РЕАЛІЗАЦІЯ ОБЧИСЛЮВАЛЬНОГО КЛАСТЕРА APACHE SPARK НА БАЗІ МІКРОКОМП'ЮТЕРІВ RASPBERRY PI

Влах-Вигриновська Г.І., к.т.н., доцент, Борецький Б. І., бакалавр, студент
Національний університет "Львівська політехніка", Україна;
e-mail: bohdan.boretskyi.ir.2021@lpnu.ua

Анотація

У цій статті представлено імплементацію кластера Apache Spark для розподілених обчислень на базі мікрокомп'ютерів Raspberry Pi. Рішення складається з трьох пристроїв Raspberry Pi 4 (один головний вузол і два робочі вузли), кожен з 8 ГБ оперативної пам'яті та високошвидкісним мережевим з'єднанням. Конфігурація кластера оптимізована шляхом налаштування параметрів SPARK_WORKER_MEMORY та SPARK_WORKER_CORES для забезпечення максимальної ефективності доступних апаратних ресурсів. Захищене з'єднання між вузлами забезпечене через автентифікацію з використанням 4096-бітних SSH-ключів. Функціональність кластера перевірена за допомогою тестового застосунку, що показав ефективний розподіл обчислювального навантаження між вузлами. Вартість розробленого рішення – 400 доларів США, що в чотири рази менше, ніж вартість використання еквівалентних хмарних ресурсів протягом року. Результати дослідження підтверджують, що кластер на базі Raspberry Pi забезпечує всі необхідні можливості для практичного вивчення технологій розподілених обчислень, забезпечуючи фізичний доступ до всіх компонентів системи при недорогій собівартості.

Ключові слова

Apache Spark, розподілені обчислення, Raspberry Pi, мікрокомп'ютери, кластер, обробка великих даних.

1. Вступ

Розподілені обчислення є фундаментальною частиною сучасних інформаційних технологій, забезпечуючи ефективну обробку великих даних та виконання складних алгоритмів [1]. Комерційні хмарні рішення часто приховують низькорівневі аспекти функціонування кластера, ускладнюючи формування глибокого розуміння принципів організації розподілених систем.

Ця стаття досліджує можливість створення обчислювального кластера Apache Spark на базі мікрокомп'ютерів Raspberry Pi. Запропонований підхід поєднує низьку вартість апаратного забезпечення з гнучкістю налаштувань, що дозволяє адаптувати кластер до різних дослідницьких задач.

Актуальність цього рішення полягає в цінності прямого доступу до апаратних компонентів системи, що дозволяє вивчати всі рівні її організації - від фізичного розташування вузлів та мережевої інфраструктури до конфігурації програмного забезпечення й оптимізації параметрів.

Повне розгортання кластера «з нуля», яке охоплює інсталяцією операційної системи, конфігурацію мережі, забезпечення безпечного доступу, а також налаштування програмного стеку Apache Spark, сприяє глибшому розумінню взаємодії компонентів у розподіленому середовищі та принципів його функціонування [2].

2. Недоліки

Сучасні підходи до вивчення технологій розподілених обчислень мають низку обмежень:

1. Приховування низькорівневих деталей інфраструктури та відсутність фізичної взаємодії з апаратними компонентами в хмарних рішеннях [3].
2. Висока вартість використання хмарних сервісів у порівнянні з розгортанням фізичних обчислювальних потужностей [4], [5].
3. Відсутність можливості моніторингу теплового режиму та енергоспоживання при хмарних обчисленнях [6].
4. Обмежені інструменти для налаштування мережевої інфраструктури [7].

3. Мета

Метою статті є розробка економічно доступного, енергоефективного та повнофункціонального обчислювального кластера для розподілених обчислень з використанням Apache Spark на базі мікрокомп'ютерів Raspberry Pi. Для досягнення цієї мети необхідно вирішити такі завдання:

- Спроектувати та побудувати архітектуру кластерної системи на базі мікрокомп'ютерів Raspberry Pi, що реалізує необхідну функціональність розподілених обчислень.

- Встановити та налаштувати мережеву інфраструктуру для стабільного зв'язку між компонентами кластера та їх доступу до інтернету.
- Налаштувати програмні компоненти Apache Spark з параметрами, які оптимізовані для ефективного використання обмежених апаратних ресурсів Raspberry Pi.
- Налаштувати доступ до вузлів кластера через асиметричні SSH-ключі.
- Провести експериментальну перевірку роботи кластера на базовому завданні обробки даних для перевірки його функціональності та ефективності.
- Порівняти витрати реалізованого рішення з витратами на еквівалентні хмарні сервіси.

4. Огляд літературних джерел

Розподілені обчислення – це модель обробки даних, у якій завдання виконуються паралельно на кількох взаємопов'язаних вузлах з метою підвищення продуктивності та масштабованості при роботі з великими обсягами даних [1]. У таких системах завдання розподіляються між вузлами, що дозволяє масштабувати обчислення, підвищити швидкість обробки даних та забезпечити безперервну роботу навіть у разі відмови окремих компонентів. Найчастіше використовується кластерна архітектура з централізованим керуванням ресурсами та задачами [8].

Одним із провідних фреймворків для розподілених обчислень є Apache Spark - високопродуктивний рушій з відкритим кодом, що надає високорівневі API для Java, Scala, Python та R [9]. На відміну від традиційних рішень, які зберігають проміжні результати на диску [10], Spark використовує оперативну пам'ять, що дозволяє досягти значно вищої швидкості обробки даних. Його архітектура включає ядро Spark Core та низку спеціалізованих бібліотек: Spark SQL для обробки структурованих даних; MLlib для машинного навчання; GraphX для обчислень на графах і Structured Streaming для потокової обробки даних, що дозволяє використовувати платформу для широкого спектра задач.

Концепція стійких розподілених наборів даних (RDD) [11] стала основою архітектури Spark. RDD забезпечують абстракцію, яка дозволяє ефективно відновлюватися після збоїв без запису проміжних результатів на диск, значно підвищуючи продуктивність системи порівняно з більш ранніми реалізаціями розподілених обчислень. Spark виконує ітеративні задачі обробки даних до 26 разів швидше за MapReduce, тому є оптимальним рішенням для задач машинного навчання та інтерактивної аналітики.

Архітектура Apache Spark реалізує модель "master-worker", у якій центральну роль виконує Cluster Manager - компонент, що забезпечує координацію комунікації та розподіл обчислювальних завдань між виконавцями (workers), що функціонують на окремих вузлах [11], [12]. Такий підхід забезпечує масштабованість та відмовостійкість системи при зростанні навантаження. Платформа підтримує кілька режимів розгортання, зокрема Standalone, YARN, Mesos і Kubernetes, що дозволяє адаптувати інфраструктуру відповідно до технічних обмежень та вимог користувача.

Існуючі реалізації кластерів на базі Raspberry Pi підтверджують ефективність цієї платформи для розподілених обчислень. Зокрема, TopADDPi - енергоефективний кластер для задач топологічної оптимізації, побудований на відкритому програмному забезпеченні, демонструє високу продуктивність при мінімальних витратах [13]. У роботі [14] представлено кластер із 25 Raspberry Pi 2B, орієнтований на вивчення співвідношення між енергоспоживанням і обчислювальною продуктивністю в освітньому середовищі. Також у [13] розглядається середовище, що поєднує Raspberry Pi та Google Colab для викладання основ паралельних і гетерогенних обчислень, підкреслюючи гнучкість і доступність цієї апаратної платформи.

Додаткове підтвердження освітньої цінності таких рішень наведено в [15], де продемонстровано ефективне використання кластерів Raspberry Pi у навчанні базових концепцій паралельних та розподілених обчислень. Запропоновані навчальні модулі включають попередньо налаштоване програмне середовище, яке дозволяє швидко розгортати повноцінні кластери без складної підготовки інфраструктури. Результати практичного застосування засвідчили зростання рівня розуміння студентами фундаментальних принципів роботи кластерних систем і розвиток практичних навичок у цій галузі.

5. Методологія

5.1 Налаштування операційної системи для вузлів кластера

Для розгортання операційної системи використовувався офіційний інструмент Raspberry Pi Imager для запису Raspberry Pi OS (64-біт) на MicroSD-карти. Під час встановлення було попередньо налаштовано ключові параметри, включаючи облікові дані користувача та активацію доступу за допомогою SSH.

Статична IP-адресація для вузлів кластера була реалізована шляхом модифікації файлу cmdline.txt безпосередньо на MicroSD-носіях. IP-адреса вказується за допомогою параметра:

ip=<static-ip>::<gateway-ip>:<netmask>:<hostname>:<interface> hostname=<system-hostname>

де:

- <static-ip> - статична IP-адреса, призначена вузлу
- <gateway-ip> - IP-адреса маршрутизатора/шлюзу для доступу до інтернету
- <netmask> - визначає діапазон підмережі
- <hostname> - назва мережевої ідентифікації
- <interface> - вказує мережевий інтерфейс (eth0 для Ethernet)
- hostname=<system-hostname> - встановлює ім'я хоста операційної системи

До файлу cmdline.txt головного вузла було додано:
ip=192.168.137.2::192.168.137.1:255.255.255.0:raspberrypi-master hostname=raspberrypi-master
Для робочого вузла 1:
ip =192.168.137.3::192.168.137.1:255.255.255.0:raspberrypi-worker1 hostname=raspberrypi-worker1
Для робочого вузла 2:
ip =192.168.137.4::192.168.137.1:255.255.255.0:raspberrypi-worker2 hostname=raspberrypi-worker2
Ця конфігурація забезпечує статичну IP-адресу кожного вузла при перезавантаженні та забезпечує належну мережеву ідентифікацію в межах кластера.

5.2 Фізична конфігурація кластера

Фізична організація кластера включала розміщення трьох пристроїв Raspberry Pi 4 на багаторівневій підставці для ефективного охолодження (рис.1). Кожен пристрій живиться від окремого джерела живлення 5В/3А. Мережева інфраструктура реалізована за допомогою 5-портового маршрутизатора Mercury з кабелями Ethernet Cat 6, які забезпечують швидкість до 1 Гбіт/с. Один Raspberry Pi функціонує як головний вузол, а два інші - як робочі вузли. Клієнтський ноутбук також підключений до мережі для управління кластером та доступу до інтернету через функцію спільного використання інтернет-з'єднання (ICS). Це дозволяє всім вузлам Raspberry Pi використовувати інтернет-з'єднання ноутбука. Правильність мережевих з'єднань була підтверджена індикаторами активності на портах маршрутизатора.

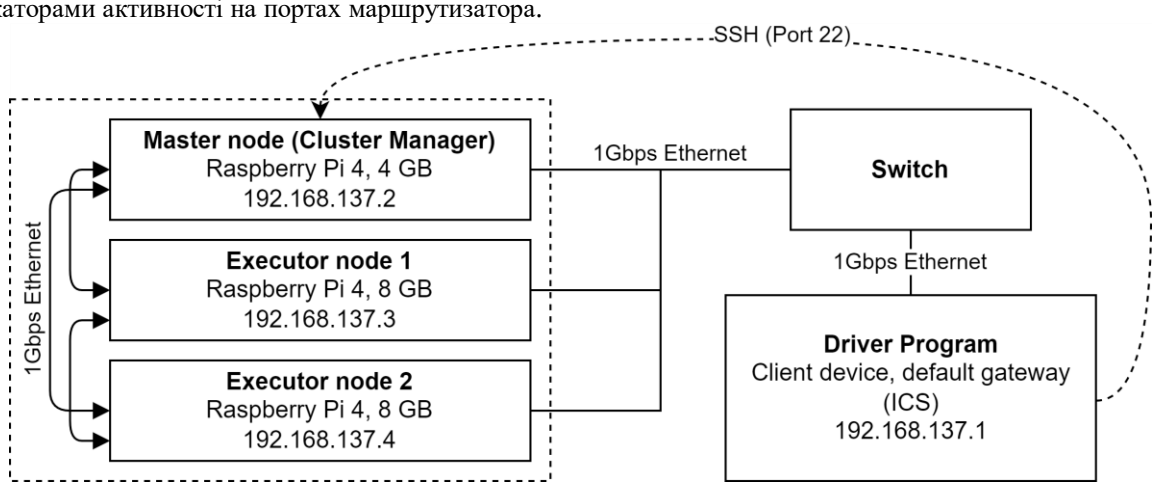

Рис. 1. Фізична та мережева архітектура кластера

5.3 Налаштування SSH та безпечного доступу

Автентифікація на основі асиметричних SSH-ключів була налаштована для управління кластером з клієнтського ноутбука. Спочатку використовувалась автентифікація за паролем для підключення клієнтського пристрою до головного вузла (ssh 192.168.137.2). Після цього на клієнтському пристрої були згенеровані RSA-ключі довжиною 4096 біт (ssh-keygen -t rsa -b 4096), публічний ключ передано на головний вузол і додано до списку авторизованих. Ті самі кроки виконано для кожного з двох робочих вузлів.
Для забезпечення безперервної взаємодії між усіма компонентами кластера додатково налаштовано безпарольну автентифікацію. На головному вузлі створено окрему пару SSH-ключів, публічну частину яких передано на обидва робочі вузли за допомогою утиліти ssh-copy-id. Коректність конфігурації перевірено через тестові підключення до кожного з вузлів.

5.4 Налаштування спільного використання інтернет-з'єднання

Щоб забезпечити доступ Raspberry Pi до інтернету, було використано функцію спільного доступу до мережі (ICS), яка доступна в операційній системі Windows. У параметрах мережевого адаптера з активним інтернет-з'єднанням було увімкнено опцію “Дозволити іншим користувачам мережі підключатися через інтернет-з'єднання цього комп’ютера”, що забезпечило маршрутизацію та трансляцію адрес між локальною мережею кластера (192.168.137.0/24) та зовнішньою мережею. На всіх вузлах Raspberry Pi для шлюзу за замовчуванням та DNS-сервера було вказано адресу 192.168.137.1. Підключення до інтернету використовувалося для оновлення системних пакетів та встановлення необхідного програмного забезпечення, зокрема Apache Spark.

5.5 Встановлення необхідного програмного забезпечення

Необхідне програмне забезпечення було розгорнуто на всіх вузлах кластера після оновлення системних репозиторіїв (apt upgrade). Крім OpenJDK 17 та Python 3.12, було встановлено Apache Spark версії 3.5.5 як основний фреймворк для розподілених обчислень. Для справного функціонування системи були налаштовані відповідні змінні середовища: SPARK_HOME з посиланням на каталог встановлення, системну змінну PATH оновлено для доступу до утиліт Spark через командний рядок, також встановлено значення JAVA_HOME для правильної взаємодії з віртуальною машиною Java [16].

5.6 Конфігурація кластера Spark

Для створення ефективного обчислювального середовища було виконано конфігурацію вузлів кластера відповідно до їх ролей. На головному вузлі кластера, який виконує функції Master (Cluster manager), було

налаштовано параметри SPARK_MASTER_HOST зі значенням IP-адреси вузла, SPARK_MASTER_PORT (7077) та SPARK_DRIVER_MEMORY (3ГБ) для ефективного управління обчислювальними процесами.
На робочих вузлах, які мають більше оперативної пам'яті (8ГБ), встановлено параметри SPARK_WORKER_MEMORY (6ГБ) та SPARK_WORKER_CORES (4) [16]. Ці значення є оптимальними для того щоб максимально використовувати доступні апаратні ресурси для паралельних обчислень. Усі параметри конфігурації були додані до файлу spark-env.sh на відповідних вузлах для збереження значень параметрів між сеансами запуску кластера та забезпечення їх стабільної роботи в кластерному середовищі.

5.7 Запуск та тестування кластера

Запуск кластера Spark може здійснюватися двома різними способами.
Перший метод передбачає поетапний запуск компонентів. Спочатку на головному вузлі запускається служба Master:
start-master.sh
Після успішного запуску вузла Master на кожному робочому вузлі ініціюється процес Worker з зазначенням адреси вузла Master:
start-worker.sh spark://192.168.137.2:7077
Ця команда забезпечує підключення робочого вузла до драйвера через протокол spark:// на порту 7077.
Альтернативним і швидшим методом є використання команди для одночасного запуску всіх компонентів кластера:
start-all.sh
Ця команда автоматично запускає процес Master на локальній машині та процеси Worker на всіх вузлах, вказаних у конфігураційному файлі conf/workers. Це значно спрощує запуск кластера, особливо при його регулярному використанні. Підтвердження успішного розгортання кластера відбувається через веб-інтерфейс Spark Master, доступний за адресою http://192.168.137.2:8080, який надає інформацію про підключені робочі вузли та їх обчислювальні ресурси.
Варто значити, що команда start-worker.sh також приймає додаткові параметри для більш гнучкого налаштування ресурсів, зокрема: -c для визначення кількості ядер та -m для обсягу оперативної пам'яті.

6. Результати

6.1 Функціональне тестування

Реалізований обчислювальний кластер був успішно протестований запуском базового застосунку, який виконує паралельне обчислення суми чисел від 1 до 999 999. Нижче наведений код мовою Python, використаний для тестування:

```
import logging
import os
from pyspark.sql import SparkSession
from pyspark.sql.functions import sum

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(levelname)s - %(message)s')
logger = logging.getLogger("ClusterTest")

def main():
    os.environ['SPARK_LOCAL_IP'] = '192.168.137.1'

    spark = SparkSession.builder \
        .appName("HelloWorldInSpark") \
        .master("spark://192.168.137.2:7077") \
        .config("spark.driver.host", "192.168.137.1") \
        .getOrCreate()

    nums_df = spark.range(1, 1000000)
    result_df = nums_df.select(sum("id").alias("sum"))

    logger.info(f"Result:")
    result_df.show()

    spark.stop()

if __name__ == '__main__':
    main()
```

Застосунок був запущений з клієнтського пристрою (особистого ноутбука) з використанням бібліотеки PySpark, налаштованої для підключення до головного вузла кластера. Spark успішно розподілив обчислювальне навантаження між робочими вузлами та завершив обробку даних, повернувши на клієнтський пристрій результат 499 999 500 000, що підтверджує функціональність розгорнутого кластера. Моніторинг виконання

завдання через веб-інтерфейс Spark Master продемонстрував ефективне використання ресурсів обох робочих вузлів.

Виконання Spark-застосунку на розробленому кластері слідує чітко визначеному алгоритму (рис.2), який демонструє взаємодію між компонентами системи [12]. Процес починається на клієнтському пристрої з формування логічного плану обчислень та його оптимізації. За цим слідує розподіл атомарних завдань між робочими вузлами через вузол Master, паралельне виконання обчислень на вузлах-виконавцях та агрегація результатів на клієнтському пристрої.

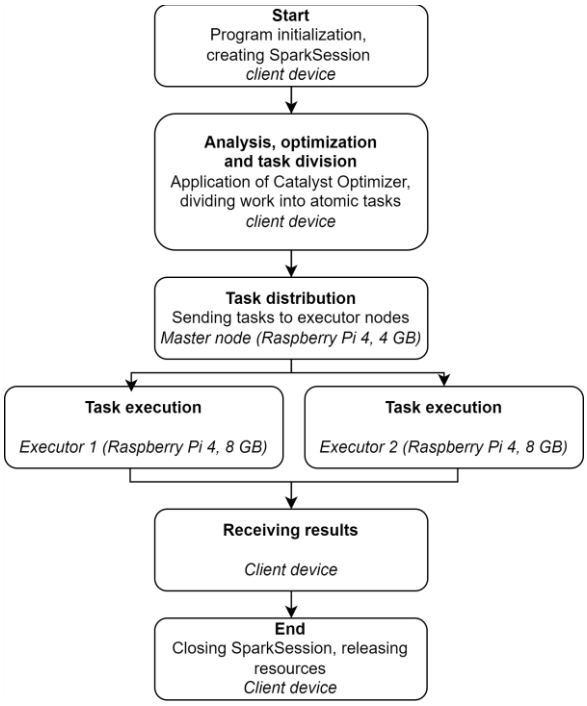

Рис. 2. Алгоритм виконання Spark-застосунку

6.2 Аналіз витрат: кластер Raspberry Pi проти хмарного середовища AWS

Однією з важливих переваг зібраного кластера є його низька вартість порівняно з комерційними хмарними рішеннями. У таблиці 1 представлено структуру витрат для побудови такої системи.

Табл. 1. Витрати на реалізацію кластера Raspberry Pi

Компонент	Кількість	Ціна за одиницю ($)	Загальна ціна ($)
Raspberry Pi 4 (8ГБ ОЗП)	2	95	190
Raspberry Pi 4 (4ГБ ОЗП)	1	75	75
MicroSD-карти (64ГБ)	3	15	45
Блоки живлення (5V/3A)	3	10	30
Маршрутизатор	1	25	25
Кабелі Ethernet Cat 6	3	5	15
Багаторівнева підставка для охолодження	1	20	20
Всього			**$400**

Для оцінки економічної доцільності створеного рішення було виконано порівняння витрат із вартістю використання аналогічних обчислювальних ресурсів у хмарному середовищі AWS EC2. Для цього підібрано еквівалентні віртуальні машини типу Graviton2, які базуються на ARM архітектурі:

- для головного вузла (4ГБ ОЗП, 4 ядра) - t4g.medium
- для вузлів-виконавців (8ГБ ОЗП, 4 ядра) - t4g.large

Використання ARM-процесорів у серії t4g дозволяє коректніше порівнювати їх із Raspberry Pi, ніж традиційними x86-архітектурпми. Незважаючи на те, що Raspberry Pi має 4 фізичних ядра, а t4g -лише 2 vCPU [17], сучасніші ядра Graviton2 з вищою тактовою частотою забезпечують співставний рівень продуктивності.

На основі цін AWS [17] t4g.medium ($0,0336 за годину використання) та t4g.large ($0,0672 за годину) витрати на експлуатацію еквівалентної конфігурації кластера протягом одного року розраховані в таблиці 2:

Табл. 2. Річні витрати на еквівалентні віртуальні машини AWS EC2

Компонент	Місячні витрати ($)	Річні витрати ($)
t4g.medium (master)	24.54	294.53
t4g.large (worker) × 2	49.09 × 2	1178.10
Data transfer costs (прибл.)	15.00	180.00
Всього		**$1652.63**

Порівняння витрат показує, що кластер Raspberry Pi вартістю $400 приблизно в 4 рази дешевший, ніж еквівалентні хмарні сервіси AWS за період експлуатації один рік.

7. Висновки

В результаті дослідження було успішно реалізовано функціональний обчислювальний кластер Apache Spark на базі мікрокомп'ютерів Raspberry Pi. Розгорнутий кластер продемонстрував можливість практичного застосування для вивчення технологій розподілених обчислень та проведення досліджень на економічно доступній платформі.

Розроблений кластер складається з трьох вузлів Raspberry Pi 4, включаючи один головний вузол та два робочі вузли. Кожен пристрій у конфігурації кластера має 64 ГБ сховища на MicroSD-картах. Робочі вузли оснащені 8 ГБ оперативної пам'яті та чотириядерними процесорами Cortex-A72.

Для забезпечення ефективної роботи кластера було оптимізовано параметри конфігурації Apache Spark. Мережева інфраструктура кластера реалізована з використанням 5-портового маршрутизатора Mercury. Для безпечного доступу до вузлів було реалізовано механізм автентифікації на основі 4096-бітних асиметричних SSH-ключів.

Функціональність розробленого кластера було підтверджено успішним виконанням тестового Spark-застосунку, який виконує паралельне обчислення суми чисел від 1 до 999 999. Аналіз процесу виконання показав, що обчислювальне навантаження ефективно розподіляється між робочими вузлами, а результати коректно агрегуються на головному вузлі.

З фінансової точки зору, кластер Raspberry Pi є хорошою альтернативою хмарним рішенням, оскільки його компоненти коштують приблизно $400, що в більш ніж 4 рази менше у порівнянні з ціною використання еквівалентних за потужністю віртуальних машини AWS EC2 протягом одного року.

Особлива цінність розробленого кластера полягає в його доступності як освітнього інструменту для практичного вивчення принципів розподілених обчислень. На відміну від віртуальних або хмарних рішень, фізичний кластер забезпечує можливість безпосередньої взаємодії з усіма компонентами системи.

Конфлікт інтересів

Автори заявляють, що фінансових чи інших потенційних конфліктів щодо цієї роботи немає.

Список використаних джерел

[1] F. Dai, M. A. Hossain, and Y. Wang, “State of the Art in Parallel and Distributed Systems: Emerging Trends and Challenges,” *Electronics*, vol. 14, no. 4, p. 677, Feb. 2025, doi: 10.3390/electronics14040677.

[2] V. Thesma, G. C. Rains, and J. Mohammadpour Velni, “Development of a Low-Cost Distributed Computing Pipeline for High-Throughput Cotton Phenotyping,” *Sensors*, vol. 24, no. 3, p. 970, Feb. 2024, doi: 10.3390/s24030970.

[3] A. Alakuu and D. K. Dake, “Cloud Computing in Education: A review of Architecture, Applications, and Integration Challenges,” *IJCA*, vol. 186, no. 66, pp. 49–65, Feb. 2025, doi: 10.5120/ijca2025924472.

[4] S. Younus, K. Kumar, I. A. Kandhro, A. A. Laghari, and A. Ali, “Systematic Analysis of On Premise and Cloud Services,” *IJCC*, vol. 13, no. 3, p. 10063641, 2024, doi: 10.1504/IJCC.2024.10063641.

[5] A. A. Abdulle, A. Farah Ali, and R. H. Abdullah, “Cost-Benefit Analysis of Public Cloud Versus In-House Computing,” *IJETT*, vol. 70, no. 6, pp. 300–307, Jun. 2022, doi: 10.14445/22315381/IJETT-V70I6P231.

[6] A. Katal, S. Dahiya, and T. Choudhury, “Energy efficiency in cloud computing data centers: a survey on software technologies,” *Cluster Comput*, vol. 26, no. 3, pp. 1845–1875, Jun. 2023, doi: 10.1007/s10586-022-03713-0.

[7] G. Agapito and M. Cannataro, “An Overview on the Challenges and Limitations Using Cloud Computing in Healthcare Corporations,” *BDCC*, vol. 7, no. 2, p. 68, Apr. 2023, doi: 10.3390/bdcc7020068.

[8] P. K. Donta, I. Murturi, V. Casamayor Pujol, B. Sedlak, and S. Dustdar, “Exploring the Potential of Distributed Computing Continuum Systems,” *Computers*, vol. 12, no. 10, p. 198, Oct. 2023, doi: 10.3390/computers12100198.

[9] “Spark Overview.” Apache Software Foundation. [Online]. Available: https://spark.apache.org/docs/latest/

[10] P. Sewal and Hari Singh, “Performance Comparison of Apache Spark and Hadoop for Machine Learning based iterative GBTR on HIGGS and Covid-19 Datasets,” *SCPE*, vol. 25, no. 3, pp. 1373–1386, Apr. 2024, doi: 10.12694/scpe.v25i3.2687.

[11] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica., “Spark: Cluster Computing with Working Sets,” 2010. [Online]. Available: https://www.usenix.org/legacy/event/hotcloud10/tech/full_papers/Zaharia.pdf

[12] N. Ahmed, A. L. C. Barczak, M. A. Rashid, and T. Susnjak, “A parallelization model for performance characterization of Spark Big Data jobs on Hadoop clusters,” *J Big Data*, vol. 8, no. 1, p. 107, Dec. 2021, doi: 10.1186/s40537-021-00499-7.

[13] Z.-D. Zhang *et al.*, “TopADDPi: An Affordable and Sustainable Raspberry Pi Cluster for Parallel-Computing Topology Optimization,” *Processes*, vol. 13, no. 3, p. 633, Feb. 2025, doi: 10.3390/pr13030633.

[14] M. Cloutier, C. Paradis, and V. Weaver, “A Raspberry Pi Cluster Instrumented for Fine-Grained Power Measurement,” *Electronics*, vol. 5, no. 4, p. 61, Sep. 2016, doi: 10.3390/electronics5040061.
[15] E. Shoop, S. J. Matthews, R. Brown, and J. C. Adams, “Hands-on parallel & distributed computing with Raspberry Pi devices and clusters,” *Journal of Parallel and Distributed Computing*, vol. 196, p. 104996, Feb. 2025, doi: 10.1016/j.jpdc.2024.104996.
[16] “Spark Configuration.” Apache Software Foundation. [Online]. Available: https://spark.apache.org/docs/latest/configuration.html
[17] “Amazon EC2 On-Demand Pricing.” AWS. [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/