УДК 004.94

ДОСЛІДЖЕННЯ ВПЛИВУ ІНТЕРВАЛУ ПЕРЕПЛАНУВАННЯ ЧЕРГИ ЗАДАЧ НА ЕФЕКТИВНІСТЬ ПАКЕТНИХ МЕТОДІВ У ТУМАННИХ СИСТЕМАХ

Сбітнєв О.Ю.[1]; Волощук Л.А., к.т.н., доц.[2]
[1,2] Одеський національний університет імені І. І. Мечникова
[1,2] Україна, Одеса
[1] alexsbitnev99@gmail.com; [2] lavstumbre@gmail.com

Анотація. *Під час порівняння пакетних і онлайн-методів розподілу задач у туманних системах за ефективністю (середньою затримкою) рідко враховується, що пакетний метод очікує моменту планування. На імітаційній моделі показано, що за помірного навантаження саме це очікування, а не якість оптимізації, робить затримку методів рою частинок, генетичного та мурашиного алгоритмів вищою, ніж у випадкового вибору. Критичний інтервал перепланування становить 13–18 мс; усунути програш можна лише його зменшенням.*

Ключові слова*: туманні обчислення; розподіл задач; пакетні методи; інтервал перепланування; імітаційне моделювання.*

Вступна частина. Розподіл підзадач між вузлами туманної системи [1] можна організувати двома способами. В онлайн-схемі вузол призначають у момент появи підзадачі. У пакетній оркестратор раз на Δ мс збирає все, що надійшло, і розв'язує одну оптимізаційну задачу для всього набору: це дає змогу врахувати взаємний вплив підзадач, але відкладає початок їхнього виконання. До пакетних належать метаевристики: оптимізація роєм частинок (Particle Swarm Optimization, PSO), генетичний алгоритм (Genetic Algorithm, GA), оптимізація мурашиною колонією (Ant Colony Optimization, ACO) та гібрид PSO з моделлю машинного навчання (Machine Learning + PSO, ML+PSO) [2]. У порівняннях пакетних методів інтервал, як правило, лишається налаштуванням за замовчуванням, і його внесок у результат окремо не оцінюють, тоді як відтворюваність висновків залежить від деталей реалізації алгоритмів [3]. Через це програш пакетного методу легко приписати слабкості алгоритму, хоча його причиною може бути сама схема планування.

Метою дослідження є з'ясувати, чому за помірного навантаження пакетні методи поступаються випадковому вибору, кількісно визначити критичний інтервал Δ*, за якого це припиняється, і оцінити, що більше впливає на затримку: інтервал чи якість оптимізації.

Основна частина. Час від появи підзадачі до завершення її виконання складається з паузи до ухвалення рішення, стояння в черзі вузла, пересилання даних і обчислення. Три останні визначаються вибраним вузлом і станом системи; перша ж виникає лише в пакетній схемі, бо підзадача мусить дочекатися найближчого запуску планувальника. За рівномірного надходження ця пауза в середньому становить половину інтервалу. Тому для пакетного методу розглянуто залежність

$$L(\Delta) \approx L_0 + \kappa\Delta, \quad \kappa \approx 0{,}5, \tag{1}$$

де L_0 — це затримка за гіпотетично миттєвого перепланування ($\Delta \to 0$), а κ — скільки мілісекунд додає кожна мілісекунда інтервалу. Випадковий вибір від Δ не залежить, тому точка, де пряма (1) досягає його рівня, є межею доцільності пакетного методу: за $\Delta < \Delta^*$ він кращий, за $\Delta > \Delta^*$ гірший (Δ^* – критичний інтервал перепланування).

Експерименти виконано на дискретно-подієвій моделі (бібліотека SimPy) [4]. Використано два набори запусків. У першому інтервал Δ змінювали від 5 до 160 мс (шість значень) у сценаріях Smart Factory, Smart City та Industrial IoT Edge за інтенсивності 30 задач/с; кожну точку усереднено за 8 прогонами по 6 с, а модель (1) підганяли до 48 спостережень на метод (табл. 1). У другому, еталонному, Δ дорівнював 20 мс у п'яти сценаріях із [5] (16 прогонів по 8 с). Методи отримували ідентичні послідовності випадкових чисел; відмінність від випадкового вибору перевіряли за критерієм Велча [6] з поправкою на множинні порівняння. Для зіставлення використано три методи без пакетування: федеративне навчання (Federated Learning, FL), багатоагентне навчання з підкріпленням (Multi-Agent Reinforcement Learning, MARL) та евристику Min-Expected-Delay-Net (MED-Net), яка обирає вузол із найменшим добутком довжини черги на вартість одного обслуговування з урахуванням мережевих хопів.

Таблиця 1. Оцінки лінійної моделі затримки пакетних методів (регресія за результатами першого набору запусків)

Сценарій	к (PSO / ACO / ML+PSO)	L_0, мс (PSO / ACO / ML+PSO)	L_{Random}, мс	Δ*, мс (PSO / ACO / ML+PSO)
Smart Factory	0,69 / 0,70 / 0,70	12,3 / 12,2 / 10,1	21,1	12,7 / 12,7 / 15,7
Smart City	0,69 / 0,69 / 0,70	13,7 / 13,4 / 11,2	23,2	13,6 / 14,2 / 16,9
Industrial IoT Edge	0,69 / 0,70 / 0,71	13,9 / 13,6 / 11,1	23,8	14,4 / 14,6 / 17,9

Три різні оптимізатори дали практично однаковий нахил (к = 0,69–0,71 за довірчих інтервалів ± 0,01–0,02), тож він зумовлений самим накопиченням пакета, а не алгоритмом розв'язання. Методи без пакетування від Δ не залежать: для FL і MARL к = 0 ± 0,02. Значення к ≈ 0,7 перевищує теоретичні 0,5: за допоміжними вимірами частка Δ, яку становить очікування рішення, зростає разом з інтервалом (від 0,46 до 0,63). Причину цього ефекту не встановлено. Жодне з дев'яти поєднань «метод — сценарій» не має Δ* вище за 17,9 мс (найменше 12,7 мс), тож стандартні 20 мс лежать за межею доцільності пакетного планування. Це підтверджує й пряме порівняння за Δ = 20 мс: у чотирьох сценаріях із п'яти PSO, GA та ACO програють випадковому вибору 11–21 %, а ML+PSO або не відрізняється від нього, або поступається на 4–10 %. Для порівняння, FL і MARL у більшості сценаріїв швидші за випадковий вибір на 29–42 %, а проста евристика MED-Net без навчання дає 11,9–12,8 мс, тобто нижче за будь-який пакетний метод.

Щоб відокремити ефект очікування від якості самої оптимізації, цільову функцію PSO і GA порівняно з бібліотечними реалізаціями pyswarms [7] і DEAP [8] на 30 випадкових пакетах кожного розміру (3, 10 і 30 підзадач): розбіжність становила від −3 до +6 %, тож різниця в затримці не пов'язана з помилками коду. Для пакетів із 30 підзадач жадібний розподіл дає значення цілі на 35–43 % краще за метаевристики, а випадковий — у 2,6–3 раза гірше, тобто пакетна оптимізація сама по собі працює краще за випадковий вибір.

Розкладемо затримку за Δ = 20 мс на дві частини (рис. 1, б). Плата за очікування кΔ становить 13,8–14,2 мс, тобто 50–58 % розрахункової затримки, тоді як виграш від оптимізації порівняно з випадковим вибором, $L_{Random} - L_0$, дорівнює лише 8,8–12,7 мс (табл. 2). Плата перевищує виграш в усіх дев'яти комбінаціях на 1,5–5,1 мс; критичний інтервал Δ* дорівнює відношенню виграшу до нахилу. Розрахункова затримка за регресією відрізняється від виміряної в базовому режимі приблизно на 2 мс (наприклад, 24,0 мс для PSO у Smart Factory проти 26,1), тож Δ* слід розглядати як оцінку, а не точне значення.

Таблиця 2. Виграш від оптимізації та плата за очікування за Δ = 20 мс, мс: $L_{\text{Random}} - L_0$ / кΔ (розрахунок за табл. 1)

Сценарій	PSO	ACO	ML+PSO
Smart Factory	8,8 / 13,8	8,9 / 14,0	11,0 / 14,0
Smart City	9,5 / 13,8	9,8 / 13,8	12,0 / 14,0
Industrial IoT Edge	9,9 / 13,8	10,2 / 14,0	12,7 / 14,2

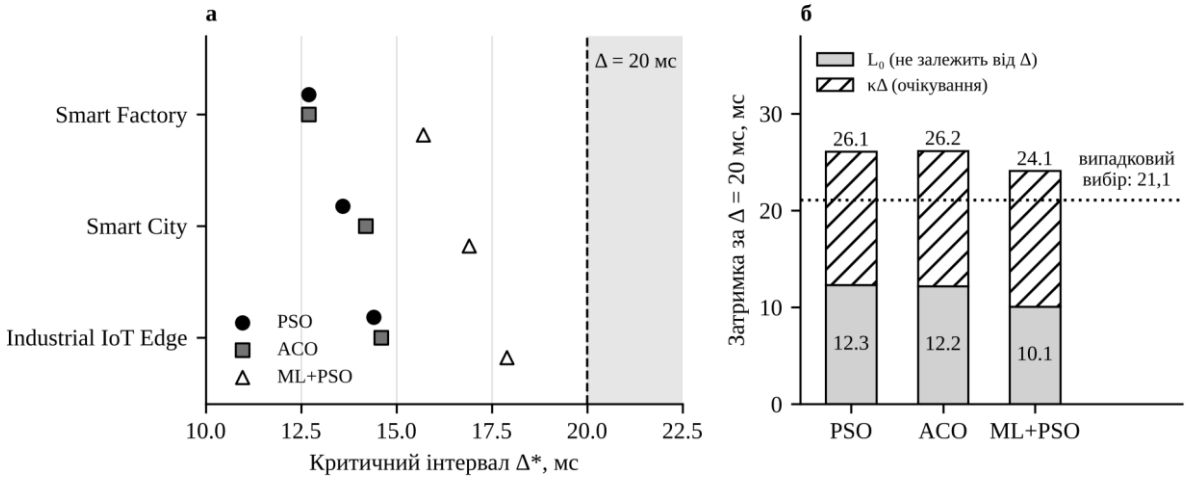


Рис. 1. Критичні інтервали Δ* (а) та складники розрахункової затримки за Δ = 20 мс у Smart Factory (б); за коефіцієнтами табл. 1

Звідси випливають два наслідки. По-перше, якість оптимізатора є слабким важелем: зменшення L_0 на 1 мс зсуває Δ* лише на 1/к ≈ 1,4 мс (перехід від PSO до ML+PSO у Smart Factory знизив L_0 на 2,2 мс і підняв Δ* на 3,0 мс). Щоб PSO досягнув Δ* = 20 мс, потрібно було б мати L_0 ≈ 7,3 мс, тобто нижче за 12,4 мс жадібного правила MED-Net, що нереалістично. Крім того, L_0 для PSO і ACO (12,2–13,9 мс) лежить у межах 1,2 мс від затримки MED-Net у базовому режимі (12,4–12,8 мс): за миттєвого планування вони лише зрівнялися б із простою евристикою, а нижче значення ML+PSO (на 1,6–2,3 мс) є екстраполяцією за межі виміряного діапазону Δ ≥ 5 мс. По-друге, резерв лежить у зменшенні Δ: щоб потрапити під Δ*, інтервал слід скоротити від 20 мс на 10–37 %, а навіть за 5 мс пакетні методи не випереджають онлайн-методи (у Smart Factory розрахункова затримка 13,6–15,8 мс проти 12,8–13,3 мс у FL і MARL). Практичним напрямом є подієвий запуск планування (за накопиченням пакета певного розміру), який не фіксує Δ, але його ефективність потребує окремої перевірки.

Висновки. Кожна мілісекунда інтервалу додає до затримки пакетного методу близько 0,7 мс незалежно від оптимізатора, а стандартні 20 мс перевищують критичний інтервал (13–18 мс). Плата за очікування (≈ 14 мс) вища за виграш від оптимізації над випадковим вибором (9–13 мс), тому для зменшення програшу ефективніше скорочувати інтервал, ніж удосконалювати оптимізатор. Оцінка Δ* не враховує реального часу розв'язання пакетної задачі й отримана за одного потоку (30 задач/с) на синтетичному трафіку; її перевірка на реальних трасах і за інших інтенсивностей є наступним кроком.

Використання інструментів ШІ. Під час підготовки тез використано Claude (Anthropic) для структурування й редагування тексту та верстки. Результати експериментів, їх інтерпретацію й висновки перевірено авторами, які несуть відповідальність за зміст.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012). Fog computing and its role in the Internet of Things. *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, 13–16. https://doi.org/10.1145/2342509.2342513
[2] Salaht, F. A., Desprez, F., & Lebre, A. (2020). An overview of service placement problem in fog and edge computing. *ACM Computing Surveys, 53*(3), Article 65. https://doi.org/10.1145/3391196
[3] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2018). Deep reinforcement learning that matters. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1), 3207–3214. https://doi.org/10.1609/aaai.v32i1.11694
[4] SimPy. (n.d.). *SimPy: Discrete event simulation for Python*. https://simpy.readthedocs.io
[5] Sbitniev, O. Yu., & Voloshchuk, L. A. (2026). Adaptive method for service placement in fog computing systems based on graph attention and multi-agent reinforcement learning. *Informatics. Culture. Technique*, *3*(1), 346–366. https://doi.org/10.15276/ict.03.2026.29
[6] Welch, B. L. (1947). The generalization of "Student's" problem when several different population variances are involved. *Biometrika*, *34*(1–2), 28–35. https://doi.org/10.2307/2332510
[7] Miranda, L. J. V. (2018). PySwarms, a research-toolkit for particle swarm optimization in Python. *Journal of Open Source Software*, *3*(21), Article 433. https://doi.org/10.21105/joss.00433
[8] Fortin, F.-A., De Rainville, F.-M., Gardner, M.-A., Parizeau, M., & Gagné, C. (2012). DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, *13*, 2171–2175.

Sbitnev O.Yu., Voloshchuk L.A.

Study of the effect of the task queue rescheduling interval on the efficiency of batch methods in fog systems

Abstract. *Comparisons of batch and online task allocation methods in fog systems in terms of efficiency (mean latency) rarely account for the batch method waiting for the scheduling moment. A simulation model shows that under moderate load this waiting, rather than the quality of optimization, makes the latency of particle swarm, genetic and ant colony algorithms higher than that of random selection. The critical rescheduling interval is 13–18 ms; the disadvantage can be removed only by reducing the interval.*
Keywords*: fog computing; task allocation; batch methods; rescheduling interval; simulation modeling.*