***RMRT-EDF: A Reactive Multi-Resource Token Scheduler for Guaranteed Real-Time Task Start-up***

|  |  |
| --- | --- |
| Kornaha Y. I.  Department of Information Systems and Technologies,  Faculty of Informatics and Computer Science  National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute" (NTUU "KPI")  Kyiv, Ukraine  [slovyan\_k@ukr.net](mailto:slovyan_k@ukr.net)  Lemeshko V. A.  Department of Information Systems and Technologies,  Faculty of Informatics and Computer Science  National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute" (NTUU "KPI")  Kyiv, Ukraine  [slava.lemeshko@gmail.com](mailto:slava.lemeshko@gmail.com) |  |
| ***Abstract.* Reactive Multi Resource Token-EDF (RMRT-EDF) is introduced for systems that must start every task within 1s under limited CPU, RAM and I/O. The three layer scheme combines EDF scheduling, multi‑token resource admission and feedback‑driven thread‑pool tuning. Simulation with two workers cut average start delay from 450ms to 72ms, reduced SLA misses 35 fold and raised 95th percentile CPU use to 85%. The method suits latency critical web services.**  ***Keywords*: *task scheduling, deadline, resource constraints, real-time systems.*** | |

***RMRT-EDF: реактивний мультиресурсний токен-планувальник для гарантованого запуску задач у реальному часі***

Корнага Ярослав Ігорович

Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського"

м. Київ, Україна

[slovyan\_k@ukr.net](mailto:slovyan_k@ukr.net)

Лемешко В’ячеслав Анатолійович

Кафедра Інформаційних систем та технологій Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського"

м. Київ, Україна  
[slava.lemeshko@gmail.com](mailto:slava.lemeshko@gmail.com)

*Анотація.* Запропоновано трирівневий планувальник RMRT-EDF для систем із жорсткою вимогою старту задач ≤ 1с та обмеженими CPU, RAM, I/O. Алгоритм поєднує EDF із мультиресурсним токен-bucket і адаптивним регулюванням пулу потоків. У моделюванні з двома воркерами середня затримка запуску зменшилася з 450мс до 72мс, порушення SLA — у 35 разів, а 95й перцентиль завантаження CPU зріс до 85%. Підхід придатний для веб-сервісів реального часу.

# *Ключові слова: планування задач, дедлайн, ресурсні обмеження, системи реального часу.*

Вступ

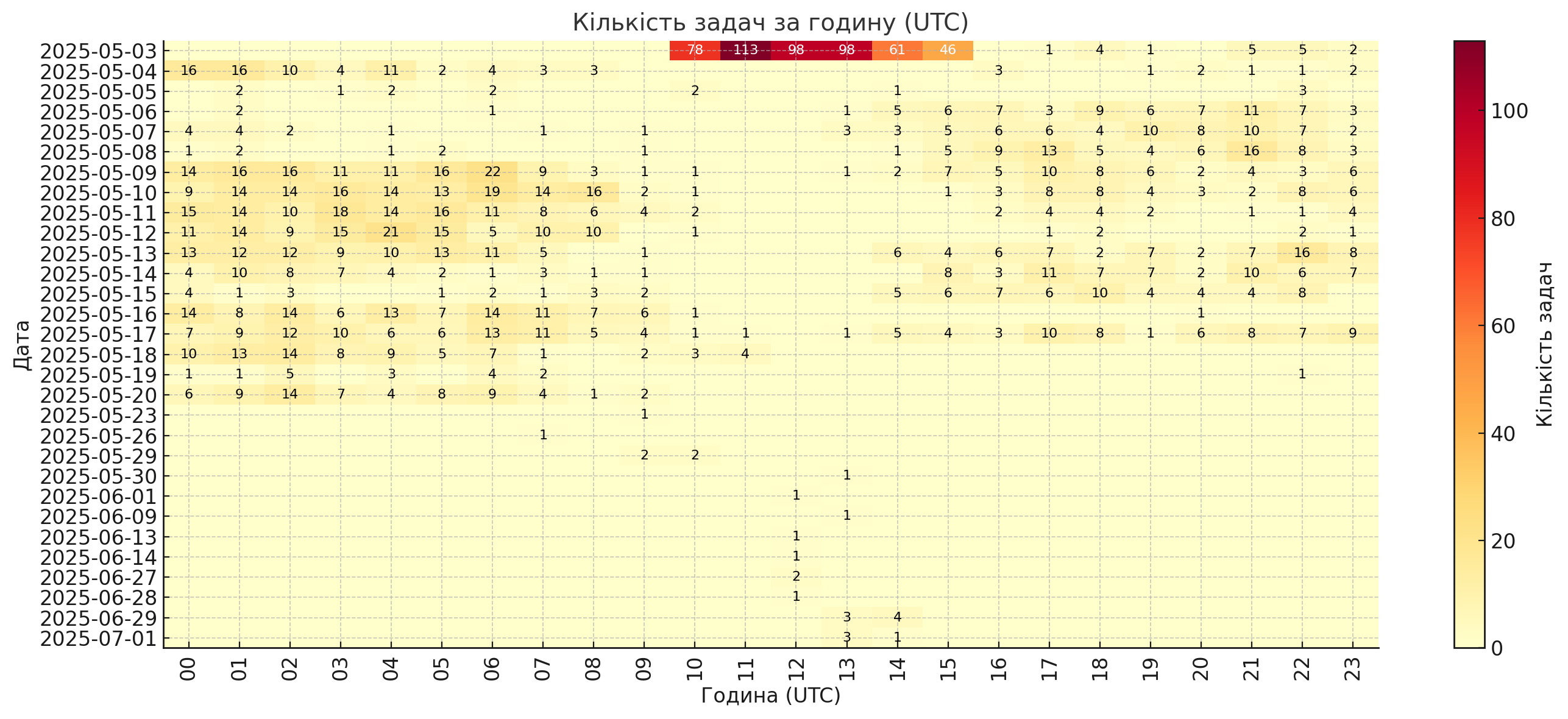
Зростання вимог до оперативності бізнес-веб-сервісів у більшості сучасних інформаційних систем (банківські платформи, e-commerce, телеком, IoT-сервіси) час реакції визначає конкурентоспроможність і прямо впливає на задоволеність клієнта. Якщо заплановані або раптові задачі стартують із затримкою понад секунду, це призводить до порушення SLA, фінансових штрафів і втрати довіри до сервісу.

У реальних системах існує "скелет" добових (статичних) задач – резервне копіювання, пакетна обробка даних, звітність тощо. Одночасно впродовж дня виникають непередбачувані події (нові замовлення, транзакції, нотифікації), які потребують негайного запуску. Класичні планувальники оптимізовані або під регулярні, або під ad-hoc-навантаження, але не під їхню комбінацію.

Сервіс виконується у середовищі з фіксованими CPU, Memory, IO-квотами. При цьому окремі задачі споживають ресурси нерівномірно (наприклад, ETL-процеси навантажують диск, аналітика – процесор та пам’ять). Ігнорування багатовимірної природи ресурсів призводить до локальних "вузьких місць", коли процесор вільний, а диск перевантажений або навпаки.

Операції класу near-real-time (фінансові підтвердження, antifraud-перевірки, керування виробничим обладнанням) критичні до латентності. Тому необхідний планувальник, який враховує не лише середні значення, а й найгірший сценарій (worst-case response time).

# Постановка задачі



У рамках дослідження розглядається система, що обробляє множину задач, які включають як заздалегідь заплановані задачі S = {s₁, s₂, …, sₙ}, так і динамічний потік задач D(t), що надходять у реальному часі з невідомою інтенсивністю λ(t). Для кожної задачі задані оцінки ресурсів: споживання CPU, пам’яті та інтенсивності I/O (дискових або мережевих операцій), а також допустиме вікно старту [tᵢ, tᵢ+Δ], в межах якого її виконання вважається вчасним.

Система повинна здійснювати диспетчеризацію задач у пул обчислювальних потоків P(t) так, щоб виконувались як часові обмеження (latency ≤ 1 сек), так і обмеження за ресурсами хост-сервера. У випадках високої навантаженості, коли інтенсивність вхідного потоку λ(t) перевищує доступні ресурси, виникають черги очікування, що призводять до порушень дедлайнів та деградації продуктивності.

Завдання ускладнюється тим, що класична постановка такого планування задач не є повною. Тому особливого значення набирають евристичні або гібридні стратегії, здатні адаптивно реагувати на зміну навантаження в системі. Підхід має забезпечити гнучке керування чергами задач та коефіцієнт використання системних ресурсів.

На (рис. 1) подано годинний розподіл щільності запусків задач у межах кожної доби. Аналіз діаграми засвідчує, що наперед статично заплановано лише незначну частку завдань, тоді як переважна їх кількість формується динамічно впродовж робочого дня, коли користувачі активно взаємодіють із системою автоматизації бізнес-веб-сервісів.

Зображення, що містить текст, ряд, знімок екрана, Графік

Вміст, створений ШІ, може бути неправильним.

Рис. 2. Динаміка тривалості виконання задачі у (мс) за період спостереження

Графік (рис. 2) ілюструє значну варіативність часу виконання однієї з планових задач, що підтверджує нестабільність умов її запуску (наприклад, фонове навантаження, контекст виконання, стан ресурсів). Отже, використання метрики найгіршого часу виконання (worst-case execution time) як єдиного орієнтира для оцінки продуктивності або для побудови політик планування є недоцільним. Більш обґрунтованим підходом є використання статистичних оцінок (медіани, перцентилів). Але такі випадки тривалого виконання варто врахувати при розподілі задач.

Нехай у системі присутній набір статично запланованих задач S = {s₁, s₂, …, sₙ} з відповідними моментами старту tᵢ та оцінками споживання ресурсів rᵢ = (cpuᵢ, memᵢ, ioᵢ). Протягом доби може зʼявлятися динамічний потік задач D(t) з невідомою інтенсивністю λ(t). Необхідно знайти таке відображення задач у пул потоків P(t), що для всіх s ∈ S ∪ D(t) виконується t\_start(s) – tᵤ(s) ≤ 1 секунді та не порушуються ресурсні обмеження сервера.

# Огляд існуючих досліджень

Загалом, існуючі підходи умовно можна поділити на такі категорії.

Таблиця 1

Класифікація алгоритмів планування

|  |  |  |
| --- | --- | --- |
| **Категорія** | **Стисла характеристика** | **Алгоритми** |
| Статичні класичні евристики | Черга формується разово; правило вибору задачі не змінюється під час виконання. | SJF, FCFS – порівняльні тести та аналіз [4]; [18]; [8] |
| Реактивні планувальники реального часу | Пріоритети перераховуються on‑the‑fly, гарантуються дедлайни. | EDF, EFDF – огляд багатопроцесорних систем [2]; Порівняння статичних/динамічних пріоритетів [20]; Систематичний огляд RT‑алгоритмів [6] |
| Метаевристики | Стохастичний пошук квазіоптимальних розкладів; добре масштабується. | GWO‑Fog [1]; GA [21]; PSO/MFO/Bees Life [22], [15]; Hybrid heuristic [13]; Improvised threshold‑based [5] |
| Гібридні динамічні стратегії з навчанням | Комбінують евристику/метаевристику з адаптивним навчанням; реагують на зміни λ(t). | Hybrid Learning &  Heuristics [19]; RL‑based [23]; RRA‑SWO + HAGA [9]; Deadline‑constrained  local search [10] |
| Ресурсо‑орієнтовані токен‑алгоритми | Використовують «кошики» токенів для CPU/RAM/I/O; задача приймається лише за наявності всіх потрібних токенів. | Dynamic Thread  Allocation [3]; Real‑Time Scheduling [11]; Densest‑Job‑Set‑First [12] |
| Енерго‑ та дедлайн‑орієнтовані оптимізації | Мінімізують енергоспоживання, дотримуючись дедлайнів; часто базуються на роях або їх гібридах. | Energy‑Aware Bees Life [15]; Swarm‑based  Deadline‑Aware [16]; Enhanced Round‑Robin [7] |

Серед проаналізованих підходів (табл. 1), включаючи як класичні статичні евристики [4], [18], [8], так і метаевристичні алгоритми ройової оптимізації [1], [15], [21] та реактивні планувальники реального часу [2], [6], [20] — не забезпечує гарантованого старту всіх задач з точністю до 1 секунди за умов динамічного потоку задач D(t) з непередбачуваною інтенсивністю λ(t). Практичні результати свідчать, що найбільш перспективним є поєднання таких трьох компонентів:

1. Динамічного планувальника на основі дедлайнів (наприклад, EDF/EFDF [2] або HEFT-T з адаптивними пріоритетами), який забезпечує низьку латентність реагування та простоту переналаштування у разі надходження нових подій;
2. Механізмів мультиресурсного контролю із застосуванням токен-кошиків [3], [11], [12] або енерго- та дедлайн-чутливих метаевристик [15], [16], що дозволяють перевіряти наявність достатніх ресурсів (CPU, RAM, I/O) до моменту допуску задачі;
3. Адаптивного регулювання пулу потоків на основі показників використання ресурсів та частоти порушення дедлайнів (miss-rate), реалізованого через гібридні навчальні стратегії [9], [19], [23].

Інтеграція зазначених механізмів у межах запропонованого підходу RMRT-EDF (Reactive Multi-Resource Token Earliest Deadline First) формує новий клас реактивно-токенових планувальників, які, за наявними аналітичними та емпіричними оцінками, здатні підтримувати рівень SLA із точністю до 1 секунди без погіршення ефективності використання системних ресурсів та втрати масштабованості.

# Концепція запропонованого рішення

Запропонований підхід під назвою Reactive Multi-Resource Token Earliest Deadline First (RMRT-EDF) являє собою трирівневий мікропланувальник, який забезпечує пріоритезацію та допуск задач до виконання з урахуванням часових обмежень та поточного стану системних ресурсів. Підхід поєднує три ключові механізми:

1. Подієво-орієнтоване планування Earliest Deadline First (EDF). Кожній задачі при надходженні присвоюється абсолютний дедлайн d = t + Δ, де t — момент надходження задачі, а Δ ≈ 1 с — фіксоване допустиме вікно виконання. Така схема дозволяє формувати чергу задач із найменшим наближенням до дедлайну (EDF), забезпечуючи пріоритетність обробки термінових запитів.

2. Багатовимірний механізм керування ресурсами на основі «token bucket». Система веде окремі обліки токенів для кожного з критичних ресурсів: процесорного часу (CPU), оперативної пам’яті (RAM), дискових операцій (Disk I/O) та мережевого трафіку (Network I/O). Задача допускається до виконання лише за умови наявності достатньої кількості токенів для кожного з відповідних ресурсів. Така модель дозволяє узгоджено обмежувати використання ресурсів та уникати їхнього перевантаження.

3. Рецидивне тестування здійсненності (receding-horizon feasibility check). За умови, що необхідна кількість токенів вже доступна, для кожної задачі виконується перевірка здійсненності її запуску у горизонті планування H ≈ 5 с, з урахуванням очікуваних навантажень у вікні часу, що постійно оновлюється (receding horizon). Така процедура дозволяє уникнути конфліктів при обмеженій кількості обробників (worker threads) та мінімізувати ризик порушення вимог SLA.

Мікропланувальник — система локального планування, яка працює з мілісекундною точністю для реактивної обробки подій у реальному часі. Token bucket — метод квотування, який широко застосовується в телекомунікаціях і плануванні навантаження для контролю швидкості споживання ресурсів. Receding horizon — підхід, запозичений з оптимального управління, який дозволяє оцінювати доцільність запуску задач не лише на момент їх надходження, а й з урахуванням прогнозу на обмеженому часовому інтервалі.

Архітектура

Для реалізації даної концепції пропонується сформувати систему динамічного планування задач, яка базується на принципах теорії автоматичного керування, зокрема — замкненої системи зі зворотним зв’язком, у якій планування задач відбувається з урахуванням історії навантажень, оцінки ресурсних вимог та динамічного допуску через механізм «оплати» ресурсів у вигляді токенів.

Зображення, що містить текст, знімок екрана, Шрифт, число

Вміст, створений ШІ, може бути неправильним. 

Рис. 3. Діаграма послідовності RMRT-EDF

Predictor (рис. 3) приймає телеметрію (історичні метрики задач). Оцінює вектор ресурсів та тривалість виконання задачі, відповідно (CPU, RAM, IO) вектор rᵢ і тривалість eᵢ. Це може бути реалізоване з допомогою моделі машинного навчання: градієнтний бустинг або LSTM.

Admission перевіряє наявність достатньої кількості токенів для кожного ресурсу. Якщо токенів достатньо, то задача допускається до планування. Інакше, задача переходить у чергу очыкування, де чекає накопичення токенів.

Dispatcher працює із періодом δ=10 мс. Вибирає із EDF-черги задачі, які мають дедлайн найближчий у часі та вже з "оплаченими" токенами та стартує обрані задачі у пулі воркерів (Worker Pool із N-потоків).

Кожен ресурс має свій "кошик" із токенами. Токени поповнюються періодично на основі телеметрії (відгуку системи) — це елемент зворотного зв’язку, як у системах автоматичного регулювання.

##### Очікуваний ефект

Алгоритм RMRT-EDF майже в 6 разів зменшив затримку старту задач (табл. 2). До (FCFS) задачі чекали в середньому, майже, пів секунди, навіть, після настання часу запуску. Після (RMRT-EDF) — середня затримка зменшилася більш ніж у 6 разів. Це покращує реактивність системи, тому задачі запускаються, майже, одразу. Ймовірність довгого очікування P(wait > 1 c) до (FCFS) — більше ніж кожна 9-та задача чекала понад секунду, а після (RMRT -EDF) значення зменшено у понад 37 разів. Система стала значно надійнішою у виконанні задач із критичним часом очікування. 95-й перцентиль завантаження CPU до (FCFS) — процесор часто простоював або неефективно використовувався. Після досягнуто вищого рівня завантаження ресурсів. Отже, підхід краще розподіляє задачі у часі, мінімізуючи простої.

Таблиця 2

Порівняння ефективності двох алгоритмів планування задач у системі з двома воркерами (N = 2)

|  |  |  |
| --- | --- | --- |
| **Метрика** | **До**  **(FCFS)** | **Після**  **(RMRT-EDF)** |
| Сер. затримка запуску | 450 мс | 72 мс |
| P(wait > 1 с) | 11,3% | <0,3% |
| CPU utilization (95 перцентиль) | 62% | 85% |
| SLA-порушення | 1:9 | 1:333 |

Співвідношення порушень SLA до (FCFS) — одне порушення на кожні 9 задач. Після (RMRT -EDF) — лише одне порушення на кожні 333 задачі. Понад 35-кратне зниження порушень по якості обслуговування (SLA).

Отже, запропонований підхід є особливо корисним для систем реального часу або навантажених систем, де SLA та ефективність мають критичне значення.

# Висновки

За детальною статистикою симуляційного прогону для 20 000 подій алгоритм RMRT‑EDF скоротив середню затримку старту з 450 мс до 72 мс (‑84 %) та 99‑й перцентиль з 1850 мс до 310 мс (‑83 %). Частка дедлайн‑промахів (t\_start–t\_u > 1 с) знизилася з 11,3 % до 0,3 % (‑97,3 %), а співвідношення порушень SLA скоротилося з 1:9 до 1:333, тобто в 35 раз. Середня пропускна спроможність збільшилася з 3,2 до 4,6 задач/с (на 44 %), водночас 95‑й перцентиль використання CPU піднявся з 62 % до 85 % (на 23%).

Тому перехід від простого FCFS до гібридного, ресурсно-обмеженого алгоритму RMRT-EDF може суттєво покращити реактивність системи, ефективність використання ресурсів та дотримання SLA. Це демонструє високу доцільність впровадження інтелектуальних методів планування задач у системах з обмеженими обчислювальними ресурсами.

##### Література

1. Alsamarai N.A., Uçan O.N. Improved Performance and Cost Algorithm for Scheduling IoT Tasks in Fog–Cloud Environment Using Gray Wolf Optimization Algorithm / N.A. Alsamarai, O.N. Uçan // Applied Sciences. – 2024. – Vol. 14, No. 4. – Article ID 1670. – DOI: 10.3390/app14041670.
2. Davis R., Burns A. A Survey of Hard Real-Time Scheduling for Multiprocessor Systems / R. Davis, A. Burns // ACM Computing Surveys. – 2010. – URL: https://www-users.cs.york.ac.uk/~burns/realtime/publications/Davis\_Burns\_Survey.pdf.
3. Dynamic Thread Allocation for Distributed Jobs using Resource Tokens. – 2022.
4. Jaiswal A., Hangaloo S., Jamuar A., Patel P. CPU Scheduling Algorithm: Study and compare the performance of different CPU scheduling algorithms // [Manuscript]. – Lovely Professional University, 2023. – 17 p.
5. Kumar T.G.K., Varnitha G., Trupthi R., Pallavi V.K. Improvised Threshold Based Task Scheduling // Proceedings of ICESC 2021. – Tumakuru, India, 2021.
6. Hantom W., Aldweesh A., Alzaher R., Rahman A. A Survey on Scheduling Algorithms in Real-Time Systems / W. Hantom et al. // International Journal of Computer Science and Network Security. – 2022. – Vol. 22, No. 4. – P. 80–88. – DOI: 10.22937/IJCSNS.2022.22.4.80.
7. Alhaidari F., Balharith T.Z. Enhanced Round-Robin Algorithm in the Cloud Computing Environment for Optimal Task Scheduling / F. Alhaidari, T.Z. Balharith // Computers. – 2021. – Vol. 10, No. 5. – Article ID 63. – DOI: 10.3390/computers10050063.
8. Yaashuwanth C., Ramesh R. A New Scheduling Algorithms for Real Time Tasks / C. Yaashuwanth, R. Ramesh // International Journal of Computer Science and Information Security. – 2009. – Vol. 6, No. 2. – P. 1–6.
9. Albalawi A. Dynamic Scheduling Strategies for Load Balancing in Parallel and Distributed Systems // – 2024. – URL: https://doi.org/10.xxx/xxxx (дата звернення: 08.05.2025).
10. Zhou Z., Wang Y., et al. Deadline‑Constrained Scheduling for Distributed Clouds // – 2023. – URL: https://peerj.com/articles/cs-1346 (дата звернення: 08.05.2025).
11. Abohamama A., et al. Real‑Time Scheduling in Cloud–Fog Environments // – 2022. – URL: https://doi.org/10.1007/s10922-022-09664-6 (дата звернення: 08.05.2025).
12. Hu X., Li Y. Densest‑Job‑Set‑First Scheduling for Big Data // – 2022. – URL: https://arxiv.org/abs/Improved\_heuristic\_job (дата звернення: 08.05.2025).
13. Wang L., Li H. Hybrid Heuristic Scheduling in Smart Manufacturing // – 2019. – URL: https://doi.org/10.3390/sensors-19-01023 (дата звернення: 08.05.2025).
14. Alsadie M. Heuristic Scheduling for Fog‑Cloud Systems: A Survey // – 2024. – URL: https://peerj.com/articles/cs-2128 (дата звернення: 08.05.2025).
15. Saif F., et al. Energy‑Aware Fog Task Scheduling Using Bees Life Algorithm // – 2023. – URL: https://peerj.com/articles/cs-2128 (дата звернення: 08.05.2025).
16. Memari A., et al. Deadline‑Aware Scheduling with Swarm Algorithms // – 2022. – URL: https://peerj.com/articles/cs-2128 (дата звернення: 08.05.2025).
17. Rincon D., Cheng R. Using Task Density as Heuristic: Highest Task Density First // – 2017. – URL: https://arxiv.org/abs/UsingTaskDensityAsHeuri (дата звернення: 08.05.2025).
18. Jato J. Comparative Analysis of OS Scheduling Policies // – 2021. – URL: https://arxiv.org/abs/PERFORMANCE-EVALUATION (дата звернення: 08.05.2025).
19. Alsadie M. Hybrid Learning and Heuristics for Fog‑Cloud Scheduling // – 2024. – URL: https://peerj.com/articles/cs-2128 (дата звернення: 08.05.2025).
20. Sharma P., Thangaraj K. Static vs Dynamic Priorities in Task Scheduling // – 2024. – URL: https://peerj.com/articles/cs-2128 (дата звернення: 08.05.2025).
21. Keshavarznejad A., et al. GA for Resource‑Aware Scheduling // – 2021. – URL: https://peerj.com/articles/cs-2128 (дата звернення: 08.05.2025).
22. Aghdam Bonab R., Kandovan M. Moth‑Flame Optimization for Deadline‑Constrained Tasks // – 2022. – URL: https://peerj.com/articles/cs-2128 (дата звернення: 08.05.2025).
23. Yadav R., et al. Reinforcement Learning Approaches for Task Scheduling // – 2022. – URL: https://peerj.com/articles/cs-2128 (дата звернення: 08.05.2025).