Одеський національний університет імені І. І. Мечникова
Факультет математики, фізики та інформаційних технологій
Кафедра оптимального керування та економічної кібернетики

КУРСОВА РОБОТА

бакалавра

на тему: **«Оптимізація використання ресурсів у Kubernetes кластерах»**

Виконав: здобувач III курсу
денної форми навчання
спеціальності 113 Прикладна математика
Власенко Олег Григорович

Керівник:
ст. вч. Платонов В. В.

Одеса — 2025 р.

ЗМІСТ

Вступ 3

1 Актуальність задачі оптимізації ресурсів у Kubernetes кластерах 4
- 1.1 Техніко-економічні аспекти проблеми 4
- 1.2 Огляд існуючих підходів та інструментів 4
  - 1.2.1 Вбудовані механізми Kubernetes: Основні функції . . 4
  - 1.2.2 Інструменти моніторингу та візуалізації: Головне призначення 5
  - 1.2.3 Сторонні рішення та розробка власних інструментів: Сутність підходів 5

2 Математичний апарат для вирішення задачі оптимізації 6
- 2.1 Огляд ключових математичних дисциплін та їх застосування в контексті Kubernetes 6
  - 2.1.1 Теорія оптимізації (лінійне та цілочисельне програмування) 6
  - 2.1.2 Статистичний аналіз та теорія ймовірностей 7
  - 2.1.3 Теорія графів та дискретна математика 7
  - 2.1.4 Чисельні методи та математичне моделювання 7

3 Результати: Аналіз та оптимізація за допомогою розробленого програмного комплексу 8
- 3.1 Опис розробленого програмного комплексу 8
- 3.2 Аналіз даних та ідентифікація проблем (на прикладі Log2_1.txt) 8
- 3.3 Оцінка впливу запропонованих оптимізацій 9
- 3.4 Приклад оптимізації YAML-конфігурації (Додаток Б) 10
- 3.5 Висновки щодо програмного комплексу 11

Висновки 12

А Приклад виводу роботи програмного комплексу (Log2_1.txt) 13

Б Приклад YAML-конфігурації (після оптимізації) 14

В Глосарій англійських термінів 16

Список літератури 18

ВСТУП

Роль Kubernetes та проблема оптимізації ресурсів

Kubernetes (K8s) став стандартом оркестрації контейнерів, автоматизуючи розгортання та управління додатками в мікросервісних архітектурах. Зі зростанням складності систем, особливо з ресурсоємними завданнями ШІ/МН, ефективне управління ресурсами (ЦП, ОЗП) стає критичним. Неоптимальне використання призводить до фінансових втрат через надлишкове виділення (overprovisioning), зниження продуктивності через недостатнє виділення (underprovisioning) (CPU throttling, OOMKilled), та проблем масштабованості. Конфігураційна оптимізація, зокрема правильне налаштування запитів (requests) та лімітів (limits), є ключовою, але часто недооцінюється, особливо враховуючи, що Kubernetes за замовчуванням не обмежує ресурси подів без явних лімітів.

Постановка задачі дослідження

Проблема дослідження – виявлення неефективного використання ресурсів у Kubernetes через неоптимальні конфігурації та розробка підходів до їх оптимізації. Це включає ідентифікацію проблемних подів/вузлів, аналіз впливу на вартість та формування рекомендацій.

Мета – дослідити методи оптимізації та продемонструвати можливості розробленого програмного комплексу для аналізу конфігурацій на основі даних kubectl.

Завдання охоплюють аналіз актуальності, огляд інструментів, визначення математичного апарату, опис комплексу, аналіз його результатів, оцінку рішень та висновки.

Об'єкт – процеси управління ресурсами, предмет – методи аналізу конфігурацій за допомогою розробленого комплексу. Складність Kubernetes може ускладнити розуміння того, де саме можливе скорочення витрат.

РОЗДІЛ 1

АКТУАЛЬНІСТЬ ЗАДАЧІ ОПТИМІЗАЦІЇ РЕСУРСІВ У KUBERNETES КЛАСТЕРАХ

1.1 Техніко-економічні аспекти проблеми

Актуальність оптимізації ресурсів у Kubernetes зумовлена економічною ефективністю (значні витрати на хмарні ресурси, марнотратство до 47% бюджетів через неефективність), продуктивністю та надійністю додатків (недостатнє виділення ресурсів призводить до збоїв), масштабованістю системи (ефективне використання ресурсів дозволяє точніше масштабувати), екологічним аспектом (менше ресурсів – менший вуглецевий слід) та складністю сучасних додатків (ШІ/МН вимагають витонченого управління ресурсами). Оптимізація є комплексною проблемою на перетині технологій, економіки та бізнес-цілей.

1.2 Огляд існуючих підходів та інструментів

Існують вбудовані механізми Kubernetes, інструменти моніторингу/візуалізаці та сторонні/власні рішення.

1.2.1 Вбудовані механізми Kubernetes: Основні функції

Kubernetes надає:

- **Запити (Requests) та Ліміти (Limits):** Гарантована та максимальна кількість ресурсів для контейнера.
- **Класи Якості Обслуговування (QoS Classes):** Guaranteed, Burstable, BestEffort – визначають пріоритет поду.
- **Автоскейлери (HPA, VPA, CA):** Горизонтальний (HPA) масштабує кількість реплік, Вертикальний (VPA) оптимізує запити/ліміти подів, Автоскейлер Кластера (CA) регулює кількість вузлів. Їх узгоджене налаштування є складним завданням.
- **Стратегії Розміщення Подів:** nodeSelector, Affinity/AntiAffinity, Taints and Tolerations для контролю розміщення.

1.2.2 Інструменти моніторингу та візуалізації: Головне призначення

Для аналізу даних використовуються:

- **Metrics Server:** Базовий агрегатор метрик для kubectl top та HPA.
- **Prometheus:** Стандарт для збору метрик, надає мову запитів PromQL.
- **Grafana:** Платформа для візуалізації даних, часто з Prometheus.

Налаштування Prometheus/Grafana вимагає експертизи, що створює нішу для простіших інструментів аналізу на основі kubectl.

1.2.3 Сторонні рішення та розробка власних інструментів: Сутність підходів

- **Сторонні рішення:** Kubecost, Densify, CAST AI, Goldilocks, StormForge пропонують розширений аналіз, right-sizing, часто з використанням МН.
- **Розробка власних комплексів:** Дозволяє реалізувати специфічну логіку, як у даній роботі, фокусуючись на аналізі статичних даних kubectl для попереднього аудиту.

РОЗДІЛ 2

МАТЕМАТИЧНИЙ АПАРАТ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ ОПТИМІЗАЦІЇ

Оптимізація ресурсів у Kubernetes спирається на математичні дисципліни для моделювання, аналізу та пошуку рішень.

2.1 Огляд ключових математичних дисциплін та їх застосування в контексті Kubernetes

2.1.1 Теорія оптимізації (лінійне та цілочисельне програмування)

Лінійне (ЛП) та цілочисельне (ЦЛП) програмування формалізують задачі розподілу ресурсів, наприклад, вибір вузлів для мінімізації витрат за обмежень на ресурси. Цільова функція може виглядати так:

$$\text{Minimize } Z = \sum_{i=1}^{N} c_i x_i$$

де c_i – вартість і-го типу вузла, а x_i – кількість вузлів і-го типу. Обмеження можуть включати задоволення сумарних потреб у ресурсах, наприклад, для ЦП:

$$\sum_{i=1}^{N} R_{cpu,i} x_i \geq D_{cpu}$$

де $R_{cpu,i}$ – кількість ЦП, що надається вузлом і-го типу, а D_{cpu} – загальна потреба в ЦП. Опукла оптимізація використовується для складніших розподілів, де моделі можуть враховувати матрицю складу ресурсів K_{ri} (кількість ресурсу r, що надається інстансом типу і) та вектор попиту d_r для кожного ресурсу r:

$$\sum_{i=1}^{N} K_{ri} x_i \geq d_r \quad \forall r$$

Ці методи лежать в основі right-sizing та bin packing.

2.1.2 Статистичний аналіз та теорія ймовірностей

Статистика та теорія ймовірностей необхідні для аналізу даних використання ресурсів та характеризування навантажень. Розрахунок середніх значень, стандартних відхилень, процентилів (P_{95}) використовується для right-sizing. Формула для середнього:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

А для дисперсії (квадрата стандартного відхилення):

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

Аналіз часових рядів (ARIMA, Prophet) – для прогнозування навантаження та проактивного масштабування. Наприклад, модель Prophet може бути виражена як:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t$$

де $g(t)$ – тренд, $s(t)$ – сезонність, $h(t)$ – ефекти свят, а ϵ_t – похибка.

2.1.3 Теорія графів та дискретна математика

Теорія графів моделює задачі розміщення подів, наприклад, podAntiAffinity як розфарбовування графа. Задачі оптимізації потоків (min-cost max-flow) застосовуються у просунутих планувальниках (Firmament) для оптимального розміщення. Ці методи зазвичай описуються алгоритмічно, а не простими загальними формулами, придатними для короткого викладу.

2.1.4 Чисельні методи та математичне моделювання

Математичне моделювання створює абстрактні представлення системи Kubernetes для аналізу "що, якщо". Чисельні методи розв'язують складні моделі. Додавання буфера (напр., 15%) є евристикою, що може бути обґрунтована складнішими моделями. Наприклад, розрахунок потреби з буфером:

$$\text{Required}_{\text{buffer}} = \text{Required}_{\text{base}} \times (1 + \text{Percent buffer})$$

РОЗДІЛ 3

РЕЗУЛЬТАТИ: АНАЛІЗ ТА ОПТИМІЗАЦІЯ ЗА ДОПОМОГОЮ РОЗРОБЛЕНОГО ПРОГРАМНОГО КОМПЛЕКСУ

Розроблений програмний комплекс аналізує конфігурації Kubernetes та надає рекомендації, спираючись на логіку, що концептуально відповідає математичним підходам.

3.1 Опис розробленого програмного комплексу

Програмний комплекс (скрипт Python) аналізує статичні файли з виводом команд kubectl для виявлення проблем конфігурації та надання рекомендацій, зокрема щодо вибору типів вузлів. Ключові функції:

- **Збір даних:** Парсинг виводів ‘kubectl describe nodes/pods‘, ‘kubectl get pods -o wide‘.
- **Агрегація та аналіз:** Визначення доступних ресурсів, запитів/лімітів, сумарних показників на вузлах.
- **Виявлення проблем:** Ідентифікація CPU/Memory Limits Overcommit, подів з високими запитами, подів без лімітів/запитів за допомогою евристик.
- **Генерація рекомендацій:** Попередження, розрахунок сумарних потреб (з буфером, напр., 15

3.2 Аналіз даних та ідентифікація проблем (на прикладі Log2_1.txt)

Аналіз Log2_1.txt (Додаток А) показує:

- **Аналіз по вузлах:**
  - Вузол ‘ip-100-100-31-113...‘: CPU Limits Overcommit 153.1
  - Вузол ‘ip-100-100-41-106...‘: Под ‘xenia-stg/xenia-stg-mongodb-0‘ без лімітів CPU/пам’яті та з високими запитами CPU (52.3
- **Загальний аналіз та рекомендація інстансу**:
  - Сумарні запити кластера (CPU=3.03 cores, Пам’ять=2.25 GiB) + 15

Табл. 3.1. Порівняння розробленого програмного комплексу з іншими підходами

Характеристика	**РПК**	**Prometheus + Grafana**	**Kubernetes VPA**	**Комерційні рішення**
Джерело даних	Статичні файли (kubectl)	Метрики реального часу	Метрики реального часу, історичні дані	Метрики реального часу, дані про витрати історичні дані
Основний фокус	Аудит конфігурацій, базові помилки, початкові рекомендації	Моніторинг, візуалізація, алертинг	Автоматичне right-sizing подів	Комплексна оптимізація витрат, right-sizing, прогнозування (МН)
Аналіз споживання	Відсутній (лише аналіз запитів/лімітів)	Детальний аналіз реального споживання	Аналіз реального споживання для подів	Глибокий аналіз реального споживання часто з МН
Автоматизація дій	Лише рекомендації	Немає (потребує дій або інтеграції)	Можливе автоматичне застосування змін	Можлива автоматизація, інтеграція з CI/CD
Простота використання	Відносно простий (скрипт)	Потребує налаштування та експертизи	Інтегрується в Kubernetes	Залежить від рішення надають UI та підтримку

– Рекомендовано інстанс ‘a1.xlarge‘ (4 vCPU, 8 GiB RAM, \$0.1020/год) як найекономічніший одиничний варіант.

3.3 Оцінка впливу запропонованих оптимізацій

Виправлення виявлених проблем покращує стабільність, продуктивність та економічність.

- **Підвищення стабільності/продуктивності:** Встановлення лімітів (напр., для ‘xenia-stg-mongodb-0‘) запобігає OOMKilled. Коригування CPU Limits Overcommit знижує ризик CPU throttling. Оптимізація подів з "високими запитами"покращує "упаковку"подів (bin packing).
- **Оптимізація витрат:** Рекомендація інстансу ‘a1.xlarge‘ (0.1020/), $i.ii$(0.2256/ "Right-sizing"завищених запитів може дозволити Cluster Autoscaler зменшити кількість вузлів.

Табл. 3.2. Ключові проблеми, виявлені програмним комплексом, та їх математична інтерпретація

Проблема, виявлена РПК	**Математична/концептуальна інтерпретація**
CPU Limits Overcommit (153.1%)	Порушення обмеження ресурсів вузла. Аналогічно до перевірки обмежень в задачах лінійного програмування.
Под з високими запитами CPU (>50% вузла)	Статистична аномалія (викид), що може вказувати на неефективне використання ресурсів або проблеми з "bin packing".
Под без встановлених лімітів CPU/пам'яті	Відсутність верхніх меж для змінних споживання ресурсів в оптимізаційній моделі.
Рекомендація типу інстансу (a1.xlarge)	Розв'язання спрощеної задачі оптимізації: min(вартість) за умов CPU_інстансу ≥ потреба_CPU та Пам'ять_інстансу ≥ потреба_Пам'яті.

3.4 Приклад оптимізації YAML-конфігурації (Додаток Б)

Додаток Б показує оптимізований YAML для StatefulSet ‘xenia-stg-mongodb‘. Початковий стан: запит CPU 1010m, пам'яті 0; ліміти 0. Оптимізований YAML включає:

- Встановлення лімітів: ‘limits: cpu: "1 memory: "2Gi„ для стабільності.
- Коригування запитів CPU: ‘requests: cpu: "500m„.
- Встановлення запитів/лімітів Пам'яті: ‘requests: memory: "1Gi limits: memory: "2Gi„.

Для баз даних часто рекомендується ‘requests = limits‘ для QoS Guaranteed.

3.5 Висновки щодо програмного комплексу

Програмний комплекс – це інструмент для первинного аудиту статичних конфігурацій.

- **Сильні сторони:** Автоматизація обробки даних ‘kubectl‘, ідентифікація поширених помилок, базові рекомендації.
- **Обмеження:** Статичний аналіз (без реального споживання), спрощені рекомендації, відсутність МН.
- **Напрямки розвитку:** Інтеграція з Prometheus, розширені статистичні методи, МН, формальні оптимізаційні моделі.

ВИСНОВКИ

Дослідження підтвердило актуальність оптимізації ресурсів у Kubernetes. Оптимізація критична для запобігання фінансовим втратам та підвищення стабільності. Існує багато інструментів, і розробка власних, як у цій роботі, заповнює нішу швидкого аудиту. Математичний апарат, що включає теорію оптимізації та статистику, є основою для вирішення цих завдань. Розроблений комплекс корисний для первинного аудиту, але статичний аналіз має обмеження, оскільки не враховує реальне споживання. Існують значні можливості для розвитку через інтеграцію з системами моніторингу та застосування методів машинного навчання.

Практичні рекомендації:

- Регулярно проводити аудит конфігурацій.
- Встановлювати запити/ліміти на основі реального споживання.
- Використовувати Prometheus/Grafana для моніторингу.
- Розглядати VPA для автоматизації налаштувань.
- При виборі інфраструктури враховувати специфіку навантажень.
- Для складних сценаріїв застосовувати просунуті методи (МН, математична оптимізація).

ДОДАТОК А

ПРИКЛАД ВИВОДУ РОБОТИ ПРОГРАМНОГО КОМПЛЕКСУ (LOG2_1.TXT)

```
=== Завершення аналізу всіх файлів ===
--- Аналіз та Рекомендації (Фінальний звіт) ---
Аналіз по вузлах:
Вузол: ip-100-100-31-113.eu-west-1.compute.internal
INFO: Активні поди (1): xenia-stg/xenia-56b6dcb558-6hdql (Running)
Доступно (Allocatable): CPU=3.92 cores, Пам'ять=6.68 GiB, Поди=58
Невикористані ресурси (Allocatable - Requested): CPU=1.90 cores, Пам'ять=4.43 GiB
Сумарні Запити АКТИВНИХ (Requests): CPU=2.02 cores (51.5%), Пам'ять=2.25 GiB (33.7
Сумарні Ліміти АКТИВНИХ (Limits): CPU=6.00 cores (153.1%), Пам'ять=4.00 GiB (59.9%
WARNINGS:
CPU Limits Overcommit (153.1%). Можливий CPU throttling.
Поди з високими запитами (> 50.0%): xenia-stg/xenia-56b6dcb558-6hdql (CPU: 2.02 co

Вузол: ip-100-100-41-106.eu-west-1.compute.internal
INFO: Активні поди (1): xenia-stg/xenia-stg-mongodb-0 (Running)
Доступно (Allocatable): CPU=1.93 cores, Пам'ять=6.95 GiB, Поди=29
Невикористані ресурси (Allocatable - Requested): CPU=920m, Пам'ять=6.95 GiB
Сумарні Запити АКТИВНИХ (Requests): CPU=1.01 cores (52.3%), Пам'ять=0.00 MiB (0.0%
Сумарні Ліміти АКТИВНИХ (Limits): CPU=0m (0.0%), Пам'ять=0 MiB (0.0%)
WARNINGS:
Поди БЕЗ ЛІМІТІВ (limits): xenia-stg/xenia-stg-mongodb-0.
Поди з високими запитами (> 50.0%): xenia-stg/xenia-stg-mongodb-0 (CPU: 1.01 cores

Загальний аналіз кластера:
Загально доступно (Allocatable): CPU=5.85 cores, Пам'ять=13.64 GiB
Загальні Запити (Requests): CPU=3.03 cores (51.8%), Пам'ять=2.25 GiB (16.5%)
Загальні Ліміти (Limits): CPU=6.00 cores (102.6%), Пам'ять=4.00 GiB (29.3%)

--- Рекомендації щодо вибору ОДИНОЧНОГО типу EC2 інстансу ---
Загальні запити кластера (з буфером 15%): CPU=3.48 cores, Пам'ять=2.59 GiB
Найбільш економічно вигідний ОДИНИЧНИЙ інстанс:
- Тип інстансу: a1.xlarge
- CPU інстансу: 4.00 cores
- Пам'ять інстансу: 8.00 GiB
- Орієнтовна вартість: $0.1020 / год
```

ДОДАТОК Б

ПРИКЛАД YAML-КОНФІГУРАЦІЇ (ПІСЛЯ ОПТИМІЗАЦІЇ)

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
name: xenia-stg-mongodb
namespace: xenia-stg
spec:
serviceName: "xenia-stg-mongodb"
replicas: 1
selector:
matchLabels:
app: mongodb
role: database
environment: xenia-stg
template:
metadata:
labels:
app: mongodb
role: database
environment: xenia-stg
spec:
terminationGracePeriodSeconds: 10
containers:
- name: mongo
image: mongo:4.4.6 # Завжди вказуйте версію
ports:
- containerPort: 27017
name: mongo
resources:
requests:
cpu: "500m"
memory: "1Gi"
limits:
cpu: "1"
memory: "2Gi"
volumeMounts:
- name: mongo-persistent-storage
mountPath: /data/db
```

```
volumeClaimTemplates:
- metadata:
name: mongo-persistent-storage
spec:
accessModes: ["ReadWriteOnce"]
storageClassName: "gp2"
resources:
requests:
storage: 10Gi
```

ДОДАТОК В

ГЛОСАРІЙ АНГЛІЙСЬКИХ ТЕРМІНІВ

Affinity/Anti-affinity (Спорідненість/Антиспорідненість): Правила в Kubernetes, що впливають на розміщення подів на вузлах.

API (Application Programming Interface): Набір визначень та протоколів для взаємодії програм.

Bin packing ("Упаковка в контейнери"): Задача оптимального розміщення подів на вузлах.

CA (Cluster Autoscaler): Компонент Kubernetes, що автоматично змінює кількість вузлів.

CPU (Central Processing Unit): Центральний процесор; в Kubernetes вимірюється в ядрах (cores).

Deployment (Розгортання): Об'єкт Kubernetes, що декларативно описує бажаний стан подів.

EC2 (Elastic Compute Cloud): Сервіс Amazon Web Services (AWS), що надає віртуальні сервери.

FinOps (Financial Operations): Практика управління хмарними витратами.

Grafana: Платформа для візуалізації даних та моніторингу.

HPA (Horizontal Pod Autoscaler): Компонент Kubernetes, що автоматично масштабує кількість реплік поду.

Instance (Інстанс): Екземпляр віртуального сервера в хмарному середовищі.

Kubernetes (K8s): Відкрита платформа для автоматизації управління контейнеризованими додатками.

kubectl: Інструмент командного рядка для взаємодії з кластерами Kubernetes.

Limits (Ліміти): Максимально дозволена кількість ресурсів, яку може споживати контейнер.

ML (Machine Learning): Галузь штучного інтелекту, що дозволяє системам навчатися на даних.

Node (Вузол): Фізична або віртуальна машина в кластері Kubernetes.

OOMKilled (Out of Memory Killed): Примусове завершення процесу через нестачу пам'яті.

Orchestration (Оркестрація): Автоматизоване управління складними комп'ютерними системами.

Overprovisioning: Виділення більшої кількості ресурсів, ніж необхідно.

Pod (Под): Найменша одиниця розгортання в Kubernetes.

Prometheus: Система моніторингу та алертингу.

QoS (Quality of Service): Механізм Kubernetes для класифікації подів за пріоритетом.

Requests (Запити): Гарантована кількість ресурсів, яку Kubernetes резервує для контейнера.

Right-sizing: Процес налаштування запитів та лімітів відповідно до реальних потреб.

Scalability (Масштабованість): Здатність системи справлятися зі зростаючим навантаженням.

Scheduler (Планувальник): Компонент Kubernetes, який призначає поди на вузли.

StatefulSet: Об'єкт Kubernetes для управління додатками зі станом.

Throttling ("Тротлінг"): Штучне обмеження продуктивності.

Underprovisioning: Виділення меншої кількості ресурсів, ніж необхідно.

VPA (Vertical Pod Autoscaler): Компонент Kubernetes, що автоматично налаштовує запити та ліміти.

YAML: Людиночитний формат серіалізації даних, що використовується для конфігурацій.

СПИСОК ЛІТЕРАТУРИ

1. Apptio. Kubernetes Cost Optimization: A FinOps Guide.
2. Hamzeh, H. (2020). A Multi-Resource Fair Scheduling Algorithm in Cloud Computing. PhD Thesis. Bournemouth University.
3. Imdoukh, M., Ahmad, I., & Al-Failakawi, M. G. (2019). Proactive autoscaling for web applications using Prophet and LSTM. Frontiers in Computer Science.
4. Khan, A. Q. (2024). Cloud Cost Modelling and Optimisation: A Taxonomy and Approaches for Storage Object Classification and Cloud Resource Placement. PhD Thesis. Norwegian University of Science and Technology (NTNU).
5. Kubernetes Authors. Official Kubernetes Documentation.
6. Mohamed, A., & Sakr, S. (2021). Statistical Techniques for Characterizing Cloud Workloads: A Survey. International Journal of Cloud Applications and Computing (IJCAC).
7. Poseidon-Firmament Scheduler: Flow Network Graph-Based Scheduler for Kubernetes. Kubernetes Blog. (2019).
8. Rahman, M. M., & Al Dmour, I. (2025). A Hybrid Model-Based Proactive Autoscaling Architecture for Kubernetes Using Facebook Prophet and Long Short-Term Memory. Frontiers in Computer Science.
9. StormForge. Machine Learning for Kubernetes Resource Management. CloudBolt Blog.
10. The Moonlight. Cloud Resource Allocation with Convex Optimization.
11. Wang, Y.-C., et al. (2023). Delay-Aware Container Scheduling in Kubernetes. IEEE Internet of Things Journal.
12. Wiz. AI/ML on Kubernetes: Best Practices for Resource Management, Security, and Observability.
13. Yakubov, D., & Hästbacka, D. (2025). Empirical Analysis of Lightweight Kubernetes Distributions for Edge Computing. arXiv preprint.
14. Zhang, Q., et al. (2025). Distributedness based scheduling. arXiv preprint.
15. Debbi, H. (2021). Modeling and Performance Analysis of Resource Provisioning in Cloud Computing using Probabilistic Model Checking. Informatica, 45, 529–541.
16. Sedai. Bin Packing and Cost Savings in Kubernetes Clusters on AWS.
17. Spot by NetApp. Kubernetes Cost Optimization: Challenges and Best Practices.
18. Zou, J., et al. (2025). Optimizing Web Application Placement in Kubernetes with a Machine Learning Custom Scheduler. Applied Sciences.

19. Reddy, G. R., et al. (2020). Cloud resource optimization based on poisson linear deep gradient learning for mobile cloud computing. Journal of Intelligent & Fuzzy Systems.
20. Ghaffari, M. (2015). Near-Optimal Distributed Maximum Flow. Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC).
21. Tarjan, R. E., & Yannakakis, M. (1984). Scheduling Graphs on Two Processors. SIAM Journal on Computing.
22. Liu, Y., & Wu, M. (2017). Real-time scheduling techniques for distributed systems. Journal of Systems Architecture.
23. Chen, Y., et al. (2020). Load balancing techniques in distributed and cloud computing. ACM Computing Surveys.
24. Kumar, A. (2019). A Survey on Task Graph Scheduling Heuristic Algorithms in Distributed Computing System. International Journal of Engineering and Advanced Technology (IJEAT).
25. Shahidinejad, A., et al. (2023). Characterizing Web Application Workloads: A Systematic Literature Review and Clustering-Based Analysis. arXiv preprint.
26. Pogonip: A Scheduler for Asynchronous Microservices in Edge Computing Environments.
27. Kubernetes Authors. Managing Resources for Containers.