УДК 004.03:004.45:004.75
DOI https://doi.org/10.32782/2663-5941/2025.5.2/18

Козельський О.В.
Хмельницький національний університет

Савенко Б.О.
Хмельницький національний університет

ЗОВНІШНЯ АДАПТИВНА СИСТЕМА КЛАСТЕРИЗАЦІЇ ДЛЯ ПІДВИЩЕННЯ ГНУЧКОСТІ ОПЕРАЦІЙНИХ СИСТЕМ РЕАЛЬНОГО ЧАСУ НА ОСНОВІ ДИНАМІЧНОГО ПЛАНУВАННЯ ЗАВДАНЬ

У роботі досліджено проблему забезпечення стабільної та передбачуваної роботи операційних систем реального часу (ОСРЧ) у кіберфізичних середовищах із високим рівнем динаміки та змінюваними умовами функціонування. Виявлено обмеження традиційних механізмів планування задач, що проявляються в умовах пікових навантажень, виникнення пріоритетних конфліктів, нестачі обчислювальних ресурсів та збоїв у системі. Для усунення цих недоліків запропоновано новий підхід, заснований на використанні адаптивного кластеризатора, який дозволяє групувати задачі за критеріями важливості, поточного навантаження на ресурси та взаємозв'язків між процесами. Ключовим елементом архітектури є система адаптивного планування з урахуванням динамічних змін пріоритетів, що забезпечує своєчасний перерозподіл задач залежно від поточного стану системи. Запропоноване рішення дозволяє уникати перевантажень окремих модулів, мінімізувати затримки виконання критично важливих процесів, знижувати час реакції на непередбачені події та забезпечувати безперервність роботи ОСРЧ навіть у разі часткових збоїв. Архітектура передбачає гнучку взаємодію між обчислювальними вузлами, використання механізмів зворотного зв'язку та автоматичну корекцію планів виконання при зміні вхідних умов. Розроблено метод функціонування кластеризатора та математичну модель, що описує зміну станів задач у реальному часі з урахуванням імовірнісних характеристик. Проведено порівняльний аналіз із класичними підходами, результати якого демонструють підвищення загальної надійності системи, стабільності виконання задач високого пріоритету та ефективності використання ресурсів. Запропонована концепція може бути застосована в автономному транспорті, промисловій автоматизації, робототехнічних комплексах і критично важливих керуючих системах, що вимагають високої відмовостійкості та точності роботи.

Ключові слова: *операційні системи, кіберфізичні системи, кластеризація задач, ОСРЧ, балансування навантаження, планувальник задач, відмово стійкість.*

Постановка проблеми. Кіберфізичні системи з інтенсивною зміною станів спираються на операційні системи реального часу (ОСРЧ), покликані забезпечувати детерміноване дотримання часових обмежень за умов невизначеності параметрів навантаження, подій та переривань. Поширені політики планування, сконструйовані під стаціонарні припущення, демонструють деградацію якості сервісу за пікових впливів, збурень, зміни критичності та корельованості між задачами [1]. Це призводить до збільшення відхилень фактичного часу початку або завершення виконання завдань від запланованих моментів, зростання кількості випадків несвоєчасного завершення завдань, виникнення непрогнозованих ефектів витіснення та втрат ефективності використання центрального процесора, а також до тимчасової його монополізації високопріоритетними завданнями в умовах аварійних сценаріїв. Вбудовані модулі машинного навчання (ML), розміщені на бортових мікроконтролерах (MCU), конкурують за обмежені обчислювальні й енергетичні бюджети з реальточасовими потоками, що підвищує ризик порушення часових гарантій [2]. Сучасні RTOS, як правило, не містять повноцінно інтегрованого безперервного моніторингу безпеки та механізмів самовідновлення з формалізованими часовими гарантіями; за навмисних впливів і аномалій погіршуються спостережуваність і резилієнтність. Архітектури залишаються

переважно монолітними: модифікація політик або включення аналізу стану потребують глибоких змін ядра, тоді як інтерфейси для зовнішніх інтелектуальних модулів відсутні або не забезпечують контрольованих накладних витрат і латентностей. Таким чином, наявний методологічний розрив між вимогою гнучкого, масштабованого керування в умовах динаміки та фактичними можливостями існуючих рішень. Наукова проблема полягає у браку принципів синтезу архітектури ОСРЧ, здатної гарантувати часові вимоги в нестаціонарних профілях навантаження, зберігати ефективність та інтегрувати моніторинг і захист без руйнування реальточасових властивостей. Необхідно формально визначити обмеження класичних політик, релевантні критерії якості та умови, за яких адаптивні механізми розподілу й пріоритизації завдань у поєднанні зі спостереженням станів мінімізують джитер і пропуски термінів та підвищують утилізацію ресурсів.

Аналіз останніх досліджень і публікацій. Кластеризація в операційних системах реального часу (ОСРЧ) є методом групування даних або об'єктів на основі схожих характеристик чи властивостей. У цій галузі використовується низка різноманітних методів, кожен із яких має свої переваги та недоліки. Основні категорії включають часові методи, методи машинного навчання, генетичні алгоритми та підходи на основі графів. Кожен із цих підходів пропонує унікальні рішення для задач кластеризації та має специфічні сфери застосування.

Часове розділення (Time Division Multiplexing, TDM) використовується для ефективного управління ресурсами в ОСРЧ шляхом виділення певних часових інтервалів для різних завдань [4, 5, 6] Це забезпечує передбачуваний і регулярний доступ до ресурсів, що є критично важливим для підтримки властивостей систем реального часу [3]. У TDM кожне завдання або група завдань отримує ексклюзивний доступ до ресурсів процесора протягом заздалегідь визначеного інтервалу часу в циклічному порядку. Такий підхід підтримує високу продуктивність і надійність системи, гарантуючи регулярне виконання завдань [6].

Можна позитивно оцінити, що кожне завдання виконується у визначений час, яке забезпечує регулярність роботи системи. Наприклад, у виробничих процесах TDM може виділяти фіксовані часові інтервали різним датчикам і механізмам, запобігаючи конфліктам і забезпечуючи безперебійну роботу. Простота реалізації цього методу дозволяє легко його впроваджувати та налаштовувати, що є важливим у вбудованих системах. Крім того, TDM сприяє ефективному використанню ресурсів через контрольоване розподілення часових інтервалів, оптимізуючи їхнє використання. Також необхідно враховувати, якщо завдання не використовує весь виділений йому часовий інтервал, невикористаний ресурс втрачається, що призводить до неефективності. У системах зі змінним навантаженням фіксовані часові інтервали можуть бути неефективними, оскільки не адаптуються до поточних потреб [7, 8]. Масштабованість також стає проблемою: у великих системах з багатьма завданнями складно забезпечити достатній часовий проміжок для кожного завдання. Оптимізація використання ресурсів стає критичною, і виникає потреба в методах динамічного перерозподілу невикористаних ресурсів між завданнями. Врахування пріоритетів завдань також важливе, щоб забезпечити своєчасне виконання критичних завдань, особливо в системах, де це може вплинути на безпеку або ефективність [9].

Методи машинного навчання (ML) дедалі частіше застосовуються для кластеризації та балансування навантаження в ОСРЧ завдяки здатності автоматично виявляти оптимальні шаблони та швидко адаптуватися до змінних умов. Роботи [11, 12] показують, що алгоритми deep reinforcement learning (DRL) дозволяють підвищити пропускну здатність і скоротити затримку під час розподілу задач у реальному часі. Прогностичні моделі на основі LSTM забезпечують точне передбачення піків навантаження та динамічне коригування використання процесорного часу [13]. У сукупності це веде до ефективнішого розподілу ресурсів і стабільнішої роботи системи навіть у мінливих умовах. Попри переваги, ML-підходи вимагають великих обсягів якісних даних і суттєвих обчислювальних ресурсів**.** Це особливо критично для вбудованих платформ з обмеженнями енергоспоживання. Дослідження [14, 15] відзначають, що складність нейронних мереж ускладнює їх інтеграцію в ОСРЧ, потребує експертизи для налаштування та породжує питання прозорості й довіри до отриманих рішень. Саме тому актуальними залишаються теми легковагових моделей, знань-дистиляції та винесення ресурсомістких обчислень на зовнішні модулі.

Генетичні алгоритми (ГА), засновані на принципах природного відбору, широко використовуються для вирішення оптимізаційних задач, таких як кластеризація та балансування навантаження в ОСРЧ. Вони використовують процеси відбору,

схрещування та мутації для пошуку оптимальних або близьких до оптимальних рішень складних проблем. ГА можуть ефективно справлятися з різними типами завдань і обмеженнями ресурсів, знаходячи глобальні оптимуми та адаптуючись до різних проблем без значних змін алгоритму. Вони також стійкі до змін та невизначеностей у системі, що дозволяє підтримувати стабільність та продуктивність в динамічних умовах. Однак генетичні алгоритми мають високу обчислювальну складність та потребують ретельного налаштування параметрів, таких як розмір популяції та частоти кросинговерів і мутацій. У контексті ОСРЧ ключовою вимогою є не лише якість планувального рішення, а його детермінованість і гарантований найгірший час обчислення (WCET). Збіжність ГА є стохастичною і залежить від розміру популяції та ймовірностей операцій, тож верхня межа часу на отримання прийнятної відповіді не є стабільною, що конфліктує з жорсткими та слабкожорсткими часовими обмеженнями. За динамічних профілів навантаження функція придатності швидко застаріває, і значна частина ітерацій витрачається на переоцінку популяції, підвищуючи нестабільність часу відгуку керувальних дій [16]. Крім того, масштабованість погіршується зі зростанням числа задач/обмежень. Витрати на оцінку фітнесу та еволюційні операції зростають щонайменше лінійно, а часто суперлінійно – що погано сумісно з обмеженнями мікроконтролера за тактом, енергією та пам'яттю [18]. Чутливість до гіперпараметрів потребує автоналаштування, яке саме є ресурсомістким. Нарешті, верифікація та сертифікація стохастичних процедур у безпеково-критичних системах ускладнені: відтворюваність і формальні докази коректності гірші, ніж у детермінованих методів. Повільна збіжність може бути проблемою в системах реального часу, де своєчасні рішення є критичними. Масштабованість також викликає занепокоєння, оскільки зі збільшенням розміру системи зростають обчислювальні вимоги. Визначення ефективних критеріїв завершення алгоритму та обробка динамічних змін у системі залишаються викликами.

Методи на основі графів використовують теорію графів для моделювання взаємодії між завданнями та ресурсами в ОСРЧ. Завдання та ресурси представляються як вузли графа, а зв'язки між ними – як ребра. Це дозволяє ефективно розподіляти та керувати завданнями, використовуючи алгоритми спектральної кластеризації, мінімального розрізу/максимального потоку та розбиття графів. Ці методи здатні моделювати складні взаємодії та залежності між завданнями і ресурсами, що сприяє точнішій кластеризації та балансуванню навантаження. Вони добре масштабуються і можуть працювати з великими системами [20], що є важливим для ОСРЧ з великою кількістю завдань. Гнучкість в оптимізації дозволяє адаптувати ці методи до різних цілей, таких як мінімізація витрат на зв'язок або рівномірний розподіл навантаження. Але слід враховувати, що методи мають високу обчислювальну складність, оскільки побудова та обробка великих графів вимагає значних ресурсів. Це може бути проблемою для ОСРЧ з обмеженими можливостями та суворими часовими обмеженнями. Складність обробки динамічних змін також є викликом, оскільки ці методи часто припускають статичну структуру графа. Необхідність точної інформації про залежності між завданнями та ресурсами може ускладнювати їх застосування у складних системах, де такі залежності не завжди чітко визначені. Масштабованість залишається проблемою, оскільки вимоги до обчислень зростають із розміром та складністю системи. Розробка більш ефективних алгоритмів та використання методів паралельної обробки може допомогти вирішити ці питання.

Отже, огляд наявних способів кластеризації в ОС реального часу свідчить, що жоден із них не є універсальним – кожен вирізняється власними сильними і слабкими сторонами. Вибір найдоречнішого підходу диктується специфікою системи: її масштабом, складністю та ресурсними обмеженнями. Подальші дослідження зосереджуються на усуненні наявних недоліків і створенні більш ефективних, адаптивних і масштабовних рішень для ОСРЧ [17, 19].

Перспективним виглядає застосування гібридних схем [21, 22], що поєднують переваги різних методів [23], а також розроблення нових алгоритмів, здатних стабільно працювати у динамічних і складних середовищах.

Метою статті є обґрунтування та розробка архітектури операційної системи реального часу, що поєднує адаптивні алгоритми машинного навчання для динамічної кластеризації та балансування навантаження, а також інтегрує механізми протидії зловмисному ПЗ. Запропоноване рішення має забезпечити підвищену ефективність, гнучкість і відмовостійкість ОСРЧ в умовах обмежених ресурсів і швидких змін робочих параметрів.

Виклад основного матеріалу. Розглянемо модель архітектури планувальника задач з адап-

тивною кластеризацією базується на принципі збирання наборів даних, які відправляються у окрему систему, яка аналізує дані і на основі цього намагається покращити ефективність управління ресурасми в майбутньому.

Початковим етапом розроблення нового підходу є формування комплексної бази даних, що відображає повну картину роботи системи. До неї входять детальні відомості про кожне завдання – від точних часових характеристик його виконання до показників споживання ресурсів і заданих пріоритетів. Поряд із цим фіксуються взаємозв'язки між завданнями, залежності від зовнішніх та внутрішніх подій, а також інші параметри, здатні впливати на поведінку системи. Такий масив даних дає змогу відобразити не лише статичні вимоги кожного процесу, але й динаміку його роботи в різних режимах, що є ключовим для подальшого аналізу та оптимізації планувальника.

Нехай $X = \{x_1, x_2, \dots, x_n\}$ – множина задач, а $R = \{r_1, r_2, \dots, r_m\}$ – множина ресурсів. Кожна задача $x_i \in X$ описується набором ознак:

$$F_i = \{f_1, f_2, \dots, f_k\}, \quad (1)$$

де f_j відповідає певному параметру, наприклад часу виконання, рівню споживання ресурсів, пріоритету чи наявності залежностей. Таким чином, для кожної задачі x_i формується набір ознак:

$$F_i = \{f_1(x_i), f_2(x_i), \dots, f_k(x_i)\}, \forall x_i \in X \quad (2)$$

Тут $f_j(x_i)$ – значення j-ої ознаки для задачі x_i, а k – кількість ознак, що описують задачу, x_i – i-та задача з множини задач X.

Для підвищення коректності роботи алгоритмів машинного навчання проводиться нормалізація векторів ознак F_i. Позначимо через $\tilde{F}_i$ нормалізований вектор:

$$\tilde{F}_i = \left(\frac{f_1(x_1) - minf_1}{max\ f_1 - minf_1}, \dots, \frac{f_k(x_i) - minf_k}{max\ f_k - minf_k}\right) \quad (3)$$

Тут $minf_i$ та $max\ f_i$ – відповідно мінімальні та максимальні значення j-ої ознаки серед усіх задач у вибірці. Процедура нормалізації використовується для приведення різних ознак до єдиної шкали, що забезпечує стабільність та коректність роботи алгоритмів.

Далі з множини ознак F_i відбирається підмножина $S \subset F$, яка має найбільший вплив на продуктивність системи. Позначимо цю множину як $S = \{s_1, s_2, \dots, s_p\}$, де кожен елемент s_j – вибрана ознака з множини F_i, причому $p < k$.

Подальша обробка передбачає кластеризацію задач за обраними ознаками. Для цього можуть застосовуватися алгоритми типу k-середніх або ієрархічна кластеризація[10]. Процес кластеризації формально описується функцією належності задачі x_i до кластера C_j:

$$\forall x_i \in X, x_i \to C_j, \text{де } j = arg\ \min_j d(\tilde{F}_i, \mu_j) \quad (4)$$

де $d(\cdot,\cdot)$ – функція обчислення відстані між векторами ознак, а μ_j – центр кластера C_j.

Результатом кластеризації є множина кластерів $C_1, C_2, \dots, C_t$, на основі якої навчається модель машинного навчання M. Ця модель використовує ознаки з множини S для віднесення нової задачі x_{new} до відповідного кластера:

$$M(x_{new}) = C_j \quad (5)$$

де x_{new} – нова задача, а C_j – кластер, до якого її віднесено.

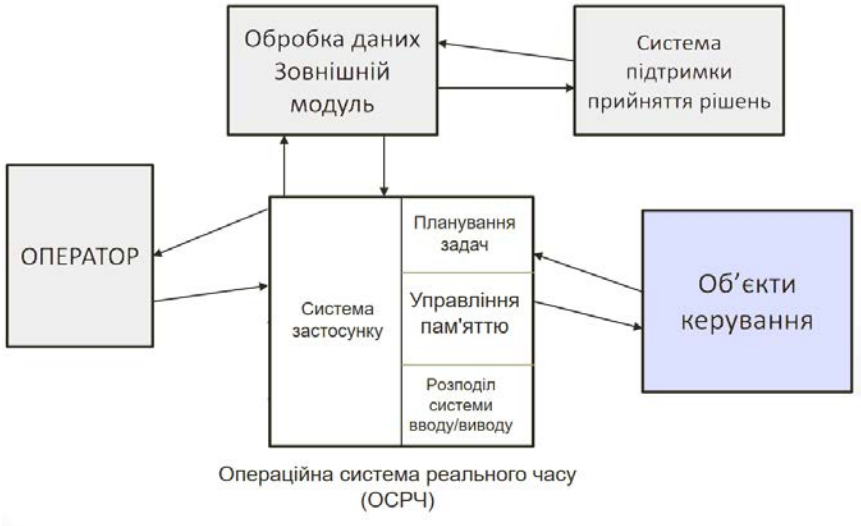

Рис. 1. Схема побудови системи з адаптивною кластеризацією та зовнішнім модулем

Побудована модель дозволяє прогнозувати майбутні потреби в ресурсах на основі поточного стану системи. Функція прогнозування $P(t)$ описує вектор прогнозованих навантажень:

$$P(t) = \{r_1(t), r_2(t), \dots, r_m(t)\}, \quad (6)$$

де $r_j(t)$ – прогнозоване навантаження на ресурс r_j у момент часу t.

Для визначення оптимального розподілу задач використовується функція мінімізації відхилень між фактичними $r_j^{fact}(t)$та оптимальними прогнозованими $r_j^{opt}(t)$ навантаженнями:

$$min\sum_{j=1}^{m}\left(r_j^{fact}(t) - r_j^{opt}(t)\right)^2 \quad (7)$$

Запропонована методологія спрямована на забезпечення рівномірного розподілу обчислювального навантаження між доступними ресурсами, що уможливлює ефективне балансування та пріоритетне виконання завдань із високим рівнем критичності. Такий підхід сприяє підвищенню продуктивності системи та зменшенню ймовірності виникнення затримок у роботі. Механізм прогнозування навантаження функціонує в режимі реального часу та динамічно перерозподіляє задачі між процесорами чи обчислювальними ядрами, запобігаючи утворенню перевантажених ділянок («вузьких місць») і забезпечуючи стабільність функціонування. Збір даних про стан виконання завдань і використання ресурсів здійснюється паралельно з їх виконанням, без негативного впливу на швидкодію системи. Обробка цих даних передається у відокремлений модуль або сервіс, де періодично проводиться аналітична обробка та оновлення моделей машинного навчання для підтримання їхньої актуальності. Результати аналізу інтегруються у процес планування в реальному часі, що дає змогу оптимізувати розподіл ресурсів та забезпечити своєчасне виконання завдань.

Для реалізації прототипу пропонується використати операційну систему FreeRTOS з модифікованими файлами tasks.c та main.c. У файлі tasks.c, який є основним для планувальника FreeRTOS і містить функції управління задачами, ми змінюємо функцію vTaskSwitchContext(), щоб вона викликала власний метод для розподілу задач. У файлі main.c додаємо код для створення задач збору даних та виклику кастомного планувальника, який буде реалізований у новому файлі custom_scheduler.c. У цьому файлі визначатимуться функції для збору даних, класифікації задач та оптимізації їхнього розподілу.

Алгоритм складається з наступних кроків:

Крок 1. Збір даних у FreeRTOS**.** Збираємо дані шляхом запису параметрів під час роботи системи або через моделювання. Визначаємо основні параметри, які будуть використовуватися в моделі, наприклад: тип задачі, використання ресурсів (CPU), час виконання задачі, пріоритет задачі. На основі цього створюємо програмну модель. Імітуємо виконання задач у системі, записуючи параметри під час їхнього виконання. Це можна реалізувати за допомогою циклів та таймерів для моделювання реального часу. Зібрані дані аналізуємо та перетворюємо у потрібний формат для подальшого використання в навчанні моделей машинного навчання. Передаємо дані з мікроконтролера через налаштований HTTP-клієнт на сервер в Інтернеті, створений на C# .NET 8, який приймає ці дані.

Крок 2. Підготовка даних для моделювання. На основі зібраних даних формуємо набір, який містить інформацію про задачі, їхні характеристики та часові параметри. Використовуємо зовнішню математичну бібліотеку Math.NET Numerics для обробки та підготовки даних до відправки на сервіс TinyML.

Крок 3. Відправка даних на навчання. За допомогою .NET-сервісу конвертуємо дані у формат, сумісний з TinyML, відправляємо їх і отримуємо навчену модель.

Крок 4. Аналіз навченої моделі та прогнозування**.** Після успішного навчання моделі на сервері, наступним кроком є її використання для виконання прогнозів. У нашому випадку модель буде застосовуватися для виконання прогнозування з метою оптимізації розподілу задач.

Крок 5. Відправка результатів на мікроконтролер**.** Після виконання прогнозування результати відправляємо назад на мікроконтролер для оновлення процесу прийняття рішень у файлі custom_scheduler.c.

Процес збору даних відбуватиметься кожний проміжок часу заданий в константах та автоматично повторюватиметься.

Тепер порівняємо ефективність в результатах експериментів. FreeRTOS використовує принцип управління часом через свій планувальник завдань (scheduler)**,** який дуже схожий на Часове розділення (Time Division Multiplexing, TDM). Ми можемо порівняти результати роботи FreeRTOS за замовчуванням і результати роботи FreeRTOS із модифікованим планувальником задач.

Для порівняння ефективності стандартного FreeRTOS і FreeRTOS з модифікованою архітектурою використаємо кілька ключових метрик.

Задача. Це буде процес обробки даних, введення/виведення, обчислювальна задача або просто фонове завдання.

Використання ресурсів (CPU). Вимірювання використання CPU під час виконання задачі.

Час виконання задачі. Вимірювання часу, який потрібен для виконання задачі.

Спочатку запустмо програму з використанням стандартного планувальника задач і зберемо дані про час виконання цих задач, використання ресурсів, пропускну здатність і затримку за допомогою логування. Після цього проаналізуємо роботу планувальника задач у FreeRTOS з модифікованою архітектурою. Для статистики кількість задач зробимо 4. Підключимо програму DSView, стандартно працююча з логічним аналізатором від DS Logic.

За допомогою логічного аналізатора DSLogic Pro у програмі DSView v1.1.2 побудуємо графік, який відображає часові діаграми виконання завдань у системі реального часу, де кожен канал відповідає окремому процесу або сигналу. Отримані часові діаграми відображають періоди активності кожної задачі, дозволяючи візуалізувати розподіл процесорного часу та оцінити ефективність роботи планувальника. Виходячи із результатів роботи звичайного планувальника задач FreeRTOS і модифікованого (рисунок 2) можна зробити деякі заключення. У звичайному планувальнику задачі 1, 3, і 4 мають рівнозначний пріоритет, що видно з їх рівномірного розподілу на графіку. Вони працюють паралельно або по черзі, але без чіткої пріоритетності. Задача 2 має високий пріоритет, і вона виконується першою після кожного переривання, яке генерується «тиканням» системного таймера (Tick). Ідл (Idle) процес не виконується часто, тому що процесор постійно зайнятий виконанням задач.

Модифікований планувальник має пріоритетне розподілення задач. На цьому графіку (рис. 3) видно, що модифікований планувальник впроваджує розподіл задач на основі їхніх пріоритетів та часу виконання. Відбувається чіткий поділ

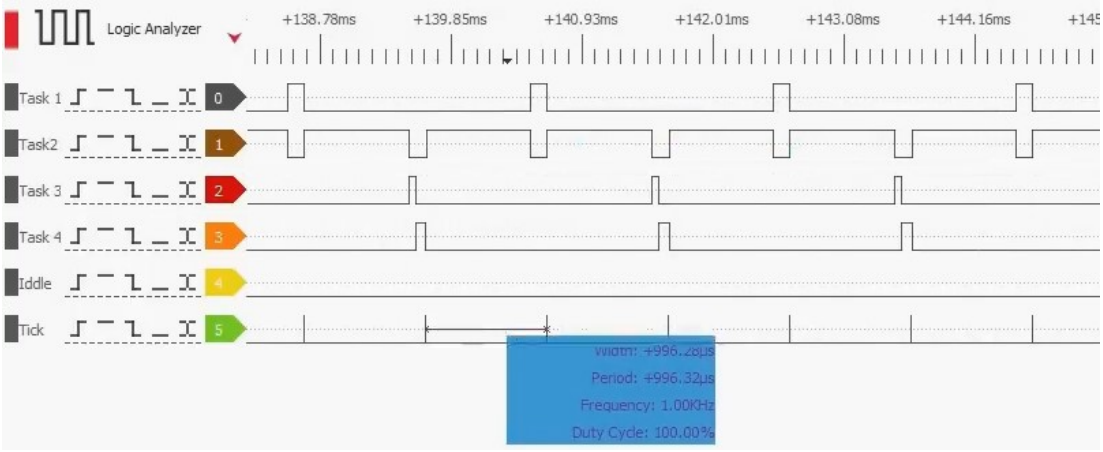

Рис. 2. Графічне відображення часу виконання задач стандартним планувальником FreeRTOS

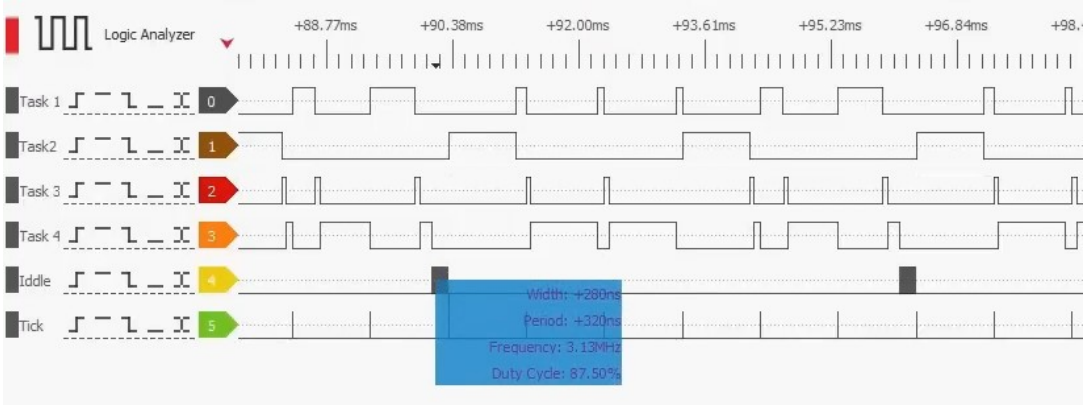

Рис. 3. Графічне представлення часу виконання задач модифікованим планувальником FreeRTOS

часу між задачами. Тут спостерігається більш кероване та структуроване використання процесорного часу. Спостерігається попереджуюче планування. Задачі перериваються іншими більш пріоритетними завданнями, що демонструє поведінку попереджуючого планувальника, коли виконання низькопріоритетних задач призупиняється для виконання важливіших задач. Ідл (Idle) процес простою тепер активніше працює, коли інші задачі завершують свій час виконання або знаходяться в очікуванні. Це вказує на те, що система працює ефективніше, оскільки процесорна потужність використовується більш рівномірно.

У звичайному планувальнику ресурси використовуються не повністю оптимально, тому що задачі з рівнозначними пріоритетами можуть переривати одна одну, а задача з високим пріоритетом повністю займає процесорний час.

У модифікованому планувальнику чітко впроваджується пріоритетне попереджуюче планування, яке дозволяє системі динамічно розподіляти час виконання задач, а також більш ефективно використовувати процесорну потужність, дозволяючи системі швидше реагувати на зміни.

Висновки. У проведеній роботі здійснено детальний аналіз існуючих методів кластеризації в операційних системах реального часу (ОСРЧ), зокрема часових методів, методів машинного навчання, генетичних алгоритмів та підходів на основі графів. Встановлено, що кожен із цих методів має свої переваги та недоліки, які обмежують їх ефективність, гнучкість та масштабованість у сучасних динамічних системах.

Для подолання виявлених обмежень було розглянуто адаптивну кластеризацію, яка базується на використанні методів машинного навчання на зовнішніх модулях. Цей підхід передбачає збір та аналіз даних про задачі та ресурси системи, класифікацію задач за їхніми характеристиками та оптимізацію розподілу ресурсів на основі прогнозування моделей машинного навчання.

Експериментальні результати показують покращення продуктивності, зокрема, зменшилося використання CPU, і скоротився час виконання задач у порівнянні з класичним планувальником FreeRTOS.

Таким чином, адаптивна кластеризація демонструє перспективність використання методів машинного навчання, а перенесення логіки аналізу на зовнішні модулі для управління ресурсами та балансування навантаження в операційних системах реального часу не навантажує без того обмежені ресурси. Подальші дослідження можуть бути спрямовані на вдосконалення моделей машинного навчання, оптимізацію обчислювальної складності алгоритмів, а також адаптацію системи до динамічних змін та масштабування для роботи в більш складних та навантажених середовищах. Робота над збільшенням ефективності розподілу процесорного часу продовжується.

Список літератури:

1. Salamun K. Weakly Hard Real-Time Model for Control Systems: A Survey. Sensors. 2023. 23(10). 4652. MDPI, Basel, Switzerland. P. 1–33. DOI: 10.3390/s230104652.

2. Malik S. An Adaptive Emergency First Intelligent Scheduling. Sustainability. 2019. 11(8). 2192. MDPI, Basel, Switzerland. P. 1–19. DOI: 10.3390/su11082192.

3. Zhang L., Li W., Wu Y., Wang X., Park S.-I., Kim H. M., Lee J.-Y., Angueira P., Montalban J. Layered-Division Multiplexing: Theory and Practice. IEEE Transactions on Broadcasting. 2016. 62(1). 216–232. https://doi.org/10.1109/TBC.2015.2505408.

4. Park S.-I. ATSC 3.0 for Future Broadcasting: Features and Extensibility. *SET International Journal of Broadcast Engineering*. 2020. 6. 21–36. Retrieved August 7, 2025, from https://revistaeletronica.set.org.br/ijbe/article/view/203.

5. Lee J., Park S.-I., Yim H.-J., Lim B.-M., Kwon S., Ahn S., Hur N. IP-Based Cooperative Services Using ATSC 3.0 Broadcast and Broadband. IEEE Transactions on Broadcasting. 2020. Vol. 66 (2). P. 440–448. DOI: 10.1109/TBC.2020.2983301.

6. Yamamoto H., Nakamura A., Itami M. A Study on LDM-BST-OFDM Transmission for the Next-Generation Terrestrial Broadcasting. IEEE Transactions on Broadcasting. 2020. Vol. 66 (2). P. 205–215. DOI: 10.1109/TBC.2019.2932340.

7. Sparsø J., Damsgaard H. J., Katsamanis D., Schoeberl M. Comparing Timed-Division Multiplexing and Best-Effort Networks-on-Chip. Journal of Systems Architecture. 2022. Vol. 133. Article 102766. DOI: 10.1016/j.sysarc.2022.102766.

8. Alkoudsi M. I., Fohler G. Scheduling Dynamic Task-Sets in Time-Triggered Real-Time Systems. In: Proceedings of the 32nd International Conference on Real-Time Networks and Systems (RTNS 2024). Porto, Portugal. 2024. P. 48–58. DOI: 10.1145/3696355.3699699.

9. Yari Karin S., Aydin H., Zhu D., Drager S. Impact of Priority Assignment on Schedule-Based Attacks in Real-Time Embedded Systems. Journal of Systems Architecture. 2023. Vol. 145 (6). Article 103021. DOI: 10.1016/j.sysarc.2023.103021.

10. Jana D. P., Shukla A. M., Gupta S. K-Means Algorithm-Based Detection for Wavelength-Division-Multiplexed OOK PD-NOMA System over Turbulent Optical Channel. Optical Engineering. 2022. Vol. 61(3). Art. 036111. DOI: 10.1117/1.OE.61.3.036111.

11. Wang J., Li S., Zhang X., Wu F., Xie C. Deep Reinforcement Learning Task Scheduling Method Based on Server Real-Time Performance. PeerJ Computer Science. 2024. Vol. 10. Article e2120. DOI: 10.7717/peerj-cs.2120.
12. Paduraru C., Patilea C. C., Iordache S. Task Scheduling: A Reinforcement Learning Based Approach. In: 15th Int. Conf. on Agents and Artificial Intelligence (ICAART 2023). Lisbon, Portugal. 2023. P. 948–955. DOI: 10.5220/0011826100003393.
13. Qiu J., Lin Y., Zwetsloot I. M. An LSTM-Based Predictive Monitoring Method for Data with Time-Varying Variability. International Journal of Production Research. 2025. Vol. 63(7). P. 2622–2637. DOI: 10.1080/00207543.2024.2408435.
14. Stefani D., Peroni S., Turchet L. A Comparison of Deep Learning Inference Engines for Embedded Real-Time Audio Classification. In: Proc. 25th Int. Conf. on Digital Audio Effects (DAFx20in22). Vienna, Austria. 2022. P. 256–263. DOI: 10.5281/zenodo.7314918.
15. Delgadillo O., Blieninger B., Kuhn J., Baumgarten U. An Architecture to Enable ML-Based Task Migration for Multi-Core RTOS. In: IEEE 14th Int. Symp. on Embedded Multicore/Many-core SoC (MCSoC 2021). Singapore. 2021. P. 405–412. DOI: 10.1109/MCSoC51149.2021.00066.
16. Hassan H.E., Ibrahiem K.H., Madian A.H. Optimizing Multiprocessor Performance in Real-Time Systems Using an Innovative Genetic Algorithm Approach. Scientific Reports. 2025. Vol. 15. Article 3842. DOI: 10.1038/s41598-024-80910-4.
17. Sun B., Theile M., Qin Z. et al. Edge Generation Scheduling for DAG Tasks Using Deep Reinforcement Learning. IEEE Transactions on Computers. 2024. Vol. 73(4). P. 912–925. DOI: 10.1109/TC.2024.3350243.
18. Gholamrezanejad M., Shahhoseini H.S., Mohtavipour S.M. A Customized Balanced-Objective Genetic Algorithm for Task Scheduling in Reconfigurable Computing Systems. Knowledge and Information Systems. 2025. Vol. 67(2). P. 1541–1571. DOI: 10.1007/s10115-024-02268-3.
19. Sun B., Theile M., Qin Z. et al. Edge Generation Scheduling for DAG Tasks Using Deep Reinforcement Learning. IEEE Transactions on Computers. 2024. Vol. 73(4). P. 912–925. DOI: 10.1109/TC.2024.3350243.
20. Wilhelm M., Pionteck T. Static Task Mapping for Heterogeneous Systems Based on Series-Parallel Decompositions. In: 34th Heterogeneity in Computing Workshop (HCW 2025) at IPDPS 2025. Dallas, USA. 2025. P. 1–8. DOI: 10.48550/arXiv.2502.19745.
21. Manjunath V., Baunach M. Framework for Static Analysis and Verification of Low-Level RTOS Code. Journal of Systems Architecture. 2024. Vol. 154. Art. 103220. DOI: 10.1016/j.sysarc.2024.103220.
22. Haur I., Béchennec J.-L., Roux O. H. Formal Verification of Inter-Core Synchronization in a Multicore RTOS Kernel. In: Formal Methods and Software Engineering (ICFEM 2022). Lecture Notes in Computer Science. 2022. Vol. 13478. P. 134–150. DOI: 10.1007/978-3-031-17244-1_9.
23. Gu X., Liu P., Yang M. et al. Efficient Scheduler of RTOS for Multi/Many-Core Systems. Computers & Electrical Engineering. 2012. Vol. 38(3). P. 785–800. DOI: 10.1016/j.compeleceng.2011.09.009.

Kozelskyi O.V., Savenko B.O. EXTERNAL ADAPTIVE CLUSTERING SYSTEM TO INCREASE THE FLEXIBILITY OF REAL -TIME OPERATING SYSTEMS BASED ON DYNAMIC TASKS PLANNING

The study addresses the problem of ensuring stable and predictable operation of real-time operating systems (RTOS) in cyber-physical environments with a high degree of dynamism and changing operating conditions. Limitations of traditional task scheduling mechanisms have been identified, manifesting under peak loads, priority conflicts, shortages of computing resources, and system failures. To overcome these shortcomings, a new approach is proposed based on the use of an adaptive clusterizer, which groups tasks according to their importance, current resource load, and interdependencies between processes. A key element of the architecture is an adaptive scheduling system that accounts for dynamic priority changes, ensuring timely task redistribution depending on the current state of the system. The proposed solution helps avoid overloading individual modules, minimizes delays in executing critical processes, reduces reaction time to unforeseen events, and ensures uninterrupted RTOS operation even in the event of partial failures. The architecture provides flexible interaction between computing nodes, incorporates feedback mechanisms, and allows automatic adjustment of execution plans when input conditions change. A method for the operation of the clusterizer and a mathematical model describing the state changes of tasks in real time with probabilistic characteristics have been developed. A comparative analysis with classical approaches demonstrates improvements in overall system reliability, stability in executing high-priority tasks, and efficiency in resource utilization. The proposed concept can be applied in autonomous transportation, industrial automation, robotic systems, and mission-critical control systems requiring high fault tolerance and precision.

Key words: *operating systems, cyber-physical systems, task clustering, RTOS, load balancing, task scheduler, fault tolerance.*

Дата надходження статті: 14.08.2025
Дата прийняття статті: 01.09.2025
Опубліковано: 16.12.2025