https://doi.org/10.31891/2219-9365-2024-80-20
УДК 621.396.969.1

ПАРХОМЕЙ Ігор
Державний університет «Київський авіаційний інститут»
https://orcid.org/0000-0002-9510-7657
e-mail: i_parhomey@ukr.net
БОЙКО Юлій
Хмельницький національний університет
https://orcid.org/0000-0003-0603-7827
e-mail: boiko_julius@ukr.net
ЛЕМЕШКО В'ячеслав
Державний університет «Київський авіаційний інститут»
https://orcid.org/0009-0008-1495-715X
e-mail: slava.lemeshko@gmail.com

АЛГОРИТМ НАЛАШТУВАННЯ КІЛЬКОСТІ ПОТОКІВ ДЛЯ ВИКОНАННЯ ФОНОВИХ ЗАДАЧ

Сучасні програмні системи часто виконують одночасно велику кількість фонових задач, що може призводити до значного навантаження на сервери, яке призведе до зниження продуктивності роботи користувачів із веб-додатком. Ефективне налаштування кількості потоків для виконання таких задач дозволяє оптимізувати використання ресурсів, підвищити швидкість обробки задач та забезпечити стабільність системи. Враховуючи це, розробка алгоритму для автоматичного або ручного налаштування кількості потоків є критично важливою для підтримання високої продуктивності та надійності програмного забезпечення, особливо в умовах зростання обсягів даних та складності обчислювальних процесів. У статті представлено платформу проектування та виконання бізнес-сервісів, яка складається з кількох підсистем для підтримки фонових процесів та масштабованої обробки запитів. Описано підхід до обробки запитів на основі потоків та подій, що забезпечує оптимальну продуктивність і гнучкість. Архітектура платформи базується на трьох основних шарах: презентаційному, логічному та шарі даних, які взаємодіють для ефективної обробки користувацьких запитів. Детально розглянуто підсистеми планувальника, виконання фонових операцій та бізнес-процесів, що дозволяють автоматизувати задачі, оптимізувати ресурси й інтегрувати систему із зовнішніми сервісами. Представлені рішення забезпечують масштабованість, зниження витрат на ресурси та покращення продуктивності веб-додатків.

Ключові слова: веб-сервіс, потік, запит, продуктивність, фонові операції, програмні системи

PARKHOMEY Igor
State University «Kyiv Aviation Institute»
BOIKO Juliy
Khmelnytskyi National University
LEMESHKO Viacheslav
State University «Kyiv Aviation Institute»

ALGORITHM FOR CONFIGURING THE NUMBER OF THREADS FOR BACKGROUND TASK EXECUTION

Modern software systems typically handle a significant number of background tasks simultaneously, which can lead to increased server load and reduced user experience performance with web services. Optimizing the number of threads for executing such tasks allows for efficient resource allocation, faster processing speeds, and enhanced system stability. In this context, developing an algorithm for automatic or manual adjustment of thread counts is essential for ensuring high performance and reliability of software, especially amid growing data volumes and increasing complexity of computational processes.

The rising number of users imposes constant pressure on business services, which must guarantee scalability, reliability, and prompt handling of concurrent requests. Contemporary servers face performance limitations not due to hardware constraints but rather due to the complexity of software systems, making the configuration and optimization of software solutions critically important.

The paper presents a platform for designing and executing business services, comprising several subsystems that support background processes and scalable request handling. It describes a stream- and event-based approach that ensures optimal performance and flexibility. The platform's architecture is built on three primary layers—presentation, logic, and data—which interact to process user requests efficiently.

The scheduler, background operations, and business process execution subsystems are examined in detail, showcasing their capabilities in automating tasks, optimizing resources, and integrating the system with external services. The proposed solutions enhance scalability, reduce resource consumption, and improve the performance of web applications.

The use of the proposed algorithm for configuring the number of background handlers reduced the number of repeated requests (e.g., "Update configuration files on the site") to the cloud service by 50%. Consequently, user complaints related to platform reconfiguration were halved. It was shown that incorrect configuration of the number of background handlers resulted in additional time spent analyzing and resolving issues after adjusting the settings. Such situations led to poor performance of business services or their complete shutdown.

Keywords: web service, stream, request, performance, background operations, software systems

ПОСТАНОВКА ПРОБЛЕМИ У ЗАГАЛЬНОМУ ВИГЛЯДІ ТА ЇЇ ЗВ'ЯЗОК ІЗ ВАЖЛИВИМИ НАУКОВИМИ ЧИ ПРАКТИЧНИМИ ЗАВДАННЯМИ

У сучасних умовах стрімкого зростання обсягу даних і складності обчислювальних процесів програмні системи стикаються з викликом забезпечення стабільної та високопродуктивної роботи. Одночасне виконання великої кількості фонових задач створює значне навантаження на сервери, що може призводити до зниження продуктивності, нестабільності роботи веб-сервісів та погіршення взаємодії з користувачами.

Оптимізація розподілу ресурсів, зокрема кількості потоків для обробки фонових задач, є важливим науковим та практичним завданням. Вона спрямована на підвищення швидкості обробки, забезпечення масштабованості та мінімізацію ризику зупинки бізнес-сервісів. Складність цього завдання полягає у необхідності створення алгоритмів, здатних автоматично або з мінімальним втручанням оператора адаптувати кількість потоків до змінних умов навантаження системи [1-3].

Розробка таких алгоритмів має вирішальне значення для забезпечення надійності та продуктивності програмного забезпечення в умовах підвищених вимог до масштабованості та обробки одночасних запитів. Крім того, врахування специфіки програмних рішень, де продуктивність обмежується не апаратним забезпеченням, а складністю їх внутрішньої архітектури, є важливим аспектом для оптимізації сучасних бізнес-сервісів.

Таким чином, мета цієї статті полягає у створенні ефективного підходу до динамічної адаптації кількості фонових потоків, що дозволить забезпечити високу продуктивність, знизити витрати на обчислювальні ресурси та покращити користувацький досвід взаємодії із системами.

АНАЛІЗ ДОСЛІДЖЕНЬ ТА ПУБЛІКАЦІЙ

Платформа автоматизації бізнес-сервісів [4, 5] реалізована із застосуванням веб-технологій та спроектована у триярусній архітектурі. Користувачі працюючи із веб-додатком формують запити з допомогою клієнтської частини, наприклад, веб-оглядача, які надсилаються до веб-сервера для подальшого опрацювання [6]. Запити, які повинні повернути результат в рамках відправленого запиту блокують роботу користувача із додатком на час виконання цього запиту [7]. Такі запити називаються інтерактивними або foreground-запитами. Для виконання кожного запиту, який надходить до веб-додатку виділяється потік, thread [8]. Окрім foreground-запитів, веб-додаток оброблює background-запити або їх ще називають фоновими операціями. Фонові операції у веб-додатках - це завдання, які виконуються на сервері без взаємодії із користувачем у реальному часі. Розглянемо декілька прикладів таких фонових операцій: відправлення сповіщень через різні канали (наприклад, SMS, push-повідомлення); отримання і обробка даних з інших систем, розрахунок нових рейтингів або рекомендацій для користувачів; генерація рахунків та квитанцій тощо. Виконання background-запитів також як foreground-запитів відбувається з допомогою потоків [9]. Таким чином виникає необхідність ефективного використання ресурсів веб-серверів, що мають обробляти значні навантаження в умовах зростання кількості фонових операцій. Зокрема, зважаючи на вартість оренди сучасних серверів, затримка у відповіді веб-серверів є критичним фактором для бізнес-сервісів, які потребують високої швидкості обробки запитів. На сервери припадає значна частина затримок у веб-транзакціях, тому покращення їхньої продуктивності є актуальною задачею.

Зі збільшенням кількості потоків зростають і витрати на координацію через суперництво за спільні ресурси. Це також стосується процесорів, адже коли кожен процесор повністю задіяний, додавання додаткових потоків збільшує витрати на координацію без збільшення кількості корисної роботи. Така ситуація називається перевантаженням ресурсів. Однак врахування лише загальної кількості корисної роботи не дає повної картини, тут важливо зосередитися також на затримці обробки запитів. Щоб врахувати компроміс між затримкою та пропускною здатністю, автори [10] пропонують враховувати фактор перевантаження для обчислення кількості активних потоків. Цей фактор вказує, на скільки кількість активних потоків має перевищувати кількість процесорів та призначений для налаштування під різні задачі.

Методи налаштування кількості потоків базуються на емпіричному способі. У роботі [11] запропоновано використання моделі зворотного зв'язку для динамічного розподілу потоків на різних етапах обробки запитів. Архітектура запропонована авторами веб-сервера розділяє обробку запиту на чотири фази: прийом з'єднання, отримання даних, обробка даних і відправлення відповіді. Для кожної з цих фаз створено власний пул потоків, і кількість потоків у кожній фазі може бути адаптивно скоригована залежно від навантаження сервера. Даний метод не враховує статистики оброблених запитів та використаних ресурсів. Також не враховує інших факторів, які впливають на визначення кількості потоків. Даний метод потребує проведення декількох експериментів, на що потрібен час. Динамічне виділення потоків призводить до частого створення та знищення потоків, що призводить до надмірного використання ресурсів серверу на підтримку таких технічних операцій [12, 13, 14].

Опис платформи

Для реалізації бізнес-процесів, які виконуються у фоновому режимі платформа проектування та виконання бізнес-сервісів, яка розглядається в даній роботі, складається із кількох підсистем, кожна з них використовує різні інформаційні технології та має окремі налаштування та оперує різними поняттями. Дана платформа використовує існуючі підходи на основі потоків і подій для забезпечення високої масштабованості та обробки одночасних запитів у веб-сервісах.

Підхід на основі потоків, коли кожен вхідний запит асоціюється з окремим потоком. Це дозволяє легко організувати паралельність, але призводить до високих витрат на переключення контексту, що обмежує продуктивність.

Підхід на основі подій використовує лише один потік, який обробляє всі події в циклі, що значно знижує витрати на ресурси та покращує масштабованість. Але цей підхід ускладнює код через так званий “callback soup” (безліч вкладених зворотних викликів) і потребує ретельного управління станом. Більш детальне порівняння цих підходів проаналізовано в роботі [15]. На рис. 1 представлено схему підсистеми обробки задач.

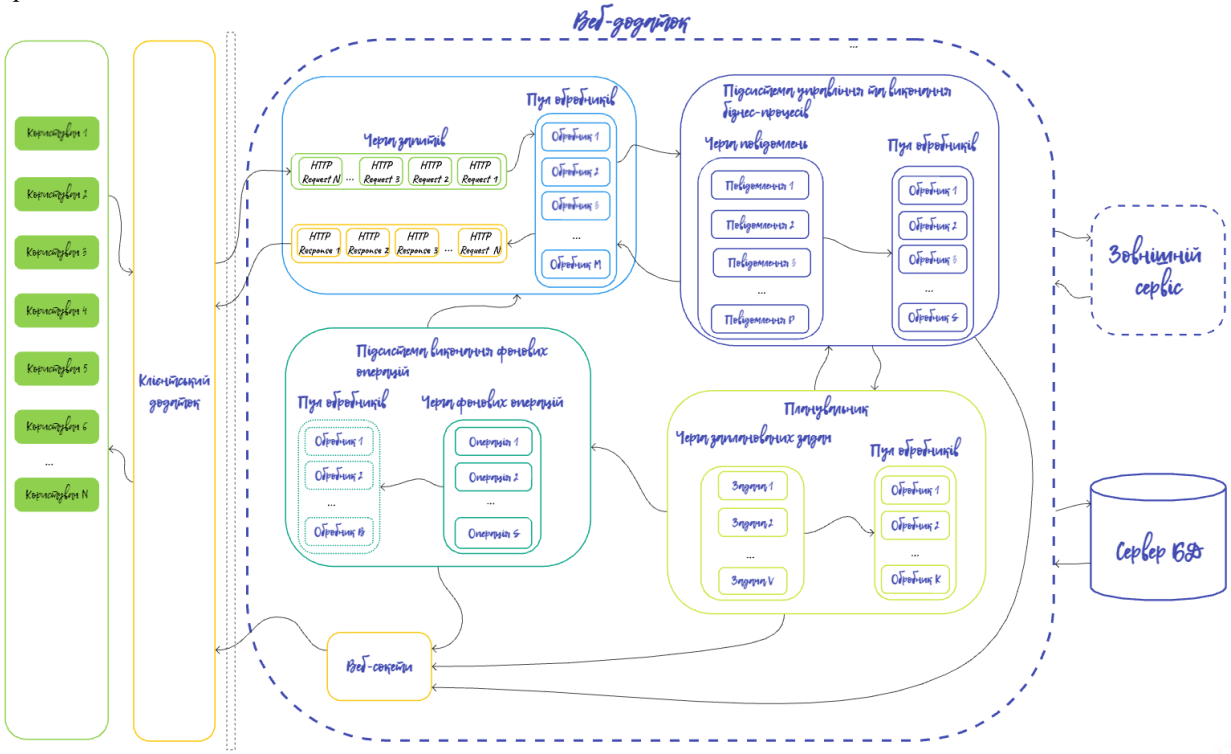

Рис.1. Загальна схема підсистема обробки задач

Розглянемо підсистему обробки запитів від користувача. Підсистема обробки запитів від користувачів у веб-додатках, розділяє обробку на три основні шари: презентаційний шар (клієнтський), логічний шар (серверний) та шар даних (база даних). Кожен із них виконує свою специфічну роль в обробці запитів, що дозволяє поліпшити масштабованість, гнучкість і розділення відповідальності в додатку.

Розглянемо презентаційний шар (клієнтський). Цей шар відповідає за взаємодію з користувачем. Він містить інтерфейс, з яким працює користувач (веб-сторінки, форми, елементи UI). Основними завданнями шару є формування запиту, відправлення запиту та обробка відповіді. Користувач виконує певну дію (натискання кнопки, заповнення форми), після чого клієнтська сторона формує HTTP-запит (GET, POST тощо). Запит надсилається через мережу на серверний шар для подальшої обробки. Після отримання відповіді від серверу, клієнтський шар обробляє дані (наприклад, оновлення інтерфейсу) та відображає їх користувачу.

Опишем особливості логічного шару (серверного). Цей шар виконує основну бізнес-логіку та обробку даних. Він приймає запит від клієнтського шару, обробляє його та взаємодіє з базою даних для отримання або збереження інформації. До основних етапів обробки запитів можна віднести отримання запиту, валідація та бізнес-логіка, взаємодія з базою даних, формування відповіді. Запит передається сервером через контролери (в архітектурі MVC це "Controller"). Сервер перевіряє коректність запиту, валідує дані та виконує необхідні бізнес-операції (обчислення, перевірки правил тощо). Логічний шар надсилає запит до бази даних через шар даних, щоб отримати, змінити чи видалити дані. Після обробки сервер формує відповідь (JSON, XML, HTML), яку потім передає назад клієнту.

Розглянемо шар даних (база даних). Цей шар відповідає за зберігання та доступ до даних. Він отримує запити від логічного шару та повертає відповідні дані або записує нові. До основні етапів даного шару можна віднести прийняття запиту, обробка запиту, повернення результату. Отримання *SQL*-запитів або використання ORM для взаємодії з базою даних. Виконання операцій над даними (читання, запис, оновлення чи видалення). Надсилання результатів запиту (наприклад, вибірка даних) назад на логічний шар.

Розглянемо як працює підсистема в цілому. Користувач взаємодіє з інтерфейсом (клієнтський шар), надсилаючи запит до серверу. Сервер приймає запит, виділяє для нього потік із пулу потоків, валідує його, виконує необхідну бізнес-логіку та запитує дані у бази. База даних надає необхідну інформацію, після чого сервер формує відповідь. Клієнт отримує відповідь від серверу, відображаючи її користувачу у зручному вигляді. Така триярусна архітектура дає можливість легше керувати та масштабувати додаток, розділяючи відповідальність між компонентами.

Підсистема планування завдань у програмному забезпеченні загалом працює як модуль або компонент, що керує автоматизованим запуском бізнес-процесів або завдань за попередньо визначеним розкладом. Ця система дозволяє запускати певні завдання (сценарії, функції, операції) автоматично без необхідності участі людини. Вона забезпечує виконання періодичних дій, таких як резервне копіювання, очищення даних, обробка великих масивів інформації тощо. Підсистема складається з таких основних компонентів: завдання, тригер, планувальника, сховища завдань і реалізована з допомогою Quartz. NET [16].

Завдання (Job) - це одиниця роботи, яку треба виконати. Завданням може бути будь-яка дія, така як запуск сценарію, обробка даних або завантаження файлу. Тригер (Trigger) визначає, коли і як часто завдання повинно виконуватися. Це може бути конкретна дата, час або періодичний інтервал (щодня, щотижня, кожну годину тощо). Планувальник (Scheduler) – це основний механізм, який координує виконання завдань згідно з їхніми тригерами. Він відстежує активні завдання і відповідає за їх запуск вчасно. Сховище завдань (Job Store) –це місце, де зберігаються дані про всі завдання та тригери (у пам'яті або в базі даних), що дозволяє їх зберігати і відновлювати.

Тепер розглянемо як працює система в цілому. Спочатку створюється завдання, яке визначає, що саме має виконуватися. Це може бути окремий процес, API-запит, сценарій, або функція, яка виконує певну дію. Завдання може включати логіку для обробки даних, взаємодії з іншими сервісами, створення звітів, відправки повідомлень тощо [17]. Після створення завдання його потрібно зв'язати з тригером, який визначає, коли завдання буде виконуватися. Тригер може бути одноразовим, коли завдання виконується один раз у визначений час або періодичним, коли завдання виконується через певний інтервал часу, наприклад, щогодини, щодня або щотижня. Планувальник моніторить активні завдання і тригери, які потрібно виконати, і керує запуском завдань відповідно до визначеного розкладу. Планувальник також враховує залежності між завданнями, перевіряє, чи є вільні потоки із власного пулу потоків для їх виконання, і може запускати завдання паралельно. Коли тригер спрацьовує, планувальник передає керування завдання підсистемі виконання фонових операцій, яка ставить в чергу дане завдання або запускає і виконує логіку завдання в залежності від вільних потоків. Завдання може взаємодіяти з іншими сервісами, API, базами даних або просто обробляти локальні дані.

Перейдемо до підсистеми виконання фонових операцій. Підсистема виконання фонових операцій платформи виконання та проектування бізнес-сервісів є важливою частиною системи, яка дозволяє виконувати довготривалі або ресурсомісткі задачі без блокування основного користувацького інтерфейсу. Це дозволяє оптимізувати продуктивність веб-додатку, надаючи користувачам змогу продовжувати роботу з системою, поки фонова задача виконується окремо. Підсистема приймає на виконання завдання від підсистеми планувальника та приймає на виконання фонові-процеси підсистеми виконання бізнес-процесів. Фонові процеси - це серверні операції, які виконуються незалежно від активності користувача. Вони можуть бути ініційовані в результаті певних подій (наприклад, запуск бізнес-процесів, обробка даних, генерація звітів тощо). Процеси створюються за допомогою бізнес-логіки або інтегрованих механізмів планування. Часто використовуються для таких задач, як масове оновлення записів, імпорт даних, розсилка електронних листів тощо. Черги використовуються для управління фоновими операціями. Вони дозволяють сортувати задачі за пріоритетом, часом виконання або іншими критеріями. В черзі завдання можуть бути поставлені на очікування, поки не з'являться необхідні ресурси для їх виконання. Фонові процеси використовують асинхронну модель для уникнення блокувань основного потоку додатку. Це означає, що коли процес запускається, він не чекає завершення перед виконанням інших дій.

Опишемо підсистему виконання бізнес-процесів. Підсистема бізнес-процесів є одним із ключових компонентів платформи, який дозволяє компаніям автоматизувати, моделювати та оптимізувати свої бізнес-процеси. Вона надає можливість організувати послідовність завдань, дій та подій у вигляді графічних схем, що дає користувачам зручний інструмент для управління робочими процесами без необхідності глибоких технічних знань. Підсистема бізнес-процесів дозволяє інтегрувати з іншими зовнішніми системами через веб-сервіси, API, або через стандартні інтеграційні інструменти платформи. Це дозволяє бізнес-процесам автоматично взаємодіяти з сторонніми додатками для обміну даними або запуску зовнішніх процесів.

Процес може бути запущений вручну користувачем, автоматично за допомогою тригерів (наприклад, при зміні даних в системі), або за допомогою планувальника. Під час виконання процес автоматично виконує вказані дії, надсилає сповіщення або запускає зовнішні операції. Після завершення процесу система реєструє його результати, а також дозволяє переглядати журнал виконаних дій для подальшого аналізу. Підсистема бізнес-процесів взаємодії із іншими вказаними підсистемами платформи.

Розглянемо налаштування кількості одночасно виконуваних фонових операцій для всіх підсистем, реалізованих у платформі (конфігурування). Оскільки ресурси (ЦП, оперативна пам’ять, операції вводу/виведення) сервера, на якому розміщено веб-додаток, обмежені, варто розглянути всі налаштування разом. Щоб зменшити час очікування фонових операцій у черзі, можна збільшити параметр, який відповідає за кількість одночасних фонових операцій (Табл. 1). Знаходження значення цього параметра має важливе значення для досягнення оптимальної продуктивності та забезпечення того, що бізнес-сервіси можуть ефективно виконувати заплановані завдання.

Таблиця 1

Конфігурування кількості одночасно виконуваних фонових операцій для всіх підсистем

Назва підсистеми	Файл конфігурування	Назва параметру	Опис параметру
Підсистема планувальника	*Web.config*	*quartz.threadPool.threadCount*	Визначає кількість потоків у пулі потоків, які використовуються для виконання завдань (*jobs*). Його основна функція полягає в тому, щоб обмежити кількість одночасно виконуваних завдань, що можуть бути оброблені планувальником. Якщо є десятки тисяч завдань, багато з яких запускаються щохвилини, то, ймовірно, потрібна кількість потоків, схожа на 50 або 100 (це сильно залежить від характеру роботи, яку виконують завдання, і системних ресурсів).
Підсистема виконання фонових операцій	*Web.config*	*nr-of-instances*	Параметр використовується в контексті конфігурації акторів у фреймворку *Akka .NET* і визначає кількість екземплярів актора, які будуть створені в рамках певного маршрутизатору. Його призначення полягає в забезпеченні масштабованості та підвищенні продуктивності шляхом одночасного виконання кількох екземплярів одного й того ж актора.
Підсистема виконання бізнес-процесі	*Web.config*	*messageProcessingThreadsCount*	Параметр використовується в системах, що працюють з чергами повідомлень або асинхронною обробкою повідомлень. Він визначає кількість потоків (*threads*), які будуть використовуватись для одночасної обробки повідомлень.

Отже всі параметри налаштовуються в одному і тому ж файлі. Збереження даного файлу призводить до перезапуску всього веб-додатку. Тому бізнес-сервіси будуть не доступні на протязі усього часу поки додаток перезапускається. Тому кількість змін налаштувань даного файлу має бути мінімізована. Це обмеження можна уникнути, реорганізувавши спосіб зберігання налаштувань, а також із допомогою автоматичної актуалізації нових значень із бази даних конфігурування. Але через дотримання правил безпеки, зміна значень відбувається з допомогою заявок до команди відповідальної за хмарні сервіси. Кількість спроб зміни налаштувань має лишатися мінімальною.

Опишемо процес визначення значення кількості фонових обробників. Щоб визначити правильні значення всіх цих параметрів, необхідно врахувати різні фактори. Адже збільшення значення цих параметрів може спричинити зависання виконання операцій, перезапуск веб-додатку через надмірне використання ресурсів, а також взаємні блокування запитів на рівні бази даних. З іншого боку, зменшення значення цих параметрів призведе до уповільнення виконання фонових операцій, значних затримок в обробці фонових операцій, а також зростання розміру черг із надмірним споживанням пам’яті, не кажучи вже про можливість втрати таких повідомлень після перезапуску веб-додатку.

Відомим є результат, що зі збільшенням кількості потоків зростають і витрати на координацію через суперництво за спільні ресурси. Це також стосується процесорів, адже коли кожен процесор повністю задіяний, додавання додаткових потоків збільшує витрати на координацію без збільшення кількості корисної роботи. Така ситуація називається перевантаженням ресурсів (overcommitment) [10]. Але врахування лише загальної кількості корисної роботи не дає повної картини, тут важливо зосередитися також на затримці.

Оскільки є кілька підсистем і всі вони використовують різні поняття, наприклад: актор (Actor), споживач (Consumer), фоновий працівник (Background worker), потік (thread), надалі пропонується використовувати єдине поняття “обробник”. Це необхідно для узагальнення та спрощення подальшого аналізу.

Отже, віртуальний сервер, на якому розміщено додаток, має певні доступні ресурси (ядра процесора, оперативна пам'ять, операції вводу/виводу).

$$Ra = \{Ca, Ma, Na\}, \quad (1)$$

де *Ra* – це множина доступних ресурсів віртуального сервера; *Ca* – кількість доступних ядер процесора (або потужність CPU). Це означає, скільки ядер процесора виділено для цього віртуального сервера; *Ma* – кількість доступної оперативної пам'яті (RAM) для віртуального сервера. Це ресурс, який дозволяє процесам на сервері працювати безпосередньо в оперативній пам'яті, що впливає на швидкість виконання програм; *Na* – кількість операцій вводу/виводу (I/O), які можуть бути виконані сервером. Це стосується продуктивності системи при роботі з дисковими операціями або іншими пристроями вводу/виводу, що може впливати на загальну швидкість обробки даних;

Кожен обробник який виконує фонове завдання чи надісланий запит від користувача тощо використовує певну частину доступних ресурсів:

$$Rw = \{Cw, Mw, Nw\}, \quad (2)$$

де *Rw* – це множина ресурсів, які використовує конкретний обробник під час виконання своєї роботи; *Cw* – кількість процесорних ресурсів (ядер CPU), які використовуються обробником (це означає, скільки часу процесора потрібно цьому обробнику для виконання свого завдання); *Mw* – кількість оперативної пам'яті (RAM), що використовується обробником. Це визначає обсяг пам'яті, який займає обробник у системі під час свого виконання; *Nw* – кількість операцій вводу/виводу (I/O), які виконує обробник. Це стосується кількості операцій читання або запису даних на диск чи інші пристрої вводу/виводу, що процес виконує в ході своєї роботи.

Тоді середню величину кількості обробників можна розрахувати наступним чином:

$$Nw = Min(Ca/Cw, Ma/Mw, Na/Nw), \quad (3)$$

Вхідні параметри: *Ra* = {*Ca, Ma, Na*} – доступні ресурси сервера: розмір черги завдань, які потрібно виконати; запити до зовнішніх сервісів, які можуть потребувати часу на очікування.

Розглянемо кроки алгоритму:

1. Розрахувати кількість обробників відносно виділених ресурсів віртуального серверу. Використовуючи доступні ресурси для визначення максимально можливої кількості обробників *Nw*, яка може бути визначається за формулою (3).

2. Визначити скільки часу обробники витрачають на очікування. Використовуючи модель, засновану на принципі того, що обробники, які чекають на результат I/O (запис/читання), можуть бути обмеженим фактором.

$$Twait = (\text{середній час очікування I/O})/(\text{середній час виконання потоку}) \quad (4)$$

Якщо *Twait* - високе, то можна збільшити кількість потоків, оскільки потоки будуть частину часу простоювати під час очікування I/O.

3. Врахування зовнішніх інтеграцій. Якщо платформа активно використовує зовнішні API або сервіси, які викликають значне блокування, необхідно визначити частку часу, протягом якого обробники очікують на відповіді зовнішніх систем. Це дозволяє підвищити кількість обробників без перевантаження локальних ресурсів.

$$Nпотоків = Nmax / (1+Tінтеграції/Tвиконання), \quad (5)$$

де *Tінтеграції* - час очікування на зовнішні сервіси; *Tвиконання* - час виконання інших операцій потоку.

4. Розрахувати кількість обробників, врахувавши розмір черги фонових операцій. Якщо черга фонових операцій велика, це може бути індикатором, що додатку потрібно більше потоків для зменшення затримок. Однак важливо не перевищити ліміти ресурсів сервера.

$$Nобробників = Min(Nпотоків, Nчерга + K), \quad (6)$$

де *Nчерга* - поточний розмір черги фонових завдань; *K* - додатковий резерв для обробки нових запитів.

5. Вивід результату. Результатом роботи алгоритму є оптимальна кількість потоків *N обробників*, яка максимізує ефективність використання ресурсів віртуального сервера та зменшує затримки у виконанні фонових операцій та обробці запитів користувачів.

Практичні та експериментальні результати

Запропонований алгоритм застосовувався і продовжує застосовуватися на середовищах із такими характеристиками - Табл.2. Для всіх випадків налаштування відбувалося лише один раз для кожного різного екземпляру платформи.

Таблиця 2

Характеристики середовища, на якому розміщена платформа

Server	CPU (cores)	Memory (GB)
Web	8	63.9
SQL	8	63.9

Після налаштування було отримано наступні результати, Табл 3.

Таблиця 3

Значення параметрів після конфігурування

Назва параметру	Значення
quartz.threadPool.threadCount	5
nr-of-instances	5
messageProcessingThreadsCount	10

При даних налаштуваннях ресурси серверів, на яких розміщувались екземпляри платформи, рівень використання ресурсів представлено на рис. 2.

Рис.2. Використання процесорного часу веб-сервера

Графік рис. 2 відображає використання процесорного часу веб-сервера у відсотках протягом певного періоду. Червона, горизонтальна лінія на графіку вказує на порогове значення використання CPU рівне 85%, після якого сервер може вважатися перевантаженим або, де його продуктивність може почати знижуватись. Більшість часу використання CPU коливається в межах 5-35%, з декількома періодичними піками, які доходять до 40-50%. Піки свідчать про періоди підвищеного навантаження на веб-сервер, але жоден із них не перевищує критичне порогове значення (85%), що означає, що сервер загалом справляється з навантаженням.

Графічна інтерпретація використання оперативної пам’яті веб-сервера, представлена на рис. 3.

Рис.3. Використання оперативної пам’яті веб-сервера

На графіку рис. 3 червона горизонтальна лінія позначає порогове значення 85% для використання пам'яті, після якого система може почати втрачати продуктивність або навіть зазнавати проблем. Використання оперативної пам'яті здебільшого коливається між 60% та 85%, із деякими спадами нижче 60% і короткочасними піками, що наближаються до порогового рівня. Загалом, графік демонструє тенденцію до стабільного використання оперативної пам'яті на рівні середнього до високого завантаження.

Кількість підключень та кількість користувачів представлена на рис. 4.

Рис.4. Кількість підключень та кількість користувачів

Цей графік рис. 4 відображає кількість підключень і кількість користувачів на сервері протягом певного періоду часу. Жовта лінія, яка має кілька піків і значні коливання відображає кількість підключень. На графіку видно періоди, коли кількість підключень різко збільшується, досягаючи високих значень (близько 380-400), а потім спадає. Зелена лінія, яка знаходиться значно нижче і менш мінлива відображає кількість активних користувачів. Вона має деякі коливання, але залишається відносно стабільною порівняно з лінією підключень. Кількість підключень має тенденцію до різких коливань, що може бути пов'язано зі збільшенням навантаження на сервер у певні моменти або виконанням обчислювально-інтенсивних операцій, які потребують більше підключень. Кількість користувачів залишається стабільною з невеликими змінами, що може свідчити про сталість бази користувачів без значних змін у загальній кількості, які були б помітні на графіку.

Використання ресурсів оперативної пам’яті та процесорного часу *SQL*-сервера представлена на рис.5. 5.

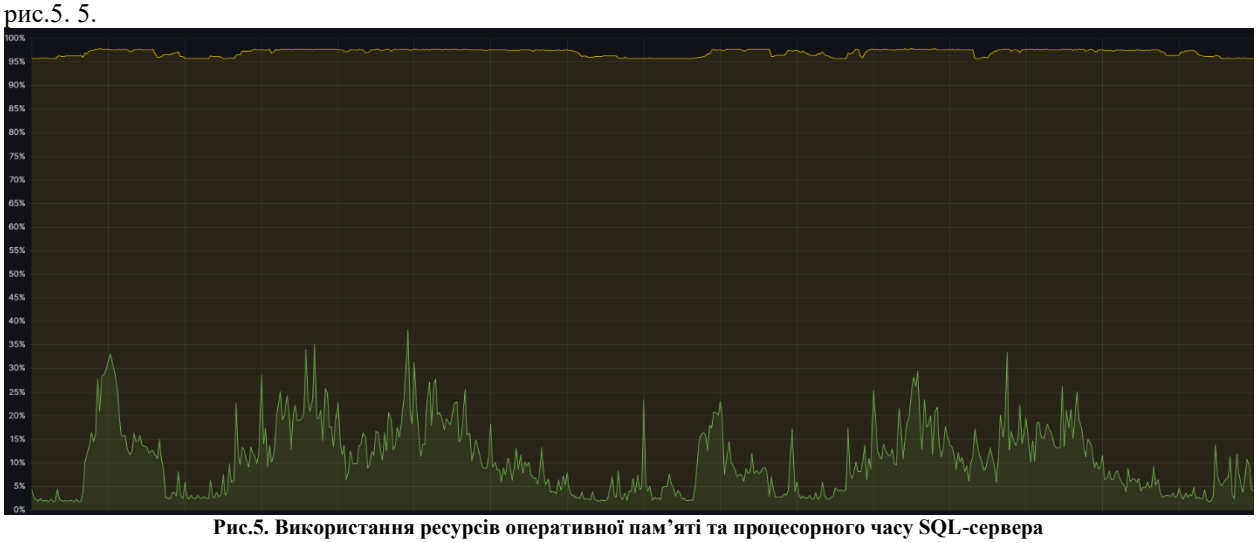

Рис.5. Використання ресурсів оперативної пам’яті та процесорного часу SQL-сервера

На рис. 5 нижча зелена лінія перебуває в межах 5-35%, відображає використання процесорного часу *SQL*-сервера. Це використання CPU варіюється, показуючи періодичні сплески, але здебільшого залишається в межах низького до середнього рівня завантаженості. Час від часу є піки, що можуть свідчити про інтенсивні запити або виконання великих обчислень, але вони не довготривалі. *SQL*-сервер стабільно використовує всю доступну оперативну пам'ять (жовта лінія), що відображає нормальну ситуацію для даного типу серверу БД. Використання CPU є динамічним і не перевищує значних значень, що може

свідчити про ефективне використання процесорного часу, хоча інколи можуть виникати короткочасні періоди навантаження.

На рис. 6 представлено заблоковані процеси *SQL*.

Рис.6. Заблоковані процесів SQL

На рис. 6 більшу частину часу значення близькі до нуля, що свідчить про нормальний стан роботи системи без значного блокування процесів. Проте спостерігаються кілька піків, де кількість заблокованих процесів різко зростає до значень понад 1,8 та навіть 2,5 у деяких випадках. Це може вказувати на періодичні моменти навантаження або проблеми з *SQL*-запитами, що спричиняють блокування. Після кожного піка значення повертаються до низьких рівнів, що може означати, що блокування тимчасові і система вміє з ними справлятися.

Дискова черга *SQL* представлена на рис. 7.

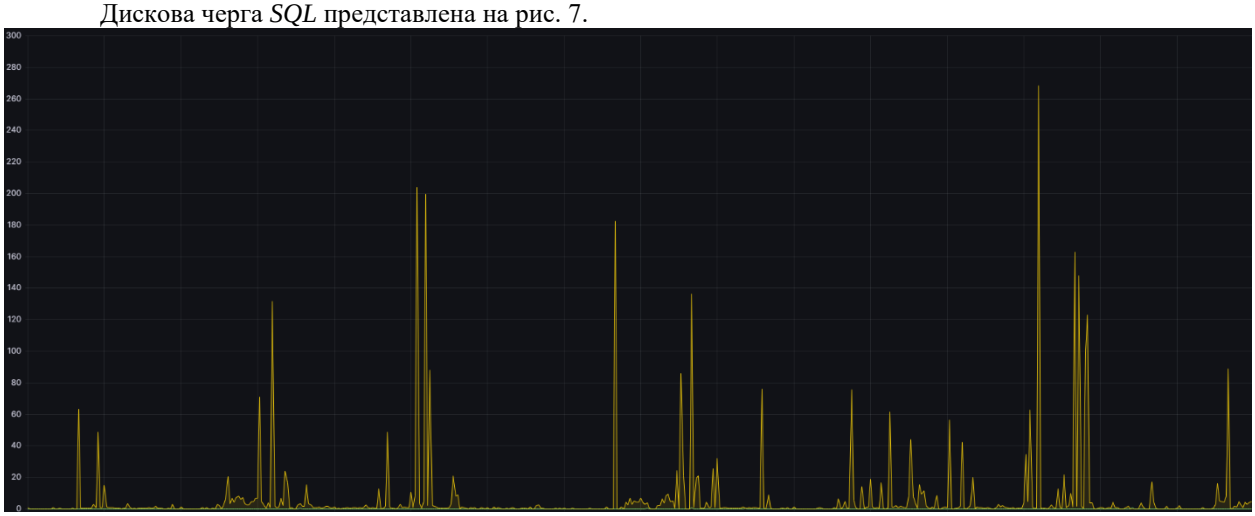

Рис.7. Дискова черги SQL

Графік дискової черги *SQL* (рис. 7) показує кількість запитів, що очікують на доступ до диска. Більшість часу значення дискової черги близьке до нуля, що свідчить про нормальну роботу диска і достатні ресурси для обробки запитів. На графіку є декілька великих піків, де кількість запитів в черзі досягає значень понад 100 і навіть до 300 у певних моментах. Це може вказувати на короткочасні періоди значного навантаження на диск. Однією із причин даних ситуацій можуть бути виконання запитів, що потребують багато ресурсів, особливо якщо відбувається паралельне виконання декількох таких запитів. Загалом, дискова черга *SQL* працює стабільно, однак присутні епізодичні піки.

На графіку затримки *SQL*-диска (рис. 8) видно, що середній час відгуку (latency) переважно коливається в межах 2-4 мс, що є хорошим показником для більшості *SQL*-операцій.

Графік на рис. 9 повідомляє, що платформа була не доступна, тобто бізнес-сервіси призупинили свою роботу.

Графік на рис. 10 відображає показники I/O (*Input/Output operations per second*), що вимірюють інтенсивність операцій вводу-виводу, виконуваних сервером. Жовта лінія на графіку показує значні коливання у величині IOPS, з численними піками, які іноді сягають рівня 3000–3500 операцій на секунду.

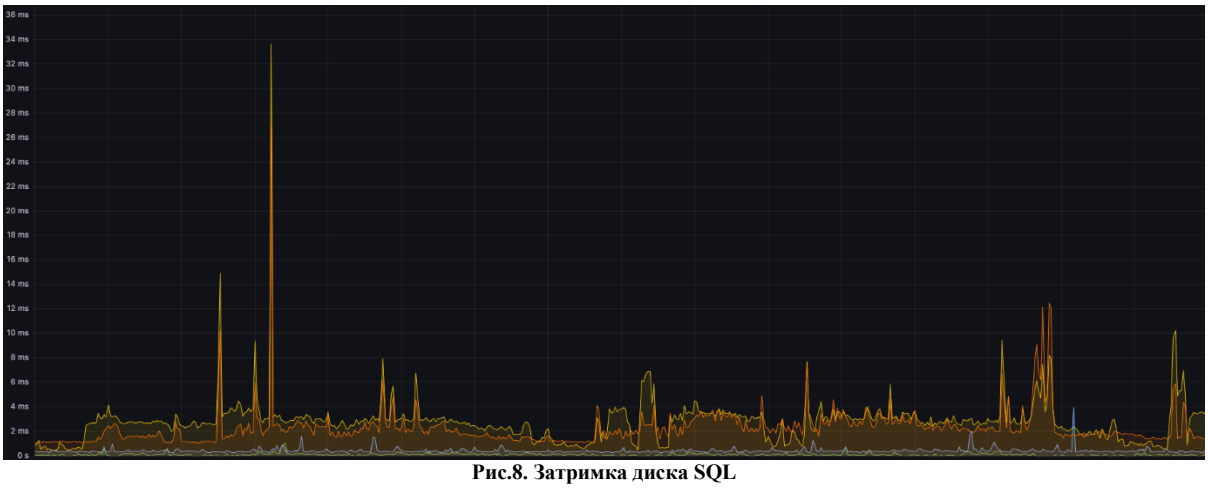

Рис.8. Затримка диска SQL

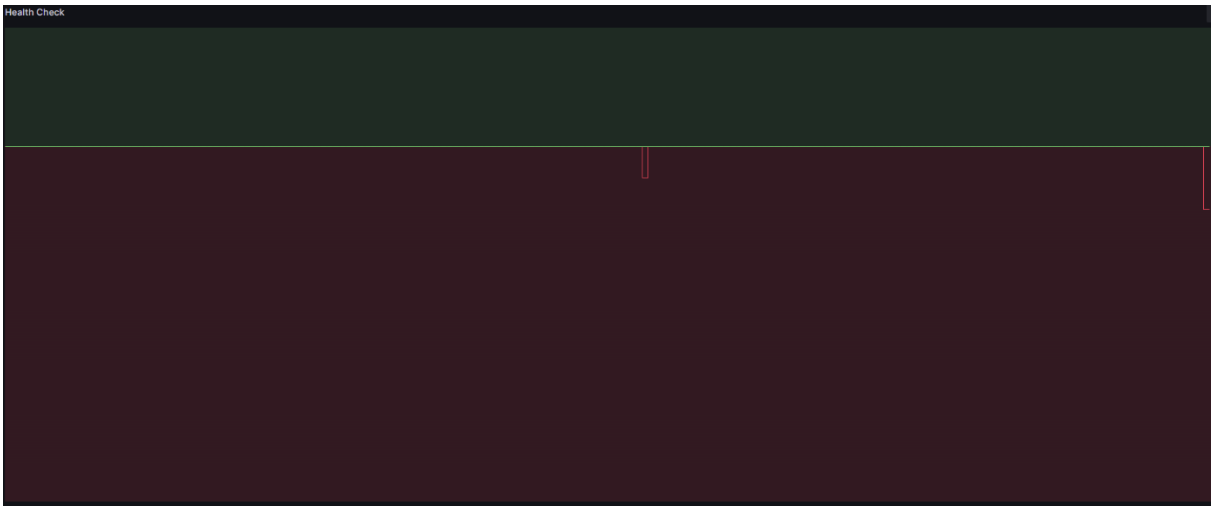

Рис.9. Активність/здоров'я веб-додатку

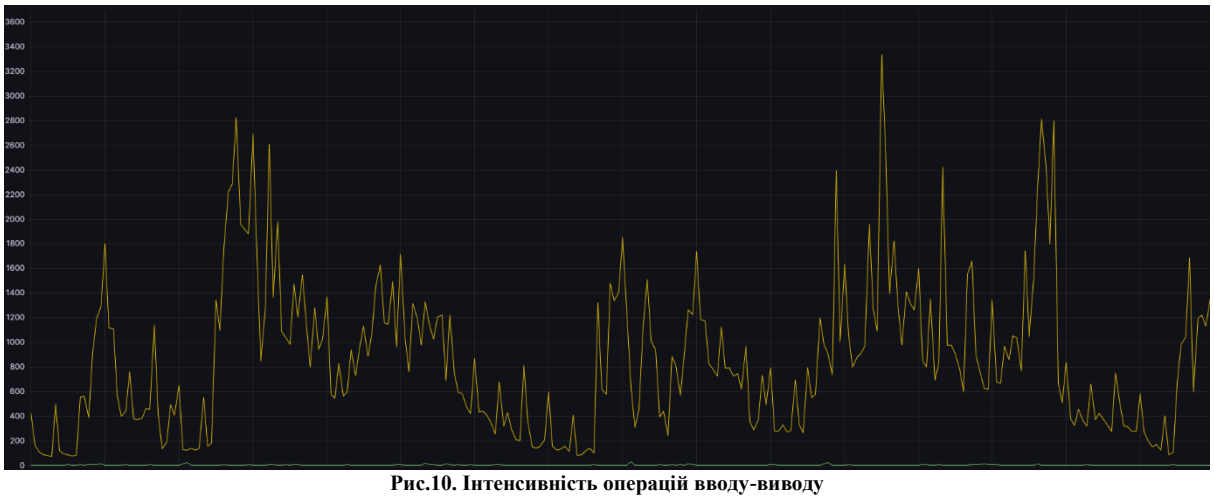

Рис.10. Інтенсивність операцій вводу-виводу

Це свідчить про періодичні сплески інтенсивності вводу-виводу. Більшу частину часу значення коливаються в межах 500-2000, з більш високими піками, які вказують на інтенсивні періоди роботи вводу-виводу. Зелена лінія, яка залишається на нижчому рівні, може відображати базове навантаження або операції, що не залежать від високих навантажень вводу-виводу. Сервер періодично піддається високим навантаженням на ввід-вивід, що може бути спричинено великими запитами до бази даних або інтенсивними операціями, які потребують високої пропускної здатності. Такі різкі піки можуть призводити до затримок або перевантаження дискової підсистеми, що може впливати на загальну продуктивність сервера.

ВИСНОВКИ З ДАНОГО ДОСЛІДЖЕННЯ І ПЕРСПЕКТИВИ ПОДАЛЬШИХ РОЗВІДОК У ДАНОМУ НАПРЯМІ

Виходячи з аналізу графіків, можна зробити наступні висновки щодо продуктивності та стабільності серверної інфраструктури. Заблоковані процеси *SQL*-сервера: Графік показує періодичні сплески блокування процесів, що вказує на моменти, коли певні *SQL*-запити викликають блокування ресурсів. Це може впливати на загальну продуктивність бази даних. Рекомендується провести аудит *SQL*-запитів для виявлення та оптимізації тих, які можуть викликати блокування. Використання CPU веб-сервера: Веб-сервер демонструє стабільне використання процесорного часу в межах допустимих значень (5-35%) з короткочасними піками до 40-50%. Це вказує на те, що веб-сервер справляється із запитами, і його продуктивність залишається стабільною. Поки що немає потреби в додаткових ресурсах для CPU, але слід відслідковувати зростання навантаження. Використання оперативної пам'яті веб-сервера: Використання пам'яті веб-сервера наближається до порогового значення 85%, що може свідчити про постійне навантаження. Зафіксований випадок (рис. 9), коли бізнес-сервіси були недоступні. В логах платформи зафіксована причина: ”*A worker process serving application pool has requested a recycle because it reached its private bytes memory limit*.” недостатньо оперативної пам’яті. Тому для забезпечення стабільної роботи серверу в майбутньому може бути доцільно розглянути збільшення обсягу пам'яті або оптимізувати існуючі бізнес-процеси для більш ефективного використання пам'яті. Кількість підключень має суттєві коливання з різкими піками, що може свідчити про періодичне навантаження або специфічні запити, що вимагають великої кількості одночасних з'єднань. Кількість користувачів залишається відносно стабільною, що означає відсутність значних коливань у базі користувачів. Варто провести оптимізацію роботи з підключеннями для забезпечення стабільності роботи сервера під час пікових навантажень. Загалом вказані значення фонових обробників дозволяє серверній інфраструктура справлятися з поточним навантаженням, але є певні ознаки обмежень у ресурсах, особливо в оперативній пам’яті веб-серверу. Виконання рекомендованих заходів допоможе підвищити продуктивність і стабільність системи, а також підготує її до можливих майбутніх навантажень.

Використання даного алгоритму налаштування кількості фонових обробників дозволило зменшити на 50% кількість повторних заявок (*Update configuration files on the site*) на комаду клауд (*Cloud service*), тобто кількість скарг зменшилось вдвічі від користувачів через причину повторного налаштування платформи. Адже при невірному налаштуванні кількості фонових обробників витрачалось більше часу на те, щоб проаналізувати та виправити ситуацію після змін налаштування. Подібні ситуації призводили до неякісної роботи бізнес-сервісів або їх зупинки.

Література

1. L. Búrdalo, A. Terrasa, A. Espinosa and A. García-Fornes, "Analyzing the Effect of Gain Time on Soft-Task Scheduling Policies in Real-Time Systems," in IEEE Transactions on Software Engineering, vol. 38, no. 6, pp. 1305-1318, Nov.-Dec. 2012, doi: 10.1109/TSE.2011.95.

2. A. Desai, Nagegowda K S and Ninikrishna T, "Secure and QoS aware architecture for cloud using software defined networks and Hadoop," 2015 International Conference on Computing and Network Communications (CoCoNet), Trivandrum, India, 2015, pp. 369-373, doi: 10.1109/CoCoNet.2015.7411212.

3. R. L. Neupane et al., "Online Self-Service Learning Platform for Application-Inspired Cloud Development and Operations (DevOps) Curriculum," in IEEE Transactions on Learning Technologies, vol. 17, pp. 1946-1960, 2024, doi: 10.1109/TLT.2024.3428842.

4. X. Guang-cai, W. Zhi-feng, Z. Xin-jia and J. Guo-jun, "Realization of business process automation based on web services and WS-BPEL," ICSSSM11, Tianjin, China, 2011, pp. 1-5, doi: 10.1109/ICSSSM.2011.5959489.

5. J. Huang, "Analysis and Development Strategy for Modern Enterprise Informatization from the Perspective of Business Administration," 2021 13th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA), Beihai, China, 2021, pp. 1-4, doi: 10.1109/ICMTMA52658.2021.00009.

6. T. Moriya, H. Ohnishi, M. Yoshida, T. Ogawa and M. Hirano, "A Support System for Designing Ubiquitous Service Composition Scenarios," 2007 IEEE International Conference on Communications, Glasgow, UK, 2007, pp. 1594-1599, doi: 10.1109/ICC.2007.267.

7. J. Ahn *et al.*, "Server load and network-aware adaptive deep learning inference offloading for edge platforms," Internet of Things, vol. 21, pp. 100644, 2023, https://doi.org/10.1016/j.iot.2022.100644.

8. Microsoft Ignite. Threads and threading [Electronic resource]. – 2024. - Microsoft 2024 - Access mode: https://learn.microsoft.com/en-us/dotnet/standard/threading/threads-and-threading (last access: 18.11.2024).

9. Microsoft Ignite. Foreground and background threads [Electronic resource]. – 2024. - Microsoft 2024 - Access mode: https://learn.microsoft.com/en-us/dotnet/standard/threading/foreground-and-background-threads (last access: 18.11.2024).

10. A. Jeffery, C. Jensen and R. Mortier, "Reducing Tail Latencies Through Environment- and Neighbour-aware Thread Management," Distributed, Parallel, and Cluster Computing, arXiv:2407.11582, 2024, https://doi.org/10.48550/arXiv.2407.11582.

11. Li, SS., Liao, XK., Tan, YS., Liu, JY. (2005). Dynamic Thread Management in Kernel Pipeline Web Server. In: Jin, H., Reed, D., Jiang, W. (eds) Network and Parallel Computing. NPC 2005. Lecture Notes in Computer Science, vol 3779. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11577188_16.

12. Бойко Ю. SAML : дефініція та принцип роботи через VPN тунель у захищених інформаційних мережах / Ю. Бойко, Б. Білявець // Вимірювальна та обчислювальна техніка в технологічних процесах. – 2022. – № 4. – С. 41-48.

13. J. Boiko, I. Pyatin, O. Eromenko, L. Karpova, Evaluation of the Capabilities of LDPC Codes for Network Applications in the 802.11ax Standard, in: Joby, P.P., Alencar, M.S., Falkowski-Gilski, P. (Eds.), IoT Based Control Networks and Intelligent Systems. Lecture Notes in Networks and Systems, volume 789, Springer, Singapore, 2024, pp. 369–383, doi:10.1007/978-981-99-6586- 1_25.

14. J. Boiko, V. Druzhynin, S. Buchyk, I. Pyatin, and A. Kulko, "Methodology of FPGA Implementation and Performance Evaluation of Polar Coding for 5G Communications", CEUR Workshop Proceedings, 3654 (2024): 15-24, urn:nbn:de:0074-3654-7.

15. S. S. Phyoe and T. N. Aung, "Web Content Charging by using Thread", Fourth Local Conference on Parallel and Soft Computing, 2009, URI: https://onlineresource.ucsy.edu.mm/handle/123456789/1464.

16. Quartz.NET. Open-source job scheduling system for .NET [Electronic resource]. – 2007- present Marko Lahma - Access mode: https://www.quartz-scheduler.net/ (last access: 18.11.2024).

17. B. Zhurakovskyi, J. Boiko, V. Druzhynin, I. Zeniv and O. Eromenko. "Increasing the efficiency of information transmission in communication channels," Indonesian Journal of Electrical Engineering and Computer Science (IJEECS), 19.3 (2020): 1306-1315. doi:10.11591/ijeecs.v19.i3.pp1306-1315.

References

1. L. Búrdalo, A. Terrasa, A. Espinosa and A. García-Fornes, "Analyzing the Effect of Gain Time on Soft-Task Scheduling Policies in Real-Time Systems," in IEEE Transactions on Software Engineering, vol. 38, no. 6, pp. 1305-1318, Nov.-Dec. 2012, doi: 10.1109/TSE.2011.95.

2. A. Desai, Nagegowda K S and Ninikrishna T, "Secure and QoS aware architecture for cloud using software defined networks and Hadoop," 2015 International Conference on Computing and Network Communications (CoCoNet), Trivandrum, India, 2015, pp. 369-373, doi: 10.1109/CoCoNet.2015.7411212.

3. R. L. Neupane et al., "Online Self-Service Learning Platform for Application-Inspired Cloud Development and Operations (DevOps) Curriculum," in IEEE Transactions on Learning Technologies, vol. 17, pp. 1946-1960, 2024, doi: 10.1109/TLT.2024.3428842.

4. X. Guang-cai, W. Zhi-feng, Z. Xin-jia and J. Guo-jun, "Realization of business process automation based on web services and WS-BPEL," ICSSSM11, Tianjin, China, 2011, pp. 1-5, doi: 10.1109/ICSSSM.2011.5959489.

5. J. Huang, "Analysis and Development Strategy for Modern Enterprise Informatization from the Perspective of Business Administration," 2021 13th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA), Beihai, China, 2021, pp. 1-4, doi: 10.1109/ICMTMA52658.2021.00009.

6. T. Moriya, H. Ohnishi, M. Yoshida, T. Ogawa and M. Hirano, "A Support System for Designing Ubiquitous Service Composition Scenarios," 2007 IEEE International Conference on Communications, Glasgow, UK, 2007, pp. 1594-1599, doi: 10.1109/ICC.2007.267.

7. J. Ahn et al., "Server load and network-aware adaptive deep learning inference offloading for edge platforms," Internet of Things, vol. 21, pp. 100644, 2023, https://doi.org/10.1016/j.iot.2022.100644.

8. Microsoft Ignite. Threads and threading [Electronic resource]. – 2024. - Microsoft 2024 - Access mode: https://learn.microsoft.com/en-us/dotnet/standard/threading/threads-and-threading (last access: 18.11.2024).

9. Microsoft Ignite. Foreground and background threads [Electronic resource]. – 2024. - Microsoft 2024 - Access mode: https://learn.microsoft.com/en-us/dotnet/standard/threading/foreground-and-background-threads (last access: 18.11.2024).

10. A. Jeffery, C. Jensen and R. Mortier, "Reducing Tail Latencies Through Environment- and Neighbour-aware Thread Management," Distributed, Parallel, and Cluster Computing, arXiv:2407.11582, 2024, https://doi.org/10.48550/arXiv.2407.11582.

11. Li, SS., Liao, XK., Tan, YS., Liu, JY. (2005). Dynamic Thread Management in Kernel Pipeline Web Server. In: Jin, H., Reed, D., Jiang, W. (eds) Network and Parallel Computing. NPC 2005. Lecture Notes in Computer Science, vol 3779. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11577188_16.

12. J. Boiko and B. Biliavets, "SAML: definition and principles of operation through a VPN tunnel in secure information networks," Measuring and computing devices in technological processes, № 4., pp. 41-48, 2022, https://doi.org/10.31891/2219-9365-2022-72-4-4.

13. J. Boiko, *et. al*., Evaluation of the Capabilities of LDPC Codes for Network Applications in the 802.11ax Standard. IoT Based Control Networks and Intelligent Systems. Lecture Notes in Networks and Systems, volume 789, Springer, Singapore, 2024, pp. 369–383, doi:10.1007/978-981-99-6586- 1_25.

14. J. Boiko, *et. al*., "Methodology of FPGA Implementation and Performance Evaluation of Polar Coding for 5G Communications", CEUR Workshop Proceedings, 3654 (2024): 15-24, urn:nbn:de:0074-3654-7.

15. S. S. Phyoe and T. N. Aung, "Web Content Charging by using Thread", Fourth Local Conference on Parallel and Soft Computing, 2009, URI: https://onlineresource.ucsy.edu.mm/handle/123456789/1464.

16. Quartz.NET. Open-source job scheduling system for .NET [Electronic resource]. – 2007- present Marko Lahma - Access mode: https://www.quartz-scheduler.net/ (last access: 18.11.2024).

17. B. Zhurakovskyi, J. Boiko, V. Druzhynin, I. Zeniv and O. Eromenko. "Increasing the efficiency of information transmission in communication channels," Indonesian Journal of Electrical Engineering and Computer Science (IJEECS), 19.3 (2020): 1306-1315. doi:10.11591/ijeecs.v19.i3.pp1306-1315.