Висновок

Повнозв'язна нейронна мережа успішно може бути змодельована на графічному ядрі загального призначення з використанням технології CUDA. Але оскільки архітектура графічного ядра і відповідна технологія, що дає змогу одержати доступ до його обчислювальних можливостей, накладає ряд обмежень, то їх необхідно врахувати ще на етапі проектування моделі нейронної мережі та на етапі реалізації відповідних алгоритмів.

Виграш продуктивності з переходом на GPU становить приблизно два порядки, що недосяжно у разі використання тільки обчислювальної потужності центрального процесора. І це відкриває широкі можливості для застосування нейронних мереж у системах реального часу.

1. Гергель В.П. Теория и практика параллельных вычислений. – М.: Бином, Лаборатория знаний, 2007. – 424 с. 2. CUDA Zone. http://www.nvidia.ru/object/cuda_home_new_ru.html (Last access: 15.08.2012). 3. Каллан Р. Основные концепции нейронных сетей: пер. с англ. – М.: Вильямс, 2001, – 288 с. 4. Изопов П.Ю., Суханов С.В., Головашкин Д.Л. Технология реализации нейросетевого алгоритма в среде CUDA на примере распознавания рукописных цифр. – М.: Институт систем обработки изображений РАН "Компьютерная оптика", т. 34. – 2010. – № 2. – С. 243–251.

УДК 004.738

Н.Я. Павич, І.П. Костирко *
Національний університет "Львівська політехніка",
кафедра програмного забезпечення,
* кафедра електронних обчислювальних машин

ОСОБЛИВОСТІ ЗАСТОСУВАННЯ ТЕХНОЛОГІЇ ВЕБ-СОКЕТІВ ДЛЯ АСИНХРОННОЇ КЛІЄНТ-СЕРВЕРНОЇ ВЗАЄМОДІЇ ВЕБ-ПРОГРАМ

Проаналізовано сучасні підходи до реалізації асинхронної клієнт-серверної взаємодії веб-програм. Розглянуто ефективність та недоліки основних реалізацій. Для основних алгоритмів наведено принципи їх функціонування. Запропоновано підхід до застосування технології веб-сокетів для взаємодії в реальному часі.

Ключові слова: веб-програма, AJAX, асинхронна взаємодія, технологія опитування, черга запитів очікування, веб-сокети.

This paper analyzed existed approaches of implementation an asynchronous client-server interaction between/for application. There were considered efficiency and disadvantages of main algorithms and described the principles of their functioning. For web socket technology is described approach for real-time interaction.

Key words: web application, AJAX, asynchronous interaction, Pull technology, Push technology, Long pooling, WebSockets.

Вступ

З розвитком технологій передавання даних і разом з ними Інтернету розвивались і веб-програми – окремий вид клієнт-серверних програм, у котрих клієнтом виступає браузер, а сервером – веб-сервер. Хоч на початку розвитку можливості їх були обмежені, завдяки бурхливому розвитку Інтернету вони відіграють важливу роль у сучасному світі технологій і впевнено входять в повсякденне життя людей. Наприклад, за даними компанії Netcraft, що займається дослідженням Інтернету, кількість сайтів на червень 2000 р. досягла позначки понад сімнадцять мільйонів. Відтак показники (у вересні 2012 року) перетнули межу шістсот двадцять мільйонів веб-програм.

Все більше й більше їм відводять роль, що раніше належала традиційним додаткам, з якою вони успішно справляються. Одна з основних проблем веб-програм, яка не дозволяє поставити їх у один ряд зі звичайними прикладними програмами – інтерактивність, швидка реакція на внесені користувачем зміни. Також, зважаючи на доволі невисоку пропускну здатність каналу між компонентами веб-програм, важливо оптимізувати передачу даних. У статті розглянуто основні алгоритми взаємодії клієнта та сервера, описуються переваги й недоліки їх реалізацій. Описано підхід до застосування веб-сокетів для використання у динамічних веб-програмах.

Стан проблеми

Впродовж останніх п'яти років кількість інтернет-користувачів збільшилась майже удвічі, кількість веб-сторінок перевищила чисельність населення на планеті, обсяг трафіку в 2011 р. досяг 27483 ПБ/міс і в найближчі роки немає причин для уповільнення темпів зростання.

Основна частина інтернет-трафіку – веб-трафік, який генерують веб-програми. В загальному випадку, веб-програми побудовані на взаємодії клієнта та сервера через протокол HTTP, системі адресації URL та мові гіпертекстової розмітки HTML [1].

Отже, за традиційного підходу до побудови веб-програм виникає проблема інтерактивної взаємодії, спілкування клієнта та сервера в режимі реального часу. Клієнт не має змоги дізнатись про зміни актуальних для нього даних, не зробивши безпосереднього запиту на отримання цієї інформації. Клієнт-серверна архітектура побудови програм передбачає ініціативу лише з однієї сторони та обробку і відповідь з іншої.

Зважаючи на особливість HTTP протоколу, який не зберігає свого стану, виникає проблема надлишковості трафіку. В разі кожного звернення та відповіді надсилається службова інформація у вигляді заголовків, що при тривалому спілкуванні кожного разу дублюються. Немає можливості опустити ці дані, коли сервер вже «впізнав» клієнта, з яким він щойно взаємодіяв, і використовувати такі самі параметри, як і у попередньому запиті.

Розглянемо приклад запиту-відповіді для типової взаємодії у веб-програмі (рис. 1).

Запит	**Відповідь**
GET / HTTP/1.1 Host: google.com User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:15.0) Gecko/20100101 Firefox/15.0.1 Accept: text/html, application/xhtml+xml, application/xml ;q=0.9, */*;q=0.8 Accept-Language: uk,ru;q=0.8,en-us;q=0.5,en;q=0.3 Accept-Encoding: gzip, deflate Connection: keep-alive	HTTP/1.1 301 Moved Permanently Location: http://www.google.com/ Content-Type: text/html; charset=UTF-8 Date: Mon, 24 Sep 2012 21:58:14 GMT Expires: Wed, 24 Oct 2012 21:58:14 GMT Cache-Control: public, max-age=2592000 Server: gws Content-Length: 219 x-xss-protection: 1; mode=block X-Frame-Options: SAMEORIGIN

Рис. 1. Лістинг запиту-відповіді типової клієнт-серверної взаємодії

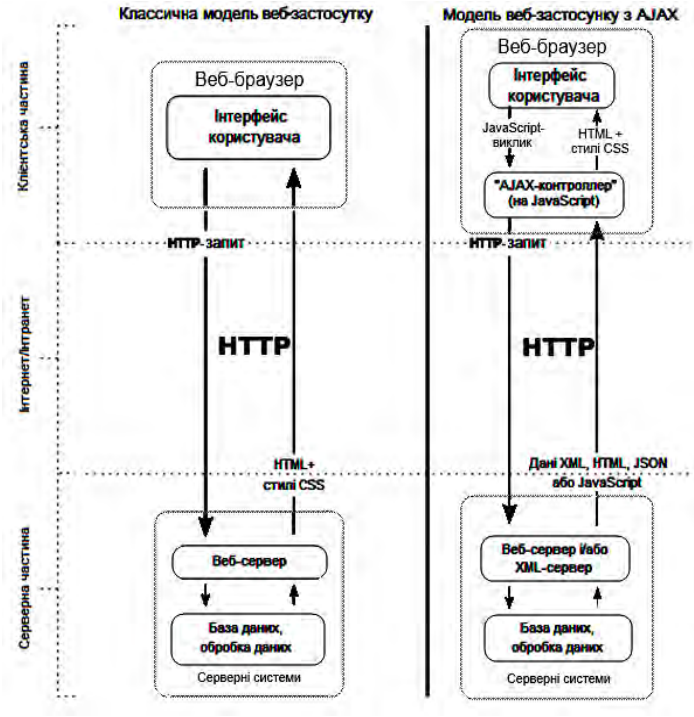
У результаті отримаємо 636 байтів (320 байтів та 316 байтів відповідно) службової інформації, що надсилатиметься під час кожного повторного звернення за цією самою адресою.

Постановка задачі

Описати підхід до застосування технології веб-сокетів для асинхронної клієнт-серверної взаємодії веб-програм. Проаналізувати основні алгоритми реалізації такої взаємодії, розглянути їх ефективність та недоліки.

Розв'язання задачі

Одне з рішень, що дає змогу реалізувати інтерактивність взаємодії – веб-сторінка, не перезавантажуючись, у фоновому режимі відправляє запити на сервер і отримує необхідні дані, відповідно до дій користувача (рис. 1). Такий спосіб взаємодії називають AJAX (Asynchronous JavaScript And XML) [2], що не є самостійною, стандартизованою технологією, а лише комбінує кілька основних методів та прийомів, базуючись на XMLHttpRequest [3].



а *б*

Рис. 2. Порівняння традиційної моделі побудови веб-програм (а) та моделі з використанням AJAX (б)

Основна особливість класичної архітектури “клієнт-сервер” чи AJAX полягає у чіткому розподіленні обов’язків, оскільки за такої взаємодії завжди є ведучий (master) та відповідач (slave). Тобто про зміну потрібних даних клієнт зможе дізнатись, лише регулярно посилаючи запити на їх отримання. При цьому дані можуть жодного разу не змінитись. Такий вид мережевої комунікації називається технологією опитування (pull technology).

На відміну від технології опитування, технологія надсилання (push technology) уможливлює таке поширення контенту в Інтернеті, коли інформація надходить від сервера до клієнта на основі ряду параметрів, що встановив клієнт [4]. Прототипом цього рішення були аплети (applets), що базуються на технології Java і дозволяють писати повноцінні програми у браузері, з поправкою на обмеження безпеки.

Безпосередньою реалізацією технології надсилання є періодичне опитування сервера стандартними запитами (polling). У відповідь сервер позначає цього клієнта як “на зв’язку” та відсилає набір подій, які відбулись з часу останнього звернення клієнта (рис. 3).

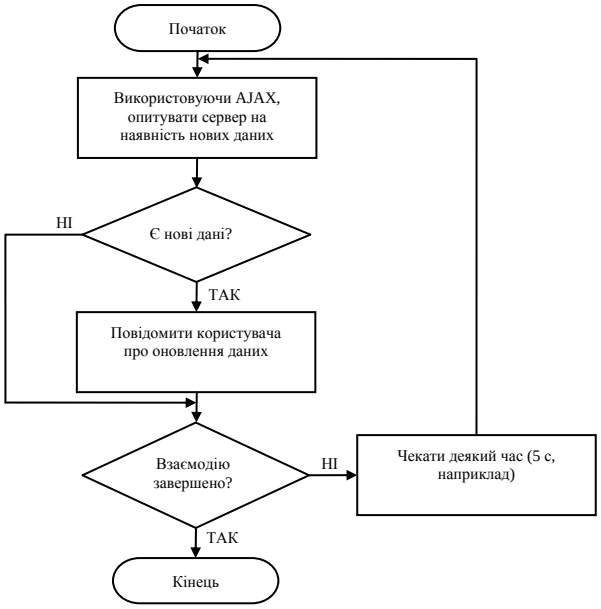


Рис. 3. Алгоритм роботи найпростішої реалізації технології опитування (polling)

За такого підходу з’являються дві основні проблеми. Перша проблема – великі затримки між отриманням даних, оскільки сервер відсилає їх не тоді, коли вони з’явились, а тоді, коли настане час чергового запиту. Затримка (T_{d1}) залежатиме [4] від часу між запитами (t_r), часу на встановлення даних (t_c) та часу на пересилання даних (t_s):

$$T_{d1} = t_r + t_c + t_s.$$

Друга проблема – це зайвий вхідний трафік на сервер. Під час кожного запиту браузер передає, відповідно до специфікації HTML, необхідний набір заголовків у нестисненому вигляді.

Досконаліший варіант реалізації технології надсилання – черга запитів очікування (long polling) та стійке з’єднання (streaming) [4]**.** Цей підхід полягає у тому, що сервер не закриває з’єднання, а залишає його відкритим. В разі надходження потрібної події він одразу надсилає відповідь про це одному чи багатьом клієнтам, що очікують її.

Загальна схема цього варіанта роботи така:

1. Клієнт виконує запит на отримання даних.
2. Створене з’єднання сервер не закриває, поки не з’явиться певна подія.
3. Подія відправляється у відповідь на запит.
4. Після отримання відповіді клієнт одразу ж відправляє новий запит для очікування події.

Отже, кожен запит означає нове з’єднання, яке буде відкрите стільки часу, скільки потрібно, поки сервер не вирішить відіслати інформацію у відповідь.

За такого підходу затримка обчислюється так:

$$T_{d2} = t_c + t_s.$$

У наш час черга запитів очікування є найпоширенішим рішенням серед усіх вищенаведених. Але й за такого підходу виникає проблема з кешуванням та HTTP-проксі, яка вимагає встановлення повторного з’єднання, в разі тривалої відсутності відповіді від сервера. Як і в попередньому випадку, наявний зайвий вхідний трафік на сервер для встановлення повторних з’єднань.

Усі вищеописані способи для забезпечення обміну даними в реальному часі використовують HTTP запити і відповіді з відповідними заголовками. Такі механізми взаємодії використовують багато надлишкової службової інформації, що забезпечує роботу протоколу, та додаткову затримку. Вони спрямовані на обхід традиційної архітектури веб-програм і намагаються надати їй більше гнучкості, імітуючи повнодуплексну комунікацію. Реалізація та підтримка таких рішень додатково ускладнює проекти, оскільки HTTP-протокол з самого початку не був призначений для двостороннього обміну даними у реальному часі.

Підхід на основі веб-сокетів. Новаторським рішенням вирішенням наведених проблем є використання веб-сокетів. Веб-сокети є одним з компонентів, що внесені до специфікації HTML5. Їх призначення – реалізація двонапрямленого каналу зв’язку для веб-програми через єдиний сокет, поверх Інтернету. Вони описують прикладний програмний інтерфейс, який дає змогу будувати масштабовані веб-програми, здатні працювати в режимі реального часу [5, 6].

Встановлення з’єднання. Для встановлення з’єднання клієнт та сервер використовують HTTP протокол зі спеціальним набором заголовків, через який відбувається “рукостискання” (“handshake”), як показано нижче:

Запит	**Відповідь**

`GET /text HTTP/1.1\r\n`
`Upgrade: WebSocket\r\n`
`Connection: Upgrade\r\n`
`Host: www.websocket.org\r\n`
`…\r\n`
	`HTTP/1.1 101 WebSocket Protocol Handshake\r\n`
	`Upgrade: WebSocket\r\n`
	`Connection: Upgrade\r\n`
	`…\r\n`

Рис. 4. Лістинг запиту-відповіді «рукостискання» між клієнтом та сервером

Додатково, разом з цими основними заголовками, можуть передаватись допоміжні. З їх допомогою можна вказати конкретну версію протоколу, передати налаштування параметрів безпеки з'єднання.

Встановивши одного разу з'єднання, клієнт та сервер мають можливість вільно обмінюватись фреймами даних між собою у будь-якому напрямі без обмежень на формат даних (рис. 5). При цьому TCP-з'єднання залишається відкритим.

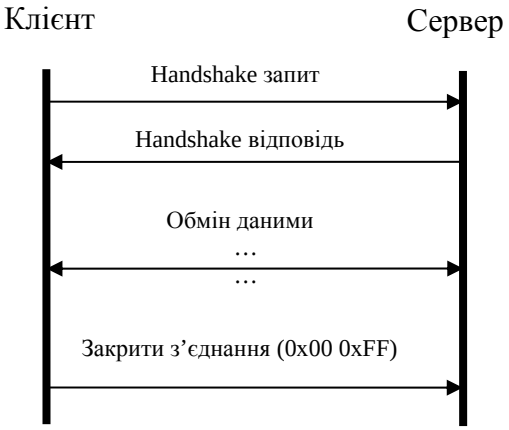


Рис. 5. Схема життєвого циклу клієнт-серверного з'єднання на основі веб-сокетів

Обмін даними. Після встановлення з'єднання сторони можуть вільно обмінюватись даними, поки сеанс не завершиться з ініціативи однієї зі сторін або не буде отримано статусу помилки. Обмін інформацією здійснюється за допомогою послідовності фреймів у спеціальному форматі і може виконуватись лише в межах відкритого сеансу.

Згідно з моделлю взаємодії відкритих систем (OSI), веб-сокети, як і HTTP протокол, належать до протоколів прикладного рівня. Хоча, технологічно, веб-сокети – це надбудова над HTTP протоколом, однак їх треба розглядати як окремий незалежний протокол. Для реалізації клієнт-серверної взаємодії веб-сокети використовують з'єднання на основі TCP-сокетів та реалізують безпечний та прозорий програмний інтерфейс для їх використання. А браузери (чи інші веб-агенти), своєю чергою, реалізовують цей інтерфейс для JavaScript.

На рис. 6 зображено схему взаємодії клієнта та сервера в разі успішного встановлення з'єднання.

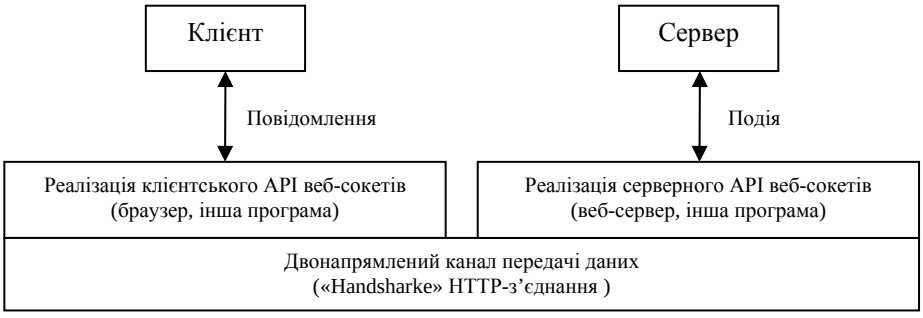


Рис. 6. Схема типової клієнт-серверної взаємодії на основі веб-сокетів

На клієнтській стороні, за допомогою JavaScript, забезпечується можливість надсилання та приймання повідомлень. Спосіб обробки цих повідомлень залежить від виконуваних задач.

Формат даних. Повідомлення, якими обмінюються клієнт та сервер, упаковуються у фрейми зі спеціальним форматом, відповідно до специфікації протоколу. Сторона, що приймає, виконує цю операцію у зворотному напрямку.

Щоб запобігти можливим проблемам під час проходження крізь мережу (таким як перехоплення проксісерверами) та з міркувань безпеки, фрейми, що надсилає клієнт до сервера маскуються. Відповідь, своєю чергою, відсилається без маски. Якщо ж клієнту надходить фрейм з маскою, то генерується помилка і з'єднання закривається.

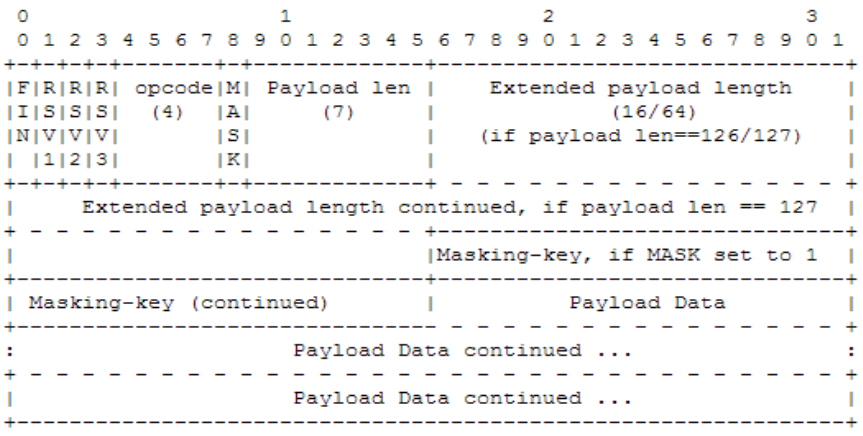


Рис. 7. Формат фрейма даних протоколу веб-сокетів

На рис. 7 зображено структуру кадру повідомлення.

FIN (1 біт) – поле, встановлене в тому випадку, якщо цей фрейм є останнім в повідомленні. Встановлюється також у тому випадку, якщо повідомлення складається лише з одного фрейма.

Наступні 3 біти зарезервовані й повинні набувати нульового значення. В разі встановлення в «1» з'єднання переривається.

OPTCODE (4 біти) визначає, як інтерпретуватимуться дані в повідомленні. Це можуть бути текстові дані, бінарні або керуючі.

MASK (1 біт) встановлюється в тому випадку, якщо дані передаються у напрямку від клієнта до сервера і це означає, що ключ маскування (Masking key) наявний.

PAYLOAD LEN (7 біт, 7 + 16 біт, 7 + 64 біт) – містить довжину корисної інформації (PAYLOAD DATA) у байтах. За першим байтом визначається розмір цього поля (0-125, 126, 127).

MASKING KEY (0 або 4 байт) – встановлюються, якщо напрям повідомлення з клієнта на сервер, тобто якщо поле «MASK» встановлено в «1». Клієнт заповнює це поле випадковими даними, причому змінюючи їх під час кожного наступного пересилання. Цей ключ повинен генеруватись джерелом з високим рівнем ентропії [6].

PAYLOAD DATA (x + y) складається з додаткових (extension data) та основних (application data) даних. Довжина додаткових даних дорівнює нулю, якщо не визначено інакше.

Крім фреймів, які містять корисну інформацію, визначені й контрольні фрейми, призначення котрих – обмін статусами між точками комунікації. Їхня довжина завжди дорівнює одному фрейму.

Основна мета використання такого формату фреймів – дозволити передавати повідомлення з невідомим наперед розміром, тоді не потрібно буферизувати повідомлення перед надсиланням чи після прийому.

Реалізація клієнтської частини. Для використання веб-сокетів на стороні клієнта має бути встановлене спеціальне програмне забезпечення. Як правило, це браузер з відповідною підтримкою.

Насамперед, слід створити об'єкт WebSocket з необхідною url-адресою (обов'язковий параметр). Додатково можна встановити один або декілька підпротоколів. У випадку, якщо порт, до якого здійснюється спроба під'єднатись, заблокований, конструктор генерує виключну ситуацію SECURITY_ERR.

Об'єкт може перебувати у чотирьох станах: підключення (connecting), відкритий (open), закриття (closing), закритий (closed). Події, які об'єкт може обробляти: onopen, onerror, onclose, onmessage.

Приклад створення:

```
var socket = new WebSocket('ws://localhost:8080/');
```

У разі помилки, під час спроби встановлення з'єднання, генерується подія "error", що надсилається WebSocket об'єкта (викликається onerror обробник), після цього тому ж об'єкту надсилається подія "close" і, відповідно, спрацьовує onclose обробник.

У разі успішного відкриття можна виконувати обмін даними. Для пересилання даних на сервер використовується метод send:

```
socket.sent("Message from client");
```

Оскільки встановлене з'єднання є асинхронним, немає жодної гарантії, що надсилання даних відбудеться безпосередньо після створення об'єкта WebSocket. Щоб запобігти ситуації, в якій повідомлення може загубитись, їх потрібно розмістити у блоці обробника, що спрацьовує після настання події onopen, що генерується, як тільки з'єднання уже готове:

```
socket.onopen = function(event) {
     socket.send("Message from client");
}
```

Оскільки зв'язок асинхронний, прикладний програмний інтерфейс веб-сокетів побудований на подіях. Отже, коли від сервера приходить повідомлення, одразу генерується подія onmessage, що виконує відповідний набір команд. Приклад обробника:

```
socket.onmessage = function(data) {
     switch (data.type) {
           case 0:
                 processNormalMessage();
                 break;
           case 1:
                 processPrivateMessage();
                 break;
           case 2:
                 processSystemMessage();
                 break;
           default:
                 break;
};
```

Аналогічний підхід необхідно використовувати і для подій onerror та onclose.

Реалізація серверної частини. Серверна частина являє собою веб-програму, яка прослуховує визначений порт.

У цій задачі слід виділити дві окремі сутності: власне WebSocktServer, що відповідатиме за опрацювання клієнтських з'єднань та їх ініціалізацію, та WebSocketConnection, яка представляє окремі з'єднання (рис. 8).

Сервер очікує на з'єднання і, після надходження, створює відповідний об'єкт, що надалі з ним асоціюватиметься та через котрий проводитимуть операції надсилання чи приймання даних. Створюючи об'єкт – сервер, йому слід визначити такі параметри: порт, адресу прослуховування та адресу розміщення сервера:

```
public WebSocketServer(int port, String origin, String location)
```

Основні поля сервера – набір встановлених з'єднань та сокет, через який відбувається комунікація. Серед методів достатньо визначити такі: ініціалізація сервера, надсилання повідомлення окремому клієнту (тобто окремому об'єкту, що представляє з'єднання), надсилання повідомлення усім приєднаним клієнтам. Єдина подія, яку повинен обробляти сервер, – встановлення з'єднання з черговим клієнтом, що має оброблятись в режимі зворотного виклику.

Для сутності WebSocketConnection варто визначити такі події: отримання даних та розрив з'єднання. Необхідні поля – сокет, через котрий відбуватиметься взаємодія, та унікальний ідентифі-

катор, щоб мати змогу розрізняти та доступатись до окремих сутностей. Обов'язкові операції – закриття з'єднання та надсилання даних (рис. 8).

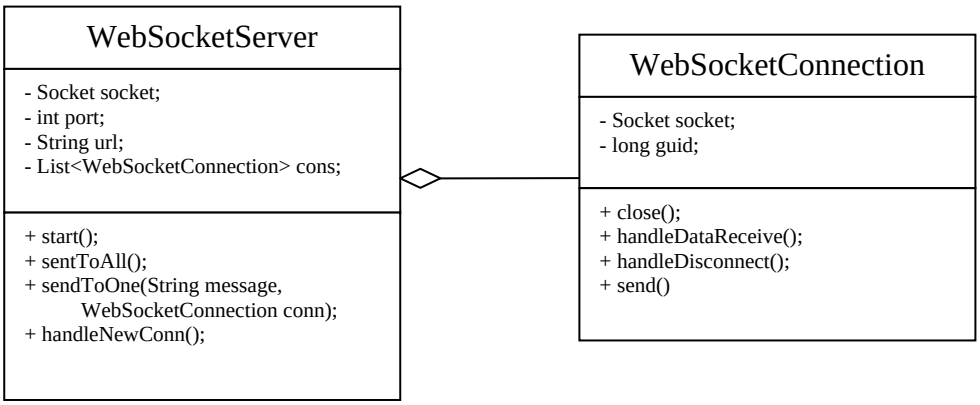


Рис. 8. Схема відношення сутностей серверної частини реалізації веб-сокетів

Порівняння підходу до асинхронної взаємодії з використанням веб-сокетів та черги опитування. Розглянемо обсяг службового трафіку (табл. 1), який генеруватиметься у випадку використання черги запитів очікування та веб-сокетів.

Припустимо, що упродовж п'яти секунд, з інтервалом в одну секунду, відбувається обмін даними. З них три у напрямку сервер-клієнт та два у зворотному.

Службовим вважатимемо трафік, що витрачається на безпосереднє встановлення та розпізнавання з'єднання браузером та сервером. У випадку черги запитів опитування (ЧЗО) це будуть заголовки з рядком типу запиту (320 байтів та 316 байтів відповідно). У випадку веб-сокетів – handshake-запит (87 байтів) та відповідь (89 байтів), для встановлення з'єднання та два байти у разі пересилання у напрямку з сервера на клієнт і чотири – з клієнта на сервер.

Приклад розрахунку службового трафіку при обслуговуванні тисячі клієнтів:

ЧЗО: *(320 + 316 + 320 + 316 + 320) * 1000 = 1592000 байт = 1,518МБ*
Веб-сокети: *(87 + 89 + 4 + 2 + 4) * 1000 =186000 байт = 0,177МБ.*

Для 10 000 та для 100 000 клієнтів підрахунок аналогічний, хоч різниться кількість клієнтів. Отримані результати наведено в табл. 1.

Таблиця 1

**Порівняння кількості службової інформації,
що генерується під час обслуговування клієнтів**

Технологія	Кількість клієнтів		
	1 000	10 000	100 000
Черга запитів очікування	1,518МБ	15,182МБ	151,824МБ
Веб-сокети	0,177МБ	1,773МБ	17,732МБ

Розділивши отримані значення для останнього випадку, одержимо частку 8,56. Це значення показує, у скільки разів менша кількість службового трафіку при використанні веб-сокетів порівняно з чергою запитів очікування.

Основна причина, через яку поки що не використовують веб-сокети, – не усі веб-браузери однаково добре підтримують цю технологію. Так, наприклад, Internet Explorer підтримує цю технологію лише в останній версії (10), в Opera за замовчуванням вимкнена опція підтримки веб-сокетів. Також на сьогодні немає стабільних версій веб-серверів з підтримкою веб-сокетів.

Узагальнене якісне порівняння черги запитів очікування та веб-сокетів (табл. 2) за основними параметрами [4, 5, 6] показує, що через деякий час веб-сокети стануть провідною технологією для асинхронної клієнт-серверної взаємодії у режимі реального часу.

Таблиця 2.

Порівняння характеристик черги запитів очікування та веб-сокетів

Характеристика	Черга запитів очікування	Веб-сокети
Підтримка браузерами	Підтримується більшістю сучасних веб-браузерів	Підтримка усіма останніми версіями основних браузерів(IE, Google Chrom, Firefox, Opera). Не підтримується більшістю мобільних браузерів
Обчислювальне навантаження на сервер	Використовує невелику кількість процесорного часу, але холостий процес для окремого клієнта, споживає пам'ять	Обслуговує запити у міру їх надходження. Процесорний час та пам'ять використовуються лише у відповідь на дію клієнта
Обчислювальне навантаження на клієнта	Залежить від способу реалізації	Реалізовано на рівні браузера. Потребує мінімуму ресурсів
Інтерактивність	Невелика затримка при отриманні відповіді та повторному формуванні запиту	Взаємодія у реальному часі
Складність реалізації	Відносно легка реалізація, за допомогою стандартних засобів серверного програмування	Має підтримуватись на рівні веб-сервера

Висновки

У статті проаналізовано сучасні підходи до реалізації асинхронної клієнт-серверної взаємодії веб-програм. Розглянуто основні алгоритми двонапрямленої взаємодії між клієнтом та сервером. Здійснено огляд шляхів обходу традиційної архітектури «клієнт-сервер» у контексті веб-програм, їх тонкощі та недоліки. Запропоновано підхід застосування веб-сокетів для асинхронної взаємодії компонентів веб-програм. Наведено порівняння службового трафіку та інших основних характеристик черги запитів очікування і веб-сокетів.

1. Berners-Lee T., Cailliau R., WorldWideWeb: Proposal for a HyperText Project - Tech, rep., CERN, 1990. 18 p. 2. Jesse James Garrett. Ajax: A new Approach to Web Application. AdaptivePath, 44p. 3. Specification of the XMLHttpRequest object from the Level 1 W3C Working Draft released on April 5th, 2006. W3.org. 67 p. 4. Vanessa Wang, Frank Salim, Marcelo Jabali, The Definitive Guide to HTML5 - WebSocket, 2012, 200p. 5. Pener Lubbers, Brian Alers, Frank Sailm, Pro HTML5 Programming. Second edition. – Apress, 2011. 353 p. 6. I. Fette, Google, Inc., A. Melnikov, Isode Ltd., The Websocket Protocol V.17., Internet Engineering Task Force (IETF), 72 p. 7. Julian A. Richter, Websockets Paper - RWTH Aachen University, Germany, 2011. 31 p. 8. Engin Bozdag, Ali Mesbah, Arie van Deursen, A Comparison of Push and Pull Techniques for AJAX - Delft University of Technology, Software Engineering Research Group, Technical Report Series, The Netherlands, 2009. 12 p.