УДК 004.75:004.4:519.8 DOI: 10.31673/2412-9070.2026.047807

Д. А. ІГНАТОВ, аспірант;
ORCID: 0009-0008-2864-8526
О. М. ШУШУРА, д-р техн. наук, професор,
ORCID: 0000-0003-3200-720X
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»

ФОРМАЛІЗАЦІЯ ЗАДАЧІ ОПТИМАЛЬНОГО АВТОМАСШТАБУВАННЯ KUBERNETES З УРАХУВАННЯМ ЗАТРИМКИ ЗАПУСКУ ПОДІВ

Проактивні автомасштабувачі Kubernetes прогнозують майбутнє навантаження і заздалегідь запускають нові поди, проте в усіх існуючих системах зроблено неявне припущення, що нова потужність з'являється миттєво або через однакову для всіх сервісів фіксовану затримку. Однак у поліглотних розгортаннях затримки запуску можуть відрізнятися в рази, що унеможливлює застосування єдиного глобального часового зсуву і породжує фундаментальну неузгодженість між рішенням оптимізатора та реальною доступністю потужності.

У роботі автомасштабування Kubernetes формалізовано як задачу дискретної оптимізації з обмеженням на «мертвий час», де затримка запуску входить до моделі ефективної потужності як окремий параметр кожного мікросервісу. Для задачі одного сервісу доведено дискретну опуклість цільової функції за рішенням про масштабування. Багатосервісний варіант з обмеженням на ресурси кластера класифіковано як NP-повну обмежену задачу про ранець, для розв'язання якої запропоновано використовувати жадібний підхід за маржинальною ефективністю, динамічне програмування по бюджету або LP-релаксацію з округленням. Зв'язок з моделлю M/M/k/setup з теорії черг аналітично обґрунтовує обмеженість реактивного масштабування, бо поліноміальне спадання штрафу з кількістю серверів унеможливлює компенсацію затримки запуску після настання пікового попиту.

Окремо досліджено структурну асиметрію реакції оптимізатора на помилки прогнозу, зумовлену обмеженням мертвого часу. Показано, що переоцінку попиту оптимізатор здатний нейтралізувати на етапі прийняття рішення, тоді як наслідки недооцінки є невідворотними: виправлення набуде сили лише після закінчення затримки запуску, протягом якої система перебуватиме у стані дефіциту потужності. Ця асиметрія є структурною властивістю моделі і не потребує модифікації цільової функції.

Отримані теоретичні результати закладають формальну основу для розробки практичних проактивних автомасштабувачів, адаптованих до гетерогенного середовища поліглотних мікросервісних систем.

Ключові слова: автомасштабування, Kubernetes, мікросервіси, затримка запуску, дискретна оптимізація, прогнозування навантаження, динамічне програмування, асиметричні втрати.

Вступ

Kubernetes є де-факто стандартом оркестрації для систем із мікросервісною архітектурою, він забезпечує також декларативне управління життєвим циклом подів, масштабування за метриками та відмовостійкість. Динамічне горизонтальне масштабування, яке відповідає за автоматичне регулювання кількості реплік сервісів відповідно до поточного попиту, є важливим компонентом, що впливає одночасно як на якість обслуговування користувачів, так і на операційні витрати кластера.

Підходи, які використовуються для ефективного автомасштабування, можна поділити на два класи: реактивні методи та проактивні. Перші реагують на вже виявлене перевищення порогових метрик, у той же час другі прогнозують майбутнє навантаження і масштабуються заздалегідь аби упередити порушення угод про рівень обслуговування (SLA). Обидва класи розвиваються паралельно: реактивні, представлені вбудованим у Kubernetes Horizontal Pod Autoscaler та його розширеннями, проактивні - рядом систем на основі статистичних і нейромережних моделей прогнозування навантаження.

Щоправда, незалежно від класу, усі існуючі підходи мають спільне обмеження, оскільки не враховують гетерогенну затримку запуску подів. У поліглотних мікросервісних розгортаннях час між тим, як було прийнято рішення про масштабування, і тим, як нова потужність стала доступною, може суттєво відрізнятись між сервісами. Дана прогалина набуває особливого значення з огляду на широке поширення поліглотних архітектур, у яких сервіси на компільованих мовах співіснують із сервісами на мовах з керованими виконавчими середовищами, що мають значно довший час прогріву. Таким чином, виникає потреба у формальній моделі автомасштабування, яка явно враховує індивідуальну затримку запуску кожного мікросервісу та аналітично обґрунтовує властивості оптимізаційної задачі у такому гетерогенному середовищі.

Постановка проблеми

Horizontal Pod Autoscaler (HPA) в Kubernetes є реактивним за своєю природою, оскільки він додає нові репліки лише після того, як стається певне явище. Під цим явищем мається на увазі перевищення певною метрикою заданого порогу. Відомо, що такий підхід призводить до порушення SLA при стрибках навантаження, оскільки новим подам потрібен час для запуску [1], [2]. У свою чергу, проактивні автомасштабувачі здатні обходити такі проблеми завдяки тому, що прогнозують навантаження, тож масштабуються заздалегідь [3] – [8].

Однак у всіх проактивних автомасштабувачів є одне неявне припущення: після того як прийнято рішення про масштабування, нова потужність з'являється миттєво або після однакової фіксованої затримки. На практиці час запуску пода суттєво відрізняється між сервісами. Go-мікросервіс стартує за 5–7 секунд, Java Spring Boot за 25–30 секунд [9], [10]. У поліглотних розгортаннях на кшталт Google Online Boutique (11 сервісів, 5 мов) виміряні нами затримки запуску розрізняються у 4.2 рази.

Така гетерогенність створює фундаментальну неузгодженість, оскільки проактивний автомасштабувач, який зсуває свій прогноз на одну фіксовану константу, як це робить AHPA [3], не здатний однаково працювати одночасно і з найшвидшим, і з найповільнішим сервісом. Якщо зсув прогнозу замалий, то повільні сервіси не встигнуть перейти до готовності на старті, а якщо зсув завеликий, то сервіси зі швидким стартом даремно використовуватимуть ресурси.

Окрім цього, взаємозв'язок між затримкою запуску і якістю прогнозу не є простим. Коли прогноз недооцінює майбутній попит, подальше виправлення може набути сили лише після того, як спливе затримка запуску, тож весь цей час система вимушена мати недостатню потужність. З іншої сторони, переоцінку оптимізатор гасить ще на етапі прийняття рішення, оскільки здатний просто масштабуватися менше. Тобто обмеження на «мертвий час» автоматично створює асиметричний штраф за помилки прогнозу – і його величина зростає разом із затримкою запуску. Таким чином, актуальною є розробка моделі та методів проактивного автомасштабування Kubernetes, які враховують індивідуальну затримку запуску кожного мікросервісу. Ігнорування цієї гетерогенності призводить до або недостатньої реакції повільних сервісів, або марного використання ресурсів швидкими сервісами, а також створює асиметричну чутливість до помилок прогнозування навантаження.

Аналіз останніх досліджень та публікацій

Проактивні автомасштабувачі прогнозують попит і масштабуються заздалегідь, прикладами сучасних систем є AHPA [3] (RobustSTL-декомпозиція, продакшн Alibaba), DeepScaler [4] (просторово-часові GNN), TempoScale [5] (Informer), AAPA [6] (архетип-орієнтований з невизначеністю), E3Former [11] (онлайн-ансамбль, продакшн ByteDance на 600 тис. CPU-ядер)

та інші [7], [8], [12] – [14]. Усі прогнозують навантаження $\hat{w}(t+k)$ і рахують кількість реплік як $\lceil \hat{w}/\mu \rceil$. Інакше кажучи, трактують потужність як миттєво доступну. AHPA – єдина система, яка компенсує затримку запуску, використовуючи один глобальний часовий зсув. Наскільки відомо авторам, жодна система не оцінює затримку окремо для кожного сервісу і не використовує її в оптимізаторі.

Запуск пода в Kubernetes складається з планування, підтягування образу, init-контейнерів, старту додатка і необхідного часу для readiness-проб. Виміряні значення мають діапазон від <1 с (закешований образ Go-сервіса) до >60 с (довгий image pull + прогрів JVM у випадку Spring Boot сервіса) [9], [10], [15]. Kubernetes декларує SLO $p99 \leq 5$ с, але він явно не враховує при цьому image pull та init-контейнери [17]. Strasser і Kounev [16] запропонували прогнозувати час запуску контейнера за метаданими маніфесту. Khan [10] виміряв варіацію 2.7x в залежності лише від інфраструктурного шару. Однак жодна з цих робіт не інтегрує прогнози в рішення про масштабування.

Процеси з транспортною затримкою вивчають з часів предиктора Сміта [18]. Dellkrantz та ін. [19] застосували компенсацію «мертвого часу» до автомасштабування VM. MPC був використаний Nguyen та ін. [20] для планування холодних стартів у безсерверних обчисленнях, завдяки чому вдалося скоротити tail-latency на 85%. Проте, жодна з цих робіт не опрацьовує гетерогенні індивідуальні часові затримки.

Модель M/M/k/setup [21], [22] описує багатосерверні черги, коли вмикання сервера потребує часу налаштування. Williams та ін. [22] довели той факт, що при детермінованому часі налаштування $1/\alpha$ середня затримка у M/M/k/setup становить $\Omega\left(1/(\alpha\sqrt{k\rho})\right)$. Отже, спадає лише поліноміально з кількістю серверів k, на відміну від експоненційного спадання у M/M/k без налаштування. Даний результат формально обгрунтовує, чому реактивне масштабування не здатне компенсувати затримку запуску: просто збільшити кількість подів недостатньо, оскільки штраф за час налаштування нікуди не зникає.

V. Sachidananda, A. Sivaraman у своєму дослідженні автомасштабувача Erlang [23] спираються на теорію черг задля того аби оптимізувати поділ VM між мікросервісами. Спорідненими напрацюваннями є робота Cai і Buyya [24], у якій для визначення кількості контейнерів за цільовою затримкою була використана обернена модель черги. При цьому, моделювання затримки запуску не проводилось.

Стандартні метрики прогнозу, такі як MSE чи MAPE, мають однакове ставлення до переоцінки і недооцінки, що не завжди є коректним, оскільки у задачах виділення ресурсів ці помилки мають різну вартість: переоцінка веде до over-provisioning (марно витрачені ресурси), недооцінка – до under-provisioning (порушення SLA). Eramo та ін. [25] у задачі розподілу пропускної здатності в NFV запропонували для LSTM асиметричну функцію втрат, яка була параметризована одним коефіцієнтом, що задавав співвідношення штрафів за переоцінку/недооцінку. Аналіз порівняння із використанням симетричної RMSE показав, що експлуатаційну вартість вдалося скоротити на 40%. Окрім цього, Dartois та ін. [26] використали квантильну регресію для повернення невикористаних хмарних ресурсів. Був застосований параметр квантиля τ, який напряму керує балансом між SLA та вартістю. Zhang та ін. [27] запропонували вартісно-орієнтований прогноз навантаження, у якому функція втрат виводиться з самої оптимізаційної задачі. Концепція «прогнозуй-і-оптимізуй» була досліджена Abolghasemi та ін. [28], яка підтвердила, що точність прогнозу і вартість рішення корельовані позитивно, але не ідеально. Ці всі роботи встановлюють, що асиметричні штрафи за помилки прогнозу важливі для виділення ресурсів. Однак, жодне з досліджень не розглядає, як пов'язані між собою зміщення прогнозу та затримка запуску, хоча обмеження «мертвого часу» призводить до збільшення штрафу пропорційно величині затримки.

Мета і задачі дослідження

Метою роботи є формалізація задачі оптимального автомасштабування Kubernetes з урахуванням гетерогенної затримки запуску мікросервісів як задачі дискретної оптимізації, аналітичне дослідження її властивостей та обґрунтування методів розв'язання.

Для досягнення поставленої мети необхідно розв'язати наступні задачі:

– побудувати модель ефективної потужності мікросервісу, у якій затримка запуску є окремим параметром кожного сервісу, та сформулювати на її основі цільову функцію оптимізаційної задачі автомасштабування;

– розширити одно-сервісну формалізацію на багатосервісний випадок з обмеженням на сумарні ресурси кластера;

– дослідити структурні властивості отриманої задачі: дискретну опуклість для одного сервісу та обчислювальну складність багатосервісного варіанта;

– через зіставлення з моделлю M/M/k/setup з теорії черг аналітично обґрунтувати принципову обмеженість реактивного масштабування у задачах з ненульовою затримкою запуску;

– проаналізувати структурну асиметрію реакції оптимізатора на помилки прогнозу навантаження, яка зумовлена обмеженням на «мертвий час»;

– обґрунтувати методи розв'язання багатосервісної задачі: жадібний за маржинальною ефективністю, динамічне програмування по бюджету, LP-релаксація з округленням.

Результати дослідження

Введемо множину M мікросервісів $\{s_1, \dots, s_M\}$ на кластері Kubernetes. Час в межах задачі масштабування будемо розглядати як дискретний з кроком керування Δt. Для спрощення формул час далі представляється цілочисельним номером кроку. На кроці t сервіс i має $n_i(t) \geq 1$ готових реплік, навантаження у вигляді кількості запитів на секунду $w_i(t) \geq 0$ (RPS), пропускну здатність пода $\mu_i > 0$ (RPS/под), необхідний обсяг ресурсів для пода $r_i > 0$ та затримку запуску $d_i(t) > 0$ секунд, яку можна виразити у кроках $\delta_i(t) = \lceil d_i(t)/\Delta t \rceil$.

На кроці t для кожного сервісу i контролер обирає кількість подів, які треба запустити: $\Delta_i \in \{0, 1, \dots, \Delta_{max}\}$.

Масштабування вниз обробляється окремими cooldown-евристиками, оскільки воно часово симетричне: потужність падає одразу, а ресурси ще продовжують використовуватись, створюючи дзеркальну задачу до запуску.

Нехай прогнозатор навантаження видає $\hat{w}_i(t+k)$ запитів для $k = 1, \dots, H$, а оцінювач затримки – $\hat{\delta}_i$ кроків. Тоді ефективна потужність на кроці $t+k$ за умови запуску Δ_i подів обчислюється так:

$$C_i(t+k \mid \Delta_i) = [n_i(t) + \Delta_i \cdot \mathbb{1}[k \geq \hat{\delta}_i]] \cdot \mu_i . \quad (1)$$

Нові поди дають нуль потужності перших $\hat{\delta}_i$ кроків, поки вони стартують, і повну потужність згодом. Це і є визначальне обмеження задачі.

Цільова функція представлена наступним чином:

$$J_i(\Delta_i) = \sum_{k=1}^{H} [\lambda_V \cdot \underbrace{\max(0, \hat{w}_i(t+k) - C_i(t+k \mid \Delta_i))}_{V_i(t+k)\text{: порушення SLA}} + \lambda_R \cdot \underbrace{(n_i + \Delta_i) \cdot \mu_i}_{R_i(t+k)\text{: вартість ресурсів}}], \quad (2)$$

де $\lambda_V > 0$ штрафує перевищення попиту над потужністю, $\lambda_R > 0$ штрафує використання ресурсів, обидва є безрозмірними ваговими коефіцієнтами, а на практиці значення має їхнє співвідношення. Перший доданок $V_i(t+k) = max(0, \hat{w}_i(t+k) - C_i(t+k|\Delta_i))$ є мірою порушення SLA. Другий доданок $R_i(t+k) = (n_i + \Delta_i) \cdot \mu_i$ – мірою сумарної наданої потужності та виступає у ролі проксі для ресурсної вартості за припущення, що витрати на ресурси пропорційні потужності.

Задача оптимізації для одного сервісу:

$$\Delta_i^* = \arg \min_{\Delta_i \in \{0,1,\dots,\Delta_{max}\}} J_i(\Delta_i) \quad (3)$$

при обмеженні $n_i + \Delta_i \leq n_{max}$.

У випадку, коли сервісів багато і вони конкурують за ресурси кластера, задача оптимізації має вид:

$$\min_{\Delta} \sum_{i=1}^{M} J_i(\Delta_i) \quad (4)$$

за умов:

$$\sum_{i=1}^{M} (n_i(t) + \Delta_i) r_i \le R_{\text{cluster}},$$

$$n_i(t) + \Delta_i \le n_{max,i},$$

$$\Delta_i \in \{0, \dots, \Delta_{max}\}\ \forall i \in \{1, \dots, M\},$$

де r_i – запит на ресурси одного пода, $R_{cluster}$ – сумарна ємність кластера. Бюджетне обмеження моделює жорсткий ліміт фізичних ресурсів кластера.

Розглянемо варіант задачі для одного сервісу. Доданок вартості $J^R(\Delta) = H \cdot \lambda_R \cdot \mu \cdot (n + \Delta)$ є лінійним, а отже опуклим. На кожному кроці горизонту $k \ge \hat{\delta}_i$ доданок порушення $V_k(\Delta) = max(0, \hat{w}_k - \mu \cdot (n + \Delta))$ є максимумом двох опуклих функцій (нуля та лінійної з від'ємним кутом – μ), а отже і сам є опуклим. Оскільки сума опуклих функцій – опукла, то, таким чином, задача для одного сервісу $J_i(\Delta_i)$ є дискретно-опуклою, тому другі різниці $J(\Delta + 1) - 2J(\Delta) + J(\Delta - 1) \ge 0$ для всіх Δ. З цього слідує те, що J_i унімодальна над цілими, тому для вирішення задачі можна робити перебір з раннім виходом або бінарний пошук за $O(log\Delta_{max} \cdot H)$. Дане спостереження було також перевірено емпіричним шляхом.

Задача масштабування одного сервісу являє собою керувальний аналог моделі M/M/k/setup з теорії черг [21], [22]. Кожен под відповідає серверу, а затримка запуску – детермінованому часу налаштування. Williams та ін. [21] доводять, що середній час очікування в M/M/k з детермінованим налаштуванням $1/\alpha$ задовольняє $E[T_Q] = \Omega\left(\frac{1/\alpha}{\sqrt{k\rho}}\right)$, де k – кількість серверів (подів), $\rho = \frac{\lambda}{k\mu}$ – коефіцієнт завантаження системи.

Режим QED [29], у свою чергу, аргументує, що запас безпеки масштабується як $\sqrt{R}$, де $R = \lambda/\mu$ – запропоноване навантаження. Тож таке поліноміальне, а не експоненційне, спадання за k має два наслідки. По-перше, слід зазначити, що реактивне масштабування є принципово обмеженим, оскільки додавання подів після того, як попит уже надійшов, не усуває штрафу $1/\alpha$. По-друге, проактивне масштабування має явно враховувати δ_i. Окрім цього, зв'язок із варіантом M/M/k/setup/delayedoff [21] формалізує гістерезис scale-down: час налаштування впливає на оптимальний час очікування перед відключенням сервера, що є аналогом зваженого за затримкою запуску cooldown.

Сформульована задача охоплює кілька класичних формалізацій: модель прогнозувального керування (MPC) з цілочисельними змінними та гетерогенним «мертвим часом»; обмежену задачу про ранець (для спільної задачі з обмеженням на ресурси); задачу газетяра в одноперіодному випадку як баланс між переплатою (вартістю) і недоплатою (порушенням); задачу планування машин з переналаштуванням, де поди відповідають машинам, а запуск – налаштуванню.

Загальна задача (4) у випадку багатьох сервісів має сепарабельну опуклу ціль з одним лінійним обмеженням, а структурно являє собою обмежену NP-повну задачу про ранець. Ефективних підходів для її вирішення є декілька. Першим є жадібний за маржинальною ефективністю: сортуємо всі пари (сервіс, Δ) за $\Delta J/r_i$, додаємо по порядку, складність $O(M\Delta_{max}log(M\Delta_{max}))$. Другим варіантом є динамічне програмування по бюджету ресурсів: стан $(i, R_{\text{залишок}})$, складність псевдополіноміальна $O(M\Delta_{max}R)$. Ще одним варіантом є LP-релаксація з округленням: релаксована задача є опуклою, розв'язується за $O(MlogM)$, розрив цілочисельності обмежений. На практиці більшість кластерів Kubernetes використовують окремий cluster autoscaler для отримання нод, завдяки чому обмеження ресурсів розв'язується незалежно. У такому випадку задача для кожного сервісу незалежна і поліноміальна.

Окрім цього, модель потужності сама по собі вносить структурну асиметрію в реакцію оптимізатора на помилки прогнозу. Розглянемо прогноз із систематичним зміщенням b, тобто $\hat{w}_i(t + k) = w_i(t + k) \cdot (1 + b)$, де $b > 0$ – переоцінка, а $b < 0$ – недооцінка.

При переоцінці ($b > 0$) оптимізатор при прийнятті рішення про масштабування бачить роздутий попит. Якщо попит роздутий, але не настільки, щоб виправдовувати додавання по-

дів, то $\Delta^* = 0$, отже зміщення не дає жодного ефекту. Якщо ж виправдовує, то додаткові поди надійдуть через δ_i кроків і дадуть надлишкову потужність, яку поглине доданок вартості λ_R, але порушень SLA не виникне. Тобто завдяки тому, що оптимізатор може вирішувати не масштабуватись, то він здатний «згладжувати» переоцінку на етапі прийняття рішення.

При недооцінці ($b < 0$) оптимізатор бачить занижений попит і може вирішити не збільшувати кількість реплік ($\Delta^* = 0$) у випадку, коли насправді це слід було б зробити. Коли справжній (вищий) попит прийде, то система не зможе миттєво отримати необхідну потужність і щонайменше ще δ_i кроків буде залишатись у дефіциті. Таким чином, будь-яке виправлення неодмінно буде запізнюватись на час запуску. При прийнятті рішення оптимізатор не має змоги скасувати недооцінку. Це призводить до того, що штраф, який триватиме весь час затримки, є неминучим. Дана асиметрія є вбудованою в обмеження на «мертвий час» і не потребує модифікації цільової функції, а кількісну оцінку цього ефекту необхідно дослідити експериментально.

Висновки та перспективи подальших досліджень

Таким чином, було формалізовано задачу оптимального управління масштабуванням Kubernetes з явним урахуванням гетерогенної затримки запуску подів у вигляді задачі дискретної оптимізації. Запропоновано модель ефективної потужності, в якій затримка запуску входить як окремий per-service параметр, а рішення про масштабування приймається з урахуванням обмеження «мертвого часу». Показано, що для одного сервісу дана задача є дискретно-опуклою, а у випадку багатьох сервісів, які конкурують за ресурси кластера, задача є NP-повною, що належить до класу обмежених задач про ранець. Для вирішення задачі оптимізації запропоновано використати жадібний підхід, динамічне програмування по бюджету або LP-релаксацію.

Через зіставлення з моделлю M/M/k/setup з теорії черг аналітично обґрунтовано принципову обмеженість реактивного масштабування, оскільки поліноміальне, а не експоненційне спадання штрафу з кількістю серверів унеможливлює компенсацію затримки запуску після настання пікового попиту.

Встановлено структурну асиметрію реакції оптимізатора на помилки прогнозу: переоцінку попиту він здатний нейтралізувати на етапі прийняття рішення (не масштабуватися), тоді як наслідки недооцінки є невідворотними і тривають щонайменше протягом усього часу затримки запуску. Зазначена асиметрія є структурною властивістю моделі і не потребує модифікації цільової функції.

Отримані теоретичні результати закладають формальну основу для розробки практичних проактивних автомасштабувачів, адаптованих до гетерогенного середовища поліглотних мікросервісних систем.

Перспективи подальших досліджень охоплюють низку напрямків, що випливають з отриманих теоретичних результатів. По-перше, доцільною є експериментальна оцінка кількісного впливу структурної асиметрії на показники SLA та ресурсні витрати у реальних поліглотних мікросервісних розгортаннях. Це дозволить емпірично підтвердити закономірності, виведені аналітично. По-друге, перспективною є побудова окремого для кожного сервісу оцінювача затримки запуску. По-третє, слушною ідеєю є інтеграція запропонованої моделі з адаптивним налаштуванням вагових коефіцієнтів λ_V та λ_R залежно від поточного SLA-бюджету сервісу та цілей оптимізації.

Внесок авторів

Олексій ШУШУРА – аналітичне опрацювання джерел, загальна постановка проблематики роботи, методика, редагування та доопрацювання; Дмитро ІГНАТОВ – підготовка та систематизація джерел, аналітичне опрацювання джерел, формалізація задачі оптимального управління масштабуванням подів Kubernetes, теоретичні дослідження, методика, підготовка первинного тексту.

Декларація про штучний інтелект

Автори декларують, що штучний інтелект не використовувався для генерування наукового змісту, результатів дослідження, інтерпретації даних або формулювання висновків. Усі наукові положення статті є результатом самостійної роботи авторів.

Конфлікт інтересів

Автори заявляють про відсутність конфлікту інтересів та підтверджують, що під час підготовки цієї роботи не існувало жодних комерційних, фінансових чи інших взаємовідносин, які могли б бути розцінені як такі, що здатні вплинути на результати дослідження або їх інтерпретацію. Робота виконана відповідно до принципів академічної доброчесності, етичних норм проведення наукових досліджень та вимог редакційної політики щодо запобігання конфлікту інтересів.

Список використаної літератури

*1. Kubernetes Authors. (2024). *Horizontal Pod Autoscaler*. Kubernetes Documentation. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/*

*2. Kedify. (2026). *Autoscaling delay: Resource-based vs. proactive metrics*. Kedify Blog. https://www.kedify.io/resources/blog/autoscaling-delay-resource-based-hpa-vs-proactive-metrics*

*3. Zhou, Z., Zhang, C., Ma, L., Gu, J., Qian, H., Wen, Q., Sun, L., Li, P., & Tang, Z. (2023). AHPA: Adaptive Horizontal Pod Autoscaling Systems on Alibaba Cloud Container Service for Kubernetes. In *Proceedings of the AAAI Conference on Artificial Intelligence* (pp. 15621–15629). https://doi.org/10.1609/aaai.v37i13.26852*

*4. Meng, C., Song, S., Tong, H., Pan, M., & Yu, Y. (2023). DeepScaler: Holistic Autoscaling for Microservices Based on Spatiotemporal GNN with Adaptive Graph Learning. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 53–65). https://doi.org/10.1109/ASE56229.2023.00038*

*5. Wen, L., Xu, M., Toosi, A. N., & Ye, K. (2024). TempoScale: A Cloud Workloads Prediction Approach Integrating Short-Term and Long-Term Information. In *Proceedings of the 2024 IEEE 17th International Conference on Cloud Computing (CLOUD)* (pp. 183–193). https://doi.org/10.1109/CLOUD62652.2024.00030*

6. Zhang, G., Vippagunta, S., Nandagopal, R., Raman, S., Xu, J., Pfeiffer, M., Chatterjee, S., Tan, Z., Guo, W., & Jiang, H. (2025). AAPA: An Archetype-Aware Predictive Autoscaler with Uncertainty Quantification for Serverless Workloads on Kubernetes (arXiv:2507.05653). arXiv. https://doi.org/10.48550/arXiv.2507.05653

7. Shaikh, F., Reali, G., & Femminella, M. (2026). Mitigating Temporal Blindness in Kubernetes Autoscaling: An Attention-Double-LSTM Framework (arXiv:2603.28790). arXiv. https://doi.org/10.48550/arXiv.2603.28790

8. Dashtbani, M., & Tahvildari, L. (2025). STaleX: A Spatiotemporal-Aware Adaptive Autoscaling Framework for Microservices (arXiv:2501.18734). arXiv. https://doi.org/10.48550/arXiv.2501.18734

9. Medel, V., Tolosana-Calasanz, R., Bañares, J. Á., Arronategui, U., & Rana, O. F. (2024). Characterising resource management performance in Kubernetes (arXiv:2401.17125). arXiv. https://doi.org/10.48550/arXiv.2401.17125

10. *Khan, S. (2026). Decomposing Docker Container Startup Performance: A Three-Tier Measurement Study on Heterogeneous Infrastructure (arXiv:2602.15214). arXiv. https://doi.org/10.48550/arXiv.2602.15214*

11. Chen, J., He, X., Ye, H., Jiang, F., Zhang, T., Chen, J., & Gao, X. (2025). Online Ensemble Transformer for Accurate Cloud Workload Forecasting in Predictive Auto-Scaling (arXiv:2508.12773). arXiv. https://doi.org/10.48550/arXiv.2508.12773

12. Nguyen, H. X., Zhu, S., & Liu, M. (2022). Graph-PHPA: Graph-based Proactive Horizontal Pod Autoscaling for Microservices using LSTM-GNN (arXiv:2209.02551). arXiv. https://doi.org/10.48550/arXiv.2209.02551

*13. Xue, S., Qu, C., Shi, X., Liao, C., Zhu, S., Tan, X., Ma, L., Wang, S., Wang, S., Hu, Y., Lei, L., Zheng, Y., Li, J., & Zhang, J. (2022). A Meta Reinforcement Learning Approach for Predictive Autoscaling in the Cloud. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (pp. 4290–4299). https://doi.org/10.1145/3534678.3539063*

*14. Qian, H., Wen, Q., Sun, L., Gu, J., Niu, Q., & Tang, Z. (2022). RobustScaler: QoS-Aware Autoscaling for Complex Workloads. In *Proceedings of the 2022 IEEE 38th International Conference on Data Engineering (ICDE)* (pp. 2762–2775). https://doi.org/10.1109/ICDE53745.2022.00252*

*15. Harter, T., Salmon, B., Liu, R., Arpaci-Dusseau, A. C., & Arpaci-Dusseau, R. H. (2016). Slacker: Fast distribution with lazy Docker containers. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST '16)* (pp. 181–195). USENIX Association.*

*16. Straesser, M., & Kounev, S. (2021). Container start times: Empirical analysis and predictability. In *Symposium on Software Performance (SSP 2021)*. CEUR-WS. https://ceur-ws.org/Vol-3043/poster4.pdf*

*17. Kubernetes Community. (2024). *Pod Startup Latency SLI/SLO*. GitHub. https://github.com/kubernetes/community/blob/master/sig-scalability/slos/pod_startup_latency.md*

*18. Smith, O. J. M. (1959). A controller to overcome dead time. *ISA Journal*, *6*(2), 28–33.*

*19. Dellkrantz, M., Dürango, J., Robertsson, A., & Kihl, M. (2015). Model-based deadtime compensation of virtual machine startup times. In *Feedback Computing 2015*.*

20. Nguyen, C., Bhuyan, M., & Elmroth, E. (2025). Taming Cold Starts: Proactive Serverless Scheduling with Model Predictive Control (arXiv:2508.07640). arXiv. https://doi.org/10.48550/arXiv.2508.07640

*21. Gandhi, A., Doroudi, S., Harchol-Balter, M., & Scheller-Wolf, A. (2014). Exact analysis of the M/M/k/setup class of Markov chains via recursive renewal reward. *Queueing Systems*, *77*(2), 177–209. https://doi.org/10.1007/s11134-014-9409-7*

*22. Williams, J. K., Harchol-Balter, M., & Wang, W. (2022). The M/M/k with deterministic setup times. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, *6*(3), Article 51. https://doi.org/10.1145/3570617*

*23. Sachidananda, V., & Sivaraman, A. (2024). Erlang: Application-Aware Autoscaling for Cloud Microservices. In *Proceedings of the Nineteenth European Conference on Computer Systems (EuroSys)* (pp. 888–923). https://doi.org/10.1145/3627703.3650084*

*24. Cai, Z., & Buyya, R. (2022). Inverse Queuing Model-Based Feedback Control for Elastic Container Provisioning of Web Systems in Kubernetes. *IEEE Transactions on Computers*, *71*(2), 337–348. https://doi.org/10.1109/TC.2021.3049598*

*25. Eramo, V., Lavacca, F. G., Catena, T., & Salazar, P. J. P. (2021). Application of a Long Short Term Memory neural predictor with asymmetric loss function for the resource allocation in NFV network architectures. *Computer Networks*, *193*, 108104. https://doi.org/10.1016/j.comnet.2021.108104*

*26. Dartois, J., Knefati, A., Boukhobza, J., & Barais, O. (2018). Using Quantile Regression for Reclaiming Unused Cloud Resources While Achieving SLA. In *Proceedings of the 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. https://doi.org/10.1109/CloudCom2018.2018.00030*

27. Zhang, J., Wang, Y., & Hug, G. (2021). Cost-Oriented Load Forecasting (arXiv:2107.01861). arXiv. https://doi.org/10.48550/arXiv.2107.01861

28. Abolghasemi, M., & Bean, R. (2022). How to predict and optimise with asymmetric error metrics (arXiv:2211.13586). arXiv. https://doi.org/10.48550/arXiv.2211.13586

*29. Van Leeuwaarden, J. S. H., Mathijsen, B. W. J., & Zwart, B. (2019). Economies-of-scale in many-server queueing systems: Tutorial and partial review of the QED Halfin–Whitt heavy-traffic regime. *SIAM Review*, *61*(3), 403–440. https://doi.org/10.1137/17M1133944*

O. Shushura, D. Ihnatov

FORMALIZATION OF THE OPTIMAL KUBERNETES AUTOSCALING PROBLEM WITH POD STARTUP DELAY

Proactive Kubernetes autoscalers predict future workload and provision pods in advance; however, all existing systems share an implicit assumption that new capacity becomes available instantly or after a single fixed delay that is uniform across services. In polyglot deployments, however, startup delays can differ by an order of magnitude, making a single global time-shift correction inapplicable and creating a fundamental mismatch between the optimizer's decision and the actual availability of capacity.

This paper formalizes Kubernetes autoscaling as a discrete optimization problem with dead-time constraints, where the startup delay enters the effective capacity model as a per-service parameter. For the single-service case, discrete convexity of the objective function in the scaling decision is proved. The multi-service variant with a cluster resource budget is classified as an NP-hard bounded knapsack problem, for which three solution approaches are proposed: a greedy algorithm ranked by marginal efficiency, budget-state dynamic programming, and LP relaxation with rounding. The connection to the M/M/k/setup queueing model analytically establishes the fundamental limitation of reactive scaling: the waiting-time penalty decays only polynomially with the number of servers, which makes it impossible to compensate for the startup delay after a demand peak has arrived.

The structural asymmetry in the optimizer's response to forecast errors, induced by the dead-time constraint, is analyzed separately. It is shown that demand overestimation can be neutralized at decision time because the optimizer simply refrains from scaling, whereas the consequences of underestimation are irrecoverable: any corrective action takes effect only after the startup delay has elapsed, during which the system operates in a capacity deficit. This asymmetry is a structural property of the model and requires no modification of the objective function.

The theoretical results obtained provide a formal foundation for designing practical proactive autoscalers adapted to the heterogeneous environment of polyglot microservice systems.

Keywords: autoscaling, Kubernetes, microservices, startup delay, discrete optimization, load forecasting, dynamic programming, asymmetric loss.

Надійшла до редакції: 10.07.2026
Прийнята до друку: 24.07.2026
Опубліковано: 17.08.2026