UDK 004.8 DOi 10.30837/ bi.2020.2(95).04

І.В. Кириченко [1], А. В. Назаренко [2], Р. О. Попов [3]

[1] к.т.н., ст.викладач кафедри програмної інженерії,
Харківський національний університет радіоелектроніки, Україна,
iryna.kyrychenko@nure.ua, ORCID iD: 0000-0002-7686-6439

[2] студент кафедри програмної інженерії,
Харківський національний університет радіоелектроніки, Україна,
anton.nazarenko@nure.ua

[3] студент кафедри програмної інженерії,
Харківський національний університет радіоелектроніки, Україна,
rostyslav.popov@nure.ua

ОПТИМІЗАЦІЯ ТА МАСШТАБУВАННЯ NODE.JS ДОДАТКІВ

Кириченко І.В., Назаренко А.В., Попов Р.О. Оптимізація та масштабування Node.js додатків. Актуальність цієї роботи зумовлена тим, що Node.js швидко стає однією з найпопулярніших платформ для створення швидких, масштабованих веб та мобільних додатків. Опитування користувачів Node.js 2017 року показує, що в даний час в Інтернеті перебуває понад 7 мільйонів екземплярів Node.js, причому кожен четвертий користувач планує збільшити використання Node.js протягом наступних 12 місяців. І легко зрозуміти, чому 68 відсотків цих користувачів кажуть, що Node.js покращує продуктивність розробників, 58 повідомляє, що зменшує витрати на розробку, а 50 відсотків кажуть, що підвищує продуктивність додатків. Оскільки Node.js все частіше стає технологією, що обирається для розробки додатків, попит на досвідчених розробників Node.js також буде продовжувати зростати. У роботі розглянуто проблему масштабування та оптимізації Node.js додатків. Запропоновано декілька підходів розробки Node.js додатків, які допоможуть оптимізувати швидкість виконання програмного коду та розробити рішення, що буде легко масштабуватися.

ОПТИМІЗАЦІЯ, NODE.JS, МАСШТАБУВАННЯ, V8 GARBAGE COLLECTION, ПРОДУКТИВНІСТЬ

Кириченко И.В., Назаренко А.В., Попов Р.А. Оптимизация и масштабирование Node.js приложений. Актуальность данной работы обусловлена тем, что Node.js быстро становится одной из самых популярных платформ для создания быстрых, масштабируемых веб-сайтов и мобильных приложений. Опрос пользователей Node.js 2017 показывает, что в настоящее время в интернете пересматривается более 7 миллионов экземпляров Node.js, а также каждый четвертый пользователь планирует увеличить использование Node.js за 12 месяцев. Легко понять, почему 68 процентов этих пользователей говорят, что Node.js улучшает производительность разработчиков, 58 сообщает, что уменьшает затраты на разработку, и 50 процентов говорят, что увеличивает производительность приложений. Несмотря на то, что Node.js все чаще становится распространенной технологией для разработки приложений, спрос на опытных разработчиков Node.js также будет продолжено расти. В работе рассмотрена проблема масштабирования и оптимизации Node.js приложений. Предложено несколько подходов разработки Node.js приложений, которые помогут оптимизировать скорость выполнения программного кода и разработать решение, что будет легко масштабироваться.

ОПТИМИЗАЦИЯ, NODE.JS, МАСШТАБИРОВАНИЕ, V8 GARBAGE COLLECTION, ПРОИЗВОДИТЕЛЬНОСТЬ

Kyrychenko I., Nazarenko A., Popov R. Optimization and scaling Node.js apps. The relevance of this work is due to the fact that Node.js is rapidly becoming one of the most popular platforms for building fast, scalable web and mobile applications. In fact, the 2017 Node.js User Survey reveals that there are currently over 7 million Node.js instances online, with three in four users planning to increase their use of Node.js in the next 12 months. And it's easy to see why: 68 percent of those users say Node.js improves developer productivity, 58 report it reduces development costs, and 50 percent say it increases application performance. As Node.js increasingly becomes the preferred technology for application development, the demand for expert Node.js developers will also continue to increase. The paper considers the problem of scaling and optimization of Node.js applications. Several approaches to developing Node.js applications have been proposed to help optimize code execution speed and develop solutions that are easy to scale.

OPTIMIZATION, NODE.JS, SCALING, V8 GARBAGE COLLECTION, PERFORMANCE

Вступ

Багато написано про те, що можна зробити за допомогою Node.js, як розробники можуть почати використовувати цю платформу, і чому вона стала основною технологією на стороні сервера з деякими найбільшими корпораціями світу, але мало що написано для допомоги новачкам або середнім розробникам Node.js підняти свої навички на новий рівень.

Типовий респондент використовує Node трохи більше 2 років і проводить більше половини свого часу на розробці Node. Досвід роботи з Node.js можемо побачити на рис. 1.

Враховувати продуктивність серверу, доступність та швидкість виконання програмного коду на ньому в момент проектування архітектури — дуже важливо. Чим раніше буде розуміння того, яке навантаження

буде на сервер — тим простіше буде у подальшій розробці та підтримці проекту. Покращувати роботу Node.js серверу можна багатьма шляхами, далі будуть приведені основні методи того, як збільшити кількість запросів, що може обробити сервер за певну одиницю часу.

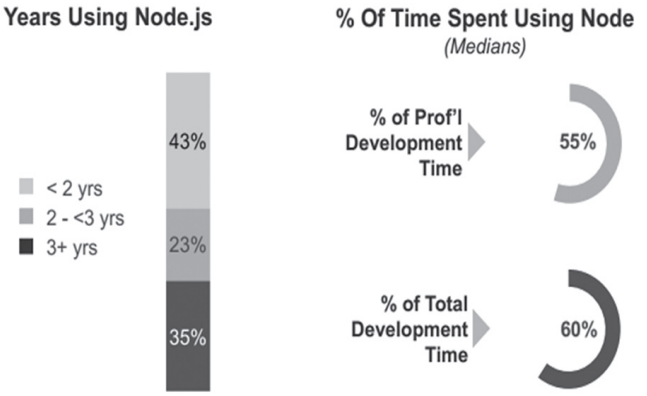

Рисунок 1. Досвід роботи з Node.js
Джерело: https://nodejs.org/en/user-survey-report/

У цій статті будуть наведені стратегії, які можуть призвести до несподіваних проблем у довгостроковій перспективі, але, якщо їх правильно впровадити, це значною мірою підвищить ефективність додатків.

1. Підготовка до запуску виробництва

Підготовка проекту завжди є критичним моментом у будь-якому шляху розробки додатків, і це, безумовно, стосується проектів Node.js. Це остання можливість команди знайти та виправити проблеми, перш ніж вони вплинуть на процес розгортання, ваших кінцевих користувачів або сам бізнес.

Тому слід звернути увагу на наступні пункти:

– оптимізація коду;
– найкращі практики щодо обробки помилок;
– підтвердження коду відповідає вимогам безпеки;
– налаштування для виробничого середовища;
– використання одного пакетного менеджеру
– міркування щодо розгортання.

Що стосується оптимізації коду, то однією з багатьох найкращих практик перед виробництвом є процес, який називається "підключення". Це передбачає запуск автоматизованого інструменту якості коду, такого як ESLint або JShint, через вашу кодову базу. Зазвичай він охоплює лише дуже основні питання якості, але в цьому полягає суть: він виявляє помилки, яких можна уникнути – і, як правило, їх дуже легко виправити – до того, як вони загрожують виробничій програмі.

Якщо порівняємо популярність пакетних менеджерів, то побачимо, що найпопулярніший - NPM від розробників Node.js (рис. 2).

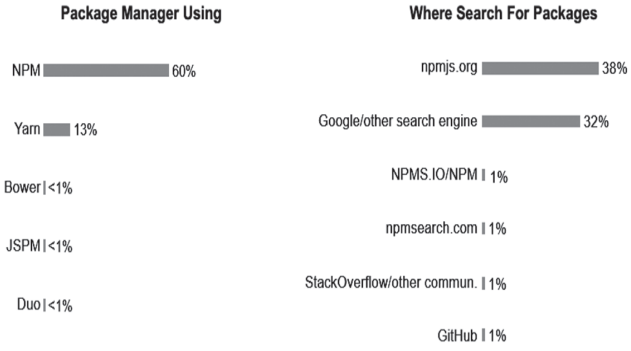

Рисунок 2. Використання менеджера пакетів
Джерело: https://nodejs.org/en/user-survey-report/

2. Розгортання програми

Охопивши основи ефективного робочого процесу перед виробництвом, далі розглянемо, чого очікувати і як реагувати протягом критичних перших годин після розгортання.

Розгортання корпоративної програми може бути важким. Приблизно до 30 відсотків усіх розгортань програм не вдається. Тим часом приблизно 80 відсотків організацій мають проблеми з випуском програмного забезпечення. Очевидно, що кожен, кому доручено розгортати програму, повинен бути готовий до того, що все піде не так – можливо, дуже неправильно.

Хоча надійний процес попереднього виробництва може допомогти мінімізувати вплив помилок конфігурації та інших проблем, яких можна уникнути, експерти-розробники Node.js повинні знати, як вирішувати загальні проблеми розгортання, особливо ті, що призводять до збоїв та проблем.

Типові проблеми, які можуть виникнути при розгортанні, включають:

– збій процесів Node.js;
– перевищення обмежень швидкості API;
– вирішення проблем із WebSocket;
– проблеми з залежностями;
– проблеми із завантаженням файлів;
– DDoS-атаки.

Хороша сторона проблем з розгортанням полягає в тому, що розробник дізнається багато нового про створення програм Node.js та про розгортання своїх програм із меншою кількістю проблем. Хоча проблеми можуть і будуть траплятися в майбутньому, справді серйозних проблем, ймовірно, буде менше.

Ще кращою стороною є, що як тільки ефективно усунуться основні проблеми розгортання, розробник матимете справу із більш стабільним та надійним додатком. Це, у свою чергу, звільняє від зосередження уваги на способах покращення продуктивності програми та модернізації власного процесу створення, тестування та розгортання програм Node.js.

Серед сервісів для деплою коду можна розглянути таких мастодонтів як AWS, Heroku, Digital Ocean. Більш детальніший список клауд платформ можна побачити на рис. 3.

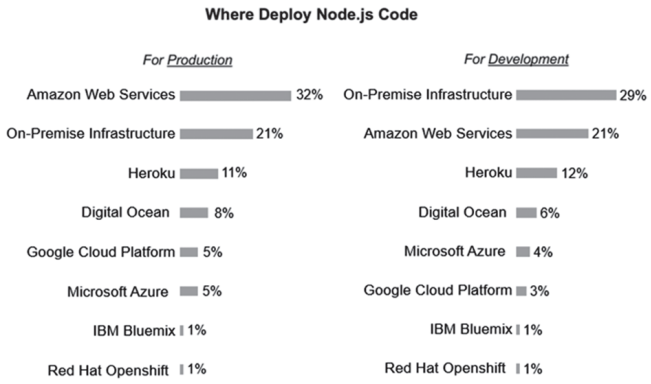

Рис. 3. Порівняння продуктивності версій Node.js
Джерело: https://nodejs.org/en/user-survey-report/

3. Управління

Після успішного розгортання слід звернути увагу на управління вашим додатком Node.js. Хоча це не надто відрізняється від будь-якого іншого поширення додатків, є кілька особливостей, на які слід звернути увагу:

– витоки пам'яті;
– керування паралельністю Node.js;
– моніторинг.

Моніторинг продуктивності додатків є життєве важливим для підтримання стабільності розгортання вашого додатка та виявлення тонких регресій, які можуть призвести до уповільнення роботи програми або відмови, якщо її не встановити.

4. Експеримент

Різниця версій Node.js.

Регулярне оновлення версій Node.js дуже необхідна річ, чим швидше відбувається оновлення на нову стабільну версію платформи Node.js, тим краще, в цілому, починає працювати сервер написаний з використанням Node.js.

Наприклад, різниця, в кількості запитів, що обробляються за секунду, між восьмою та дванадцятою версією складає понад 5 000 запитів. Тобто, лише оновлення з восьмої версії Node.js до дванадцятої дає вам приріст продуктивності сервера на 25-30 відсотків [1].

Побачити порівняння різних версій, та різницю їхньої продуктивності можна нижче на рис. 4.

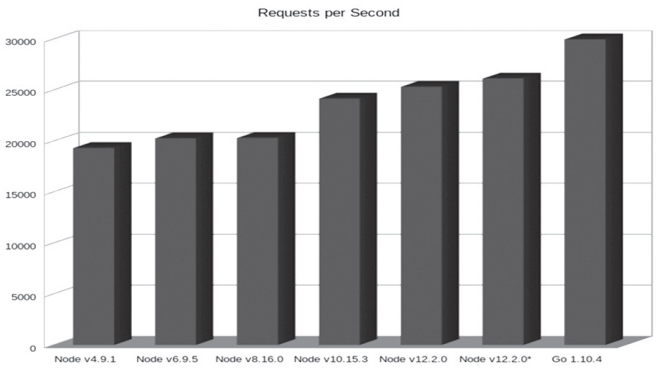

Рис. 4. Порівняння продуктивності версій Node.js
Джерело: складено авторами.

Також, після оновлення версії Node.js збільшиться й швидкість виконання вбудованих в V8 методів. Наприклад, різниця швидкості виконання методу find на V8 v7.0 відрізняється від V8 v7.1 майже на сто відсотків.

Різниця в швидкості виконання важкого асинхронного коду між Node.js 7 та Node.js 10 складає близько десяти разів. Для порівняння використовувалася одна й та сама функція з одними й тими ж параметрами, Node.js 7 виконала її за 290 мілісекунд, в той час, як Node.js 10 виконала її за 30 мілісекунд. На цьому прикладі чітко видно, що оновлення версії Node.js є дуже важливим та значно підвищує швидкість роботи вашого серверу, а значить й дає можливість обслуговувати більшу кількість користувачів.

На рис. 5 наведено приклади різниці у швидкості виконання однакового асинхронного коду на різних версіях Node.js.

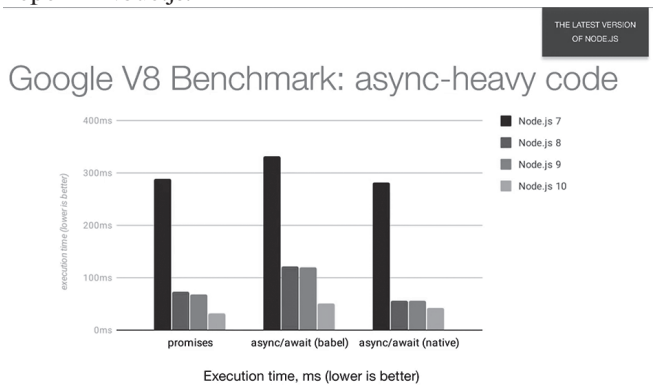

Рис. 5. Порівняння виконання асинхронного коду
Джерело: складено авторами

Вибір пакетів Node.js.

Не завжди вбудовані в V8 методи є оптимальними с точки зору їх продуктивності та швидкості виконання. Наприклад, метод stringify, котрий вбудовано в V8 у багатьох випадках є менш продуктивним, аніж аналогічні методи із пакетів, що можна завантажити за допомогою node package manager (npm). Нижче, на рис. 6 наведено порівняння популярних npm модулів та стандартного JSON.stringify.

На даному рисунку чітко видно, що для деяких випадків, методи зі сторонніх пакетів відпрацьовують у дев'ять разів швидше. Через те, що даний метод дуже часто використовується при розробці серверів з використанням Node.js, то навіть заміна лише його на метод з зовнішньої бібліотеки може суттєво прискорити час відповіді енд-поентів, що використовують stringify.

Наступним прикладом вдалого вибору є бібліотеки Socket.IO та µWebSockets, якщо їх порівняти з вбудованим WS модулем в Node.js, то можна побачити, що найбільш популярний Socket.IO є найменш продуктивнішим, цей модуль обробляє приблизно 8 000 повідомлень, вбудований WS модуль – 35 000, а µWS – понад 200 000, тому що цей модуль повністю написано на C++ [2]. Фреймворк µWS має

кращу пропускну здатність коротких повідомлень, яка більша у 23 рази, у 21 раз кращу продуктивність з'єднання та у 52 рази менше використання пам'яті в порівнянні з вбудованим WS модулем.

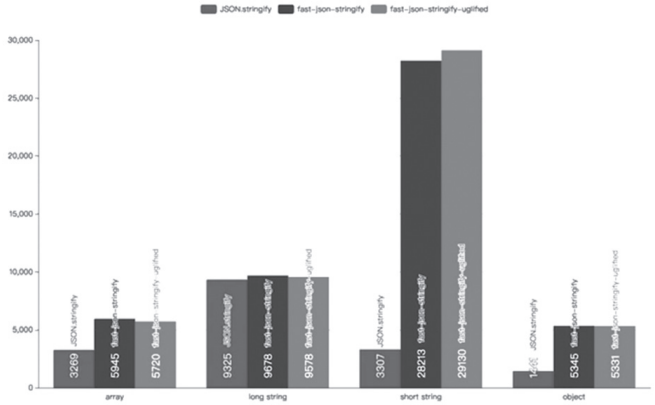

Рис. 6. Порівняння stringify методу

Джерело: складено авторами

Порівняння найпопулярніших фреймворків можна побачити на рис. 7.

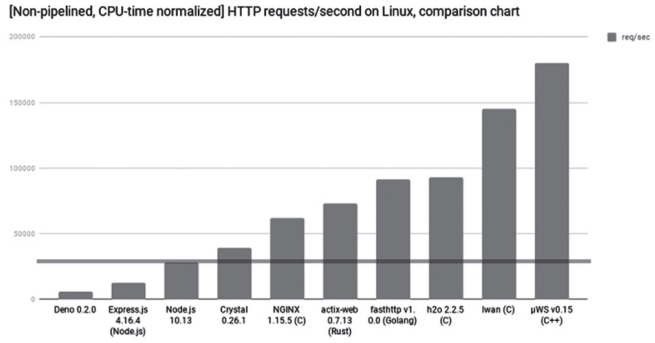

Рис. 7. Порівняння WS фреймворків

Джерело: складено авторами

Тому, якщо у додатку часто оновлюється інформація, наприклад, це крипто біржа чи тренінги літаків у реальному часі, має сенс використовувати такі високопродуктивні фреймворки, як µWS.

Інші методи.

Слід також зазначити, що є декілька методів зробити Node.js більш продуктивною:

– кластеризація Node.js додатку;
– використання інструментів для хешування;
– використання протоколу HTTP2;
– оптимізація V8 garbage collection.

Використання HTTP2 замість HTTP1.1 може призвести до збільшення швидкості обробки запиту у декілька разів [3]. Протестувавши один й той самий енд-поінт, але змінивши HTTP1.1 на HTTP2 вдалося досягти приросту більш ніж в 2 рази, завдяки більш досконалому і швидкому протоколу.

Також можна підвищити швидкість обробки запитів за допомогою хешування, використовуючи такі інструменти як Redis [4]. Використання даного інструменту дозволить зберігати певні дані в random access memory, що гарантує швидкий доступ до даних, аніж зробити вибірку з бази даних.

Кластеризація — дозволяє симулювати багатопоточність в Node.js за допомогою виконання додатку в кластерному режимі. Фактично, створюючи нові процеси з вашим додатком та балансуючи запити між усіма активними додатками за допомогою балансувальника. Це дозволить суттєво збільшити кількість запитів, що може обробити сервер [5].

Бажано, виносити важкі для CPU задачі з Node.js до веб серверу, наприклад, Nginx. Прикладі того, що можна винести з Node.js додатку до Nginx:

– HTTP/2;
– Gzip;
– SSL;
– роздачу статичних файлів;
– логування запитів та відповідей;
– метрики та моніторинг;
– авторизація.

Висновки

Було обрано кращі шляхи прискорення додатків, та збільшення їх продуктивності. Використовуючи усі техніки, що було приведено вище, можна дуже добре оптимізувати та прискорити будь-який сервер написаний на платформі Node.js. А якщо була правильно обрана архітектура для сервера, то в результаті повинен вийти додаток, який можна легко масштабувати та підтримувати на протязі довгого часу. Також, у роботі було показано, як можна ефективно використовувати потужності сервера, на якому виконується Node.js додаток.

Дані техніки без проблем імплементуються на будь-якому сервері, та призводять, у деяких випадках, до загального приросту швидкості, доступності та продуктивності серверу у десять разів, а інколи і більше.

Список літератури:

[1] *Young A., Meck B., Cantelon M.* Node.js in Action, Second Edition, 2017. 392 p.

[2] *Herron D.* Node.js Web Development: Server-side development with Node 10 made easy, 4th Edition, 2018. 494 p.

[3] *Dayley B., Dayley B.* Node.js, MongoDB and Angular Web Development: The definitive guide to using the MEAN stack to build web applications, 2017. 636 p.

[4] *Casciaro M., Mammino L.* Node.js Design Patterns: Design and implement production-grade Node.js applications using proven patterns and techniques, 3rd Edition, 2020. 660 p.

[5] *Pasquali S.* Mastering Node.js: Expert techniques for building fast servers and scalable, real-time network applications with minimal effort, 2013. 346 p.

[6] *Thota N.R.* Mastering Hyperledger Fabric: Master The Art of Hyperledger Fabric on docker, docker swarm and Kubernetes, 2020. 231 p.

[7] *Elliott E.* Programming JavaScript Applications: Robust Web Architecture with Node, HTML5, and Modern JS Libraries, 2014. 254 p.

[8] *Mead A.* Learning Node.js Development: Learn the fundamentals of Node.js, and deploy and test Node.js applications on the web, 2018. 658 p.

Надійшла до редколегії 12.11.2020