереджуватись на реалізації бізнес-процесу. Таким чином досягається максимальна повторність використання написаного коду. Архітектура таких розподілених систем називається сервісно-орієнтованою [4].

Для ефективності роботи сервісно-орієнтованої системи критичною є швидкість, з якою буде отримано відповідь з кожного із сервісів. Веб-сервіси виконуються на серверах прикладних програм, які запущені на кожному вузлі мережі. Тому ефективність роботи сервісно-орієнтованих систем напряму залежить від продуктивності обчислювальної мережі. В даній роботі розглядається проблема балансування навантаження між комп'ютерами мережі, на якій побудовано сервісно-орієнтовану систему. Розглядаються основні підходи розподілу навантажень між веб-серверами, а також створено метод балансування навантажень між серверами прикладних програм у системах із сервісно-орієнтованою архітектурою.

Існуючі реалізації

Особливістю програмних систем із сервісно-орієнтованою архітектурою є генерація великої кількості повідомлень, які передаються мережею Інтернет. Ця особливість виникає із розподіленої архітектури таких систем: для того, щоб рознесені у середині мережі частини програми працювали над єдиною задачею, вони повинні обмінюватись даними – результатами своєї роботи. Кожен запит користувача на виконання одного з бізнес-процесів може спричинити генерацію та надсилання десятків запитів та відповідей до різних веб-сервісів.

Основним підходом до збільшення продуктивності обчислювального вузла, наприклад серверного оточення, на якому працює веб-сайт, є збільшення кількості серверів, які обробляють запити від клієнта [5]. Для того, щоб з точки зору клієнта такий набір серверів залишався одним сайтом – мав однакову адресу, мав універсальний механізм авторизації, на кожному із залучених серверів встановлюють точну копію обробників, а маршрутизатор перенаправляє запит клієнта до одного із серверів [6]. В результаті на однакові запити відповідь повертається також однакова, тобто із зовнішньої точки зору працює абстрактний сервер і не важливі деталі його внутрішньої реалізації.

Розподілення або вирівнення навантаженнь, що припадають на декілька серверів, дозволяє уникнути такої ситуації, коли пакети, що передаються мережею Інтернет, потрапляють на один і той самий сервер, в той час, як на інші не потрапляє жодного. Ця процедура необхідна за тих причин, що сервер, який обробляє велику кількість запитів, кожен наступний запит оброблятиме повільніше: із зменшенням системних ресурсів все більше часу витрачатиметься на перемикання між задачами, а при досягненні повного завантаження сервера обробка запитів відкладатиметься у чергу. Задача автоматичного балансування полягає в тому, щоб визначити, на котрий із серверів перенаправити черговий запит так, щоб його обробка завершилась найшвидше [7]. Класичні балансувальники навантаження, що розподіляють трафік між пулом веб-серверів, використовують наступні підходи.

Round robin DNS (RRDNS) – підхід, за якого DNS сервер клієнта розподіляє один URL між множиною ip-адрес (і таким чином серед декількох комп'ютерів) [8]. Балансувальник навантажень, який використовує round robin, звертається до кожного із серверів незалежно від їх спроможності прийняти задачу і це може стати проблемою у випадку завантаженості сервера в той момент, як підійшла його черга обробляти запит. Також при використанні цієї схеми необхідно, щоб всі сервери були гомогенними. В WebSphere Application Server V5.0 частина, яка відповідає за балансування навантаження, використовує алгоритм Round robin DNS із зваженими вузлами [9]. При використанні цього підходу алгоритм балансування випадковим чином вибирає один із серверів кластера. Перший успішний запит браузера перенаправляється на цього члена кластера і його вага зменшується на одиницю. Новий запит браузеру послідовно передається на інший сервер прикладних програм і вага його також зменшується на одиницю. Розподіл навантаження рівний між серверами до тих пір, доки один із серверів не досягає ваги рівної 0. З цих пір запити отримуватимуть тільки сервери, у яких вага вища за 0. Єдиним винятком із цього шаблону є випадки, коли жоден інший член кластера не здатний обробити запит або коли запит пов'язаний сесією з іншим членом кластера.

Браузер, запит якого пов'язаний із сесією, на певному сервері має весь час взаємодіяти із цим сервером для того, щоб всі дані, що необхідно зберігати під час сеансу, не втрачались, коли балансувальник навантажень перенаправить черговий запит на інший сервер. Такими даними можуть бути, наприклад, авторизаційні дані користувача, і неприпустимо створювати необхідність вводити ці дані при кожному оновленні сторінки. Ця проблема вирішується різними шляхами, наприклад, збереженням сесій у спільній для всіх серверів прикладних програм базі даних [10]. Але в цьому підході є суттєвий недолік - доступ до даних сесій відбувається занадто часто як для того, щоб зберігати їх в базі даних, навантаження на сервер бази даних при цьому значно збільшується. Тому найчастіше використовують підхід, коли першому звертанні клієнта до сервера кластера у cookies клієнта, записується ідентифікатор цього сервера з тим, щоб при наступних з'єднаннях клієнта однозначно перенаправляти його на цей сервер.

Розподіл навантаження випадковим чином – інший підхід, реалізований у WebSphere [9]. Запити передаються випадково вибраному вузлу кластера. На відміну від підходу round robin, ваги не беруться до уваги. Єдиним випадком, коли сервер не вибирається випадково, є той, коли запит асоційований із сесією користувача. Також до уваги не береться час, коли був оброблений останній запит. Це означає, що новий запит може обробитись тим самим сервером, що і попередній.

Використання диспетчера балансування навантажень – диспетчер отримує запит від клієнтів, надісланий до єдиної віртуальної ip-адреси. Балансувальник після цього використовує алгоритм балансування для розподілу роботи поміж реальними ip [11]. Цей алгоритм постійно виконує такі дії:

1) перевіряє дієздантність серверів;

2) періоично надсилає запит для виміру часу відповіді;

3) визначає стан використання пропускної спроможності та центрального процесора на кожному сервері;

4) підраховує кількість активних з'єднаннь на кожному сервері;

Кластерна технологія балансування авторизацій в SAP R/3 [12] робить вибір, на який з серверів перейде користувач для здійснення авторизації, базуючись на кількості користувачів, які в цей час вже авторизовані та на часі відповіді, кількість робочих процесів при цьому не враховується. Як тільки користувач авторизується на певному сервері, сесія користувача прив'язується до цього сервера.

Так, сервероцентроване балансування навантажень може здійснюватись або програмою, що виконується на постійному сервері, або у вигляді спеціального пристрою. Використання постійного сервера зі стандартною операційною системою, такою як Windows або UNIX, має переваги в тому, що він може бути замінений іншою машиною зі схожими характеристиками. Цей підхід розвивається такими виробниками як Resonate, Rainfinity [13], та Stonebeat [14]. Використання окремого пристрою має перевагу в швидкості, оскільки він базується на спеціальному процесорі, оптимізованому під конкретну програму. Такий підхід використовують F5 та Radware [6].

Проте єдина точка доступу до диспетчера балансування є тим місцем, збій в якому виведе з ладу всю побудовану систему. Тому використовується "гарячий резерв" – комп’ютер, який сконфігуровано як дзеркало комп’ютера, що виконує задачі диспетчера і який заміняється під час збоїв [15]. Він, також, називається пасивною гілкою, оскільки не використовується до виникнення виключних ситуацій. Обидва – активний та допоміжній балансувальники – періодично надсилають повідомлення на багатоадресний IP 225.0.0.2 для контролю їх дієздатності. Це UDP пакети у форматі VRRP (Virtual Router Redundancy Protocol) [16]. Cisco має власний формат HSRP (Hot Standby Routing Protocol) [17], а Extreme Networks – ESRP (Extreme Standby Router Protocol) [18]. Багатоадресне повідомлення може задіяти декілька підмереж, проте має дістатися фінального пункту призначення менше ніж за 200 або 300 мілісекунд. Пасивна гілка переймає повноваження активної, коли не отримує повідомлення про дієздатність від основної машини.

Direct Server Return (DSR) – метод балансування, за якого весь вхідний трафік потрапляє на віртуальну ip-адресу балансувальника навантажень [19]. Далі балансувальник змінює MAC-адресу пакета на одну з реальних адрес серверів в пулі та повертає його маршрутизатору, який вже точно доставляє пакет за призначенням. Таким підходом користується сервіс NLB (Network Load Balancsing) у серверних версіях операційної системи Windows [7].

Розподіл навантажень у сервісно-орієнтованих системах

Якщо розглядати сервісно-орієнтовану систему з точки зору програмного забезпечення серверної частини, яке забезпечує роботу системи, то можна виділити наступні обробники клієнтських запитів.

Http-сервер – сервер**,** який займається обробкою повідомлень протоколу http. Його основною задачею є опрацювання запитів протоколу HTTP та визначення обробника, який формуватиме відповідь на запит. Якщо це запит на статичні файли, такі як зображення або html-документи, то файли повертаються до клієнта, в іншому випадку виклик передається серверу прикладних програм, який дозволяє динамічну генерацію файлів.

Сервер прикладних програм (application server) – програмний каркас, виділений для ефективного виконання процедур (сценаріїв, операцій, програм), необхідних для підтримки ефективної побудови програм. Сервер прикладних програм діє як набір компонент, доступних для програміста через API, визначений платформою. Ці компоненти найчастіше представлені на тому ж комп'ютері, на якому працює http-сервер, і їх основна робота – підтримка динамічного конструювання контента. Сервер прикладних програм викликається Http-сервером, якщо він отримує запит на динамічний контент.

Засоби обробки SOAP – це програмне забезпечення, якому відомо, як обробляти запити та відповіді SOAP – специфікації протоколу для обміну структурованою інформацією між веб-сервісами в комп'ютерних мережах. Формат повідомлень базується на XML, і використовує інші протоколи прикладного рівня (найчастіше RPC та HTTP) для узгодження та передачі повідомлень. При обробці SOAP-повідомлення на його основі генерується stub, у термінах віддаленого виклику процедур – це представлення віддаленого методу у локальній програмі. Засоби обробки SOAP зазвичай виконуються разом із сервером прикладних програм, який викликає їх при отриманні SOAP-запиту.

Веб-сервіси – програми, які реалізують певний набір процедур. Складаються із реалізації цієї програми – якщо, наприклад, це сервіс, написаний мовою Java, то реалізація складатиметься із Java-класу, а операції представлені методами цього класу. Також, до веб-сервіса додається WSDL-документ. WSDL – XML-подібна мова, яка надає модель опису веб-сервісу. WSDL-документ складається з декількох частин:

перелік запитів протоколу прикладного рівня, наприклад, GET та POST в HTTP, які сприймає веб-сервіс;

прив’язка цих запитів до конкретної адреси в мережі;

абстрактне визначення типів даних, якими можна обмінюватись із сервісом;

перелік операцій сервіса.

Якщо веб-сервіс містить опис WSDL, то генерація SOAP-повідомленнь відбувається автоматично без участі програміста, оскільки точно описано все необхідне для виклику сервіса – назви операцій, формати даних та адреси в мережі.

Бізнес-процеси реалізуються з використанням серверних засобів виконання бізнес-процесів, скрипти до яких написані мовою BPEL. Серверні засоби працюють разом з сервером прикладних програм та на основі опису бізнес-процесу генерують веб-сервіс, операції якого – точки входу в бізнес-процес, виконання складається з викликів веб-сервісів і проміжної логіки, результат повертається у вигляді SOAP-повідомлення.

Архітектура розподілених Інтернет-систем складається з клієнтів та серверів. Клієнтом найчастіше виступає Інтернет-браузер, який ініціює запит до сервера, сервер обробляє запит та повертає відповідний результат. Запити та відповіді побудовані навколо передачі представлень ресурсів. Ресурсом є будь-яке послідовне та значуще поняття, яке може адресуватись. Представлення ресурсу – це документ, який фіксує поточний або бажаний його стан. Запити протоколу HTTP можуть бути наступних видів: GET – отримати представлення ресурсу, PUT – додати ресурс, POST – змінити ресурс, DELETE – видалити ресурс, проте семантика запитів може бути навмисно змінена впродовж обробки.

Першим обробником http-запитів виступає http-сервер, який визначає місце розташування ресурсу та намагається виконати дію, позначену запитом. Якщо ресурсом є статичний файл, а запит вимагає передачу його представлення, http-сервер в більшості реалізацій здатен сам виконати пошук у середовищі файлової системи та сформувати результат. Але якщо запит стосується віртуального ресурсу, який потрібно додатково генерувати, то наступним обробником такого запиту стає сервер прикладних програм (рис. 1).

Рис. 1. Процес обробки запитів у трирівневих клієнт-серверних системах

Оскільки засоби підтримки сервісної орієнтованості системи є надбудовою над звичайною архітектурою розподілених Інтернет-систем, то початкова обробка запитів відбувається за таким самим сценарієм. Відмінності виникають на етапі обробки запиту на сервері прикладних програм. Оскільки запит передається в форматі SOAP, то сервер прикладних програм викликає обробник SOAP, а наступний виклик направляється у веб-сервіс або бізнес-процес (рис. 2).

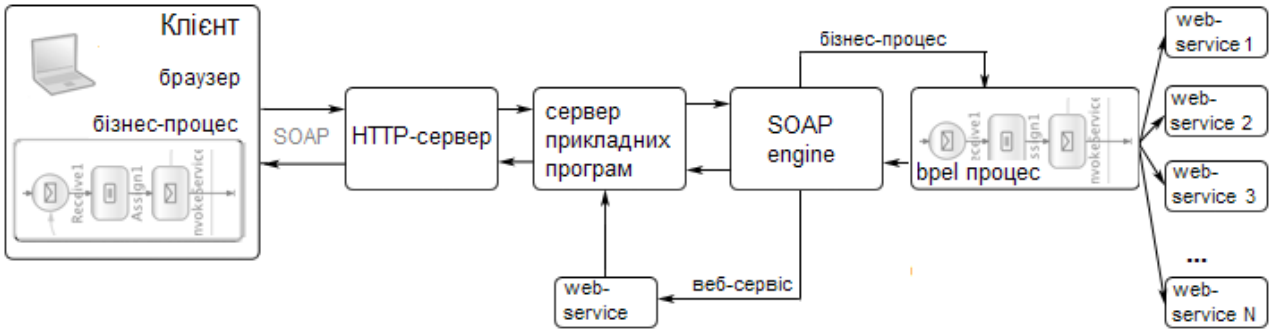


Рис. 2. Процес обробки запитів сервісно-орієнтованими системами

Балансування навантажень у Інтернет-системах здійснюється на етапі вибору одного із множини HTTP-серверів, котрий першим обробить отриманий запит. Існує реалізація балансування, за якої HTTP-сервер в системі один, а вибір здійснюється серед множини серверів прикладних програм. Тоді використовуються основні, вищеперелічені, принципи балансування навантажень. У системах із сервісно-орієнтованою архітектурою такі методи балансування також можуть використовуватись, оскільки процеси початкової обробки запитів збігаються. Ключові виробники засобів підтримки інфраструктури сервісно-орієнтованих систем, такі як ORACLE та IBM пропонують саме такий підхід розподілу навантажень – розподіляти запити на рівні http-серверів.

У сервісно-орієнтованих системах, на відміну від класичних Інтернет-систем, де клієнтом виступає браузер користувача, найчастішим клієнтом веб-сервіса є інший бізнес-процес. Крім того, відрізняється процедура обробки запиту в класичній Інтернет-системі та у сервісно-орієнтованій. Різниця між ними полягає у таких особливостях веб-сервіса:

- веб-сервіси проектуються без збереження стану між доступами. Вся інформація сесій користувачів належить, як правило, бізнес-процесу, або зовнішньому щодо нього скрипту, який передає у веб-сервіс лише необхідні дані;
- кількість програм, які виконуватимуться в оточенні сервера прикладних програм більша, ніж кількість запитів клієнта, оскільки виконання ініціюється бізнес-процесом;
- набір веб-сервісів на кожному з серверів різний, на відміну від динамічних скриптів веб-сторінок, які дублюються на кожному сервері із пулу;

- один веб-сервіс, у більшості випадків, виконує меншу кількість дій, ніж це необхідно, наприклад, при обробці дій під час доступу користувача до веб-сайту;
- множини веб-сервісів, із яких складаються різні процеси, можуть перетинатись.

Головною відмінністю балансування навантажень сервісно-орієнтованої системи від звичайних трирівневих клієнт-серверних систем є те, що визначення сервера, який оброблятиме запит, можливе ще перед тим, як запит буде надіслано. У сервісно-орієнтованих системах на фазі планування бізнес-процесу є можливість розподілу навантажень між різними серверами прикладних програм вибором сервіса, а відповідно і сервера, до якого буде здійснено виклик на наступному кроці обробки. Проте на етапі проектування доступним є лише статичний розподіл навантаження. У ситуації, коли, наприклад, відомо, що деякий сервер повільно з якихось причин обробляє запити, задля пришвидшення завершення бізнес-процесу можна викликати такий самий веб-сервіс на іншому сервері. Проте нічого не відомо про реальне навантаження на даний сервер в момент виконання бізнес-процесу.

На основі цього було реалізовано новий підхід розподілу навантажень, який використовує можливості бізнес-процесів встановити рівень навантаження на сервер, викликавши веб-сервіс, що надає таку інформацію, а виклик до нього є легким з точки зору обробки, перш ніж запустити на виконання складну для обрахунків задачу.

Модель виконання бізнес-процесу (рис. 3) з точки зору взаємодії із веб-сервісами, може бути представлена у вигляді зваженого орієнтованого графа $G=(V,E)$, де множина вершин V – операції веб-сервісів, які викликаються під час виконання процесу. Множина дуг E – множина впорядкованих пар операцій,

$$(A_1, A_2),$$

таких що, виклик операції, представленої вершиною A_2, відбувається після повернення результату від виклику операції A_1. При цьому паралельні частини бізнес-процесу представляються окремими підграфами. Ваги дуг – коефіцієнти, величина яких залежить від прогнозованого часу виконання операції веб-сервіса, представленого вершиною, в яку входить дуга.

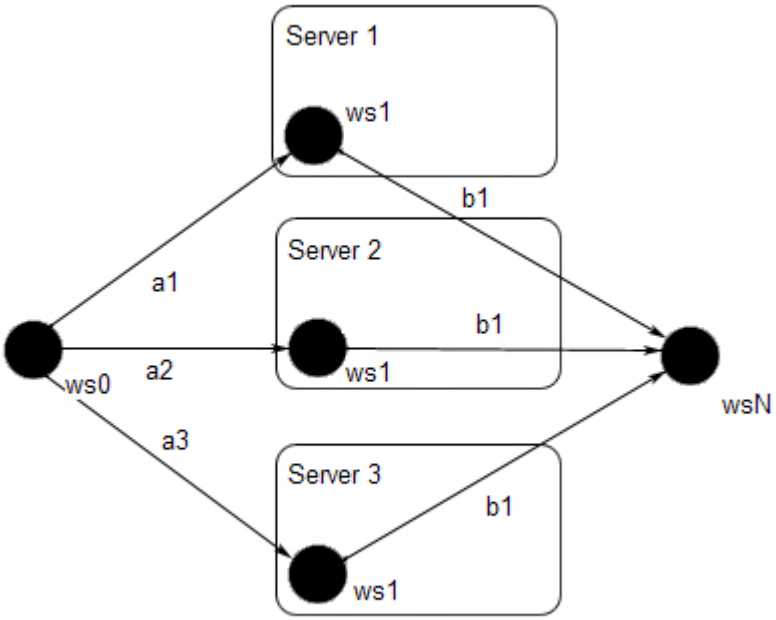


Рис. 3. Модель виконання бізнес-процесу

Із деяких вершин може вести більше ніж одне ребро. З точки зору бізнес-процесу не важливе фізичне місце розташування веб-сервіса, але в той же час копія одного і того самого веб-сервіса може міститись на різних серверах і котрий з них виконуватиметься в черговий раз, є важливим для моделі виконання бізнес-процесу, оскільки різні сервери в один і той самий момент часу можуть бути по-різному навантажені.

Задача розподілу навантаження між серверами представлена задачею вибору найдешевшого маршруту на графі G. Ваги ребер визначаються безпосередньо перед моментом вибору чергового ребра, для мінімізації похибки спричиненої тим, що навантаження на сервер може змінюватись багаторазово за проміжки часу менші, ніж час на проходження маршруту над графом. Стратегія прокладення найдешевшого маршруту полягає в приєднанні до маршруту ребер найменшої ваги.

Алгоритм визначення ваги ребер спирається на оцінку завантаженості сервера за важливими для продуктивності показниками.

Існує два основних підходи до виміру продуктивності комп'ютера:

- внутрішні зонди/агенти, які працюють на сервері як процеси або демони та надсилають значення лічильника у програму, наприклад, Windows Task Manager. Такий підхід називається білим ящиком, оскільки запит ініціюється у середині серверного оточення, в якому працює контролер;
- зовнішні монітори, які відповідають на команди оператора, надіслані мережею. Оскільки всі контролюючі запити ініціюються ззовні сервера, який контролюється, такий підхід називається чорним ящиком.

У табл. 1 наведено основні ресурси комп'ютера, завантаженість яких суттєво вливає на ефективність виконання програм, вказано величини виміру завантаженості ресурсів з поясненням щодо їх застосування до оптимізації.

Таблиця 1

Ресурс	Метрика	Опис
Центральний процесор	Довжина черги процесора	Довжина черги процесора визначає кількість потоків у черзі процесора сервера, які очікують на виконання процесором. Як правило, якщо їх кількість більша ніж два впродовж довгого часу, то процесор є вузьким місцем системи
	Процент процесорного часу	Лічильник надає інформацію про загальне використання процесора для сервера. Якщо значення впродовж довгого часу перевищує 70–80 %, процесор є вузьким місцем системи
	Процент привілейо-ваного часу	Цей лічильник вимірює, який процент загального часу виконання на процесорі виконуються в привілейованому режимі. Всі операції вводу-виводу виконуються в цьому режимі, отже високі (більше 20–30 %) значення впродовж довгого часу вказують на проблеми з мережею, жорстким диском чи будь-яким іншим інтерфейсом вводу-виводу
Оперативна пам'ять	Кількість сторінок пам'яті за секунду	Кількість сторінок за секунду – це кількість сторінок, прочитаних або записаних на диск для обробки помилки відсутності сторінки в оперативній пам'яті. Якщо значення довго перевищує 25–30 %, то пам'ять є вузьким місцем системи
	Розмір доступної пам'яті в байтах	Кількість пам'яті, доступної для виконання процесів на комп'ютері. Низьке значення (менше за 10 Мбайт) означає, що пам'яті замало
Жорсткий диск	Довжина черги жорсткого диску	Середня кількість запитів, відкладених в чергу для певного диску. Значення, тривалий час вище за 2 є індикатором вузького місця в засобах вводу-виводу
	Процент дискового часу	Процент часу, за який диск сервера був зайнятий обробкою запитів. Значення, тривалий час вище за 50 % визначає вузьке місце в жорсткому диску
Мережа	Кількість байтів у секунду	Визначає швидкість передачі даних в інтерфейсі мережі
	Довжина черги виводу	Визначає довжину черги пакетів, що очікують на передачу. Значення вищі за два є індикатором вузького місця в інтерфейсі мережі

Реалізація

З огляду на середовище застосування балансувальника навантажень, він був розроблений відповідно з вимогами сервісно-орієнтованої архітектури. У вищеописаному підході до балансування навантажень можна виділити такі основні функціональні компоненти.

Компонент для оцінки завантаженості серверного комп'ютера – працює окремо, на кожному комп'ютері із пулу. Основні задачі, які він вирішує, це надання інформації про системні характеристики комп'ютера, поточне навантаження за такими параметрами, як кількість вільної пам'яті та середнє заповнення пам'яті впродовж деякого проміжку часу, дані про навантаження на кожний із процесорів, також із статистикою щодо навантаження за останній час, швидкість відгуку на запит через мережу, кількість відкладених у чергу запитів. Компонент оцінки викликається зовнішніми бізнес-процесами, тому реалізований як чорна скриня у вигляді бізнес-процесу. Мова реалізації – Java, з використанням бібліотеки SIGAR, із відкритим кодом. SIGAR – це крос-платформний API для збору даних про апаратне забезпечення серверних систем. Бібліотека SIGAR включає підтримку Linux, FreeBSD, Windows, Solaris, AIX, HP-UX та Mac OSX крізь різноманіття версій та архітектур. Клієнту SIGAR API надається мобільний доступ до інформації про апаратуру а також данні для моніторингу. Ця інформація доступна в усіх операційних системах, проте кожна система має свій власний спосіб її надання. SIGAR забезпечує єдиний інтерфейс для доступу до ціє їнфонрмації незалежно від платформи. Ядро реалізовано на C, бібліотеки доступні для ряду мов, в тому числі для Java.

Реєстр запущених веб-сервісів – виконує задачу формування множини веб-сервісів, які реалізують функціональність, необхідну в бізнес-процесі в поточний момент. На основі побудованої множини здійснюється вибір одного із веб-серівісів, що оброблятиме запит. Реалізовано у вигляді веб-сервіса, операції якого дозволяють додати до реєстра веб-сервіс, додати адреси різних екземплярів цього сервісу, та отримати адреси всіх екземплярів необхідного сервісу.

Компонент для присвоєння ваги ребрам графу у побудованій моделі виклику веб-сервісів. Виконує задачу оцінки перспективності запуску кожного із набору веб-сервісів, доступних для запуску у поточний час. Працює разом із бізнес-процесами, тому встановлюється на кожному із серверів, що підтримують виконання бізнес-процесів. Реалізовано у вигляді бізнес-процесу, який надсилає запит до реєстра та отримує перелік веб-сервісів, встановлених на різних прикладних серверах в мережі. Далі, у паралельних потоках викликає веб-сервіси із оцінки навантаження на кожному із серверів у переліку. У тих самих паралельних потоках відбувається формування оцінки. Наприкінці бізнес-процес обирає найменшу із оцінок та повертає адресу веб-сервіса, зв'язок із яким вважатиметься оптимальним. Бізнес-процес реалізовано мовою BPEL.

Інтерфейс для виклику веб-сервісів. Оскільки мова BPEL не дозволяє змінювати адреси веб-сервісів під час виконання, для цієї мети був створений додатковий веб-сервіс, який містить операцію для виклику інших веб-сервісів. Операція параметризована адресою веб-сервіса, ім'ям операції, яка викликатиметься та параметрами цієї операції. Її додатковим завданням є збереження інформації щодо часу, який знадобився для виклику операції веб-сервіса. Ця інформація необхідна для побудови статистики щодо співвідношення реального часу та прогнозованого, а також для ранжування операцій веб-сервісів за часом опрацювання запитів.

Експеримент

Для оцінки ефективності запропонованого методу балансування навантажень була реалізована розподілена тестова система. Як обчислювальну задачу взято алгоритм Ерміта [20] пошуку псевдообернеої матриці. Вибір такого прикладу зумовлений тим, що в процесі пошуку псевдо-оберненої, над вхідною матрицею конвеєрно здійснюються важкі для обрахунку перетворення. Конвеєрна обробка найкраще демонструє використання бізнес-процесів та сервісно-орієнтованої архітектури, оскільки кожний етап конвеєра реалізується у вигляді операцій веб-сервісу, а інфраструктура – у вигляді бізнес-процесів.

Будь-яка не вироджена дійсна або комплексна матриця A має єдину обернену матрицю із такими властивостями:

$$AA^{-1}=A^{-1}A=I.$$

Наявність оберненої матриці гарантує, що у системи лінійних рівнянь $x = A^{-1}b$ існує єдине рішення. Обернену може мати лише квадратна матриця. Квадратна матриця A може мати обернену тоді і тільки тоді, коли A не вироджена, тобто тоді і тільки тоді, коли $\det A \neq 0$ або стовбці чи рядки А лінійно незалежні.

Якщо матриця прямокутна або квадратна і вироджена, то вона не має оберненої, проте існує узагальнена обернена матриця (g-обернена), яка має наступні властивості:

1) g-обернена існує для більш широкого класу матриць, ніж клас не вироджених матриць;

2) g-обернена розділяє деякі властивості звичайної оберненої матриці;

3) g-обернена збігається з звичайною оберненою матрицею, якщо А – квадратна та не вироджена.

Визначення. A – матриця розміром $m \times n$, $G-$ матриця розміром $n \times m$. Використаємо наступні матричні рівняння:

$$AGA = A, \tag{1}$$

$$GAG = G, \tag{2}$$

$$(AG)^T = AG, \tag{3}$$

$$(GA)^T = GA. \tag{4}$$

Якщо для матриці G виконуються рівності (1) – (4), то вона називається g-оберненою Мура – Пенроуза (псевдооберненою) та позначається A^+. Для обрахунку A^+ використаємо алгоритм діагональної редукції матриці $M=(AA^T)^2$ до форми

$$R = PAQ = \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix}$$

(канонічної форми Ерміта), де r – ранг матриці A, I_r – одинична матриця порядку r. Алгоритм Ерміта використовує рівняння

$$A^+ = A^T M_R A^T. \tag{5}$$

У цьому рівнянні легко знайти матрицю $M=(AA^T)^2$, найважчу частину складає обрахунок M_R^-, який потребує здійснення наступних кроків. Існує не вироджена матриця E, така, що $EM = M_1$ і яка має просту рядкову нормальну форму. Матриця M_1^T матиме просту стовпчикову нормальну форму, тому для неї знайдеться не вироджена матриця F, така, що

$$FM_1^T = R = \begin{bmatrix} I_k & 0 \\ 0 & 0 \end{bmatrix},$$

де k – ранг матриці M, а квадратна матриця R має одночасно і стовпчикову, і рядкову нормальну форми. Таким чином ,

$$FM^T E^T = R,,$$

а оскільки R симетрична, $R = EMF^T$. Звідси $M_R^- = F^T RE.$

Програма, яка реалізує цю задачу, базується на веб-сервісі для роботи із матрицями. Він містить операції множення матриць, транспонування матриць, формування одиничної матриці, зведення матриці у квадрат.

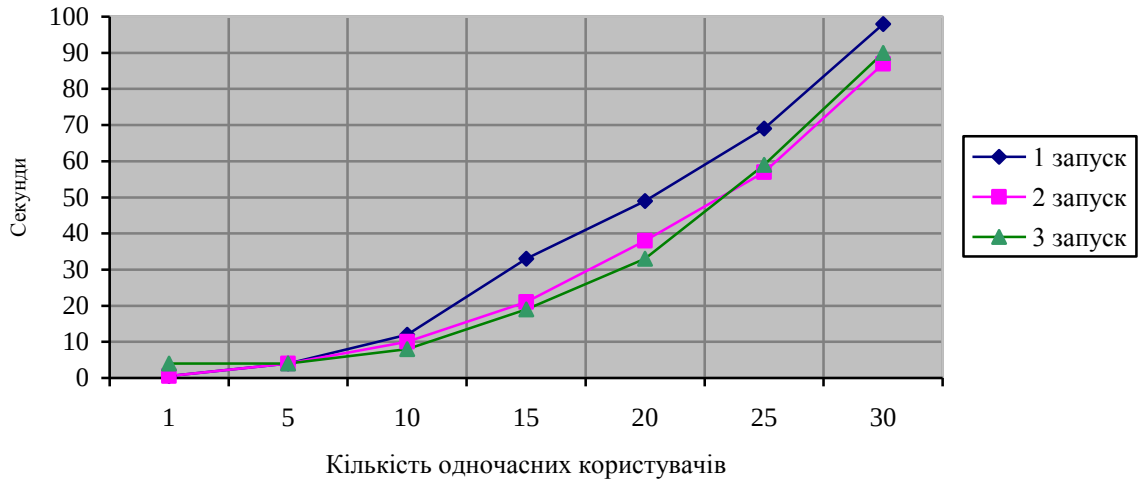
Окремими бізнес-процесами реалізовано знаходження матриці M, M_1, R та F. Бізнес-процес, який поєднує усі попередні бізнес-процеси обчислює матрицю M_R^-. Фінальний бізнес-процес підставляє знайдені значення у рівняння (5).

Тестова станція складається із шести комп'ютерів, на які встановлено сервер прикладних програм GlassFish v3 та HTTP-сервер Apache. Табл. 2 містить інформацію щодо конфігурації комп'ютерів та розподіленню веб-сервісів між ними.

Таблиця 2. Конфігурація тестової станції

Номер	Конфігурація	Задачі
1	Intel Pentium Dual Core E2180 2 GHz, RAM 2 Gb	Веб-сервіс основних матричних операцій, бізнес-процес M
2	Intel Pentium Dual Core E2140 1.6 GHz, RAM 2 Gb	Веб-сервіс основних матричних операцій, бізнес-процес M_1
3	Intel Pentium Dual Core E2140 1.6 GHz, RAM 3 Gb	Веб-сервіс основних матричних операцій, бізнес-процес R
4	Intel Pentium Dual Core E2140 1.6 GHz, RAM 2,99Gb	Веб-сервіс основних матричних операцій, бізнес-процес F
5	Intel Pentium Dual Core E2180 1.6 GHz, RAM 2 Gb	Бізнес-процес M_R^-
6	Intel Pentium Dual Core E2140 1.6 GHz, RAM 2 Gb	Бізнес-процес основного рівняння

Перший тестовий запуск проводився без використання балансування навантажень, другий запуск – з використанням балансувальника навантажень GlassFish, треті – з використанням розробленого підходу.



Графік показує, що при невеликій кількості клієнтів балансування сповільнює обрахунок задачі за рахунок підвищення кількості обмінів даними між комп'ютерами в оптимальній ситуації, коли вільних ресурсів більше, ніж потрібно для здійснення обрахунків. Проте з підвищенням кількості запитів з'являється необхідність пошуку вільних ресурсів і балансування допомагає в цій ситуації з їх локалізацією. Третя критична точка – коли ресурси зайняті повністю і навіть балансування не допомагає вирішити проблему.

Висновок

У роботі запропоновано метод розподілу навантажень, який покращує стандартні методи балансування за рахунок використання додаткових можливостей сервісно-орієнтованих систем. Було експериментально продемонстровано покращення результату з використанням побудованого методу, проте експеримент також показав можливості для покращення алгоритму системи за рахунок динамічного перерозподілу розташування веб-сервісів серед множини тестових комп'ютерів.

1. Bass, Len; Paul Clements, Rick Kazman (2003). Software Architecture In Practice, Second Edition. Boston: Addison-Wesley. pp. 21–24. ISBN 0-321-15495-9.
2. Elmasri, Ramez; Navathe, Shamkant B. (2000), Fundamentals of Database Systems (3rd ed.), Addison–Wesley, ISBN 0-201-54263-3 .
3. OASIS Reference Model for Service Oriented Architecture 1.0 http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm
4. SOA Glossary. Definitions for Service-Oriented Computing Terms. (http://soaglossary.com/stateful_services.asp
5. Bell, Michael (2008). Introduction to Service-Oriented Modeling. Service-Oriented Modeling: Service Analysis, Design, and Architecture. Wiley & Sons. pp. 3. ISBN 978-0-470-14111-3.
6. Tony Bourke: Server Load Balancing - O'Reilly, ISBN 0-596-00050-2
7. Network Load Balancing Technical Overview http://technet.microsoft.com/en-us/library/bb742455.aspx
8. Mourad A, Liu H., Scalable Web Server Architectures - Computers and Communications, IEEE Symposium on, pp. 12, 2nd IEEE Symposium on Computers and Communications (ISCC '97), 1997
9. WebSphere Application Server V5 Architecture, REDP-3721-00 - *Redpapers*, published 21 August 2003, last updated 19 August 2004
10. H Bryhni, E Klovning, O Kure A comparison of load balancing techniques for scalable web servers - IEEE Network, 2000
11. Rishi Khasgiwale Dispatcher-based Transparent-Dynamic Load Balancing algorithm for Web server clusters. - http://www.unix.ecs.umass.edu/~rkhasgiw/ece697a/rishi_report_ece697a.pdf
12. Server Selection and Load Balancing Using the SAP Web Dispatcher http://help.sap.com/saphelp_nw70/helpdata/EN/5f/7a343cd46acc68e10000000a114084/content.htm
13. EMC Symmetrix Optimizer. A Detailed Review http://www.emc.com/collateral/software/white-papers/c1062-emc-symmetrix-optimizer-wp.pdf
14. StoneBeat WebCluste Manual http://www.stonesoft.com/export/download/sb_man/SB_WC_25_AG.pdf
15. Dugan J.B. Sullivan K.J. Coppit D. Developing a Low-Cost, High-Quality Software Tool for Dynamic Fault Tree Analysis – Department of Computer Science, University of Virginia
16. RFC 3768, RFC concerning VRRP - http://tools.ietf.org/html/rfc3768
17. Cisco: HSRP Features http://www.cisco.com/application/pdf/paws/9234/hsrpguidetoc.pdf
18. Extreme Standby Router Protocol and Virtual Routing Redundancy Protocol www.extremenetworks.com/libraries/whitepapers/VRRPvsESRP_WP.pdf
19. On the Effectiveness of DNS-based Server Selection Shaikh A. Tewari R. Agrawal M. - IEEE INFOCOM, 2001.
20. Р. Грегори, К.Кришнамурти (1998) Безошибочные вычисления. Методы и приложения, М.Мир.