https://doi.org/10.31891/2307-5732-2026-365-8
УДК 004.8

ГУСАК ВІТАЛІЙ
Національний Університет «Львівська політехніка»
https://orcid.org/0009-0002-8415-2767
e-mail: vitalii.husak.asp.2025@lpnu.ua

АНАЛІТИЧНИЙ ОГЛЯД МЕТОДІВ І ТЕХНОЛОГІЙ ОПРАЦЮВАННЯ ВЕЛИКИХ ОБСЯГІВ ДАНИХ У ДЕЦЕНТРАЛІЗОВАНИХ СИСТЕМАХ

У статті виконано аналітичний огляд методів і технологій опрацювання великих обсягів даних у децентралізованих системах. Розглянуто ключові інженерні компроміси застосунків, що вимагають обробки великої кількості даних, зокрема масштабованість, узгодженість, надійність, продуктивність і здатність зберігати працездатність за умов зростання обсягів та швидкості надходження даних. Пояснено логіку побудови наскрізних конвеєрів даних, що включають приймання, зберігання, перетворення та надання даних споживачам з урахуванням операційних обмежень. Окрему увагу приділено практичному оцінюванню систем обробки даних через характеристики очікуваного робочого навантаження та аналіз поведінки системи при масштабуванні: як змінюється продуктивність при фіксованих ресурсах та скільки ресурсів потрібно додати для збереження цільових показників. Підкреслено різницю між критеріями оцінювання пакетних систем, зокрема пропускною здатністю та часом виконання задач, і онлайн-сервісів, для яких ключовою є затримка відповіді.

Описано базові структурні компоненти сучасних конвеєрів даних, зокрема сховища, кеші, індексація, обмін повідомленнями, потокова й пакетна обробка, а також підходи до оцінювання продуктивності через характеристики робочого навантаження і ресурсні обмеження. Окрему увагу приділено огляду технологій Apache Kafka, Hadoop, Spark та Flink як інструментів побудови масштабованих систем з обробкою подій у реальному часі й пакетною аналітикою, а також їх ролі у децентралізованих середовищах великих даних.

Ключові слова: *великі дані, децентралізовані системи, розподілені системи, data-intensive системи, масштабованість, відмовостійкість, узгодженість, потокова обробка, пакетна обробка, конвеєри даних, Apache Kafka, Hadoop, MapReduce, Apache Spark, Apache Flink.*

HUSAK VITALII
Lviv Polytechnic National University

ANALYTICAL REVIEW OF METHODS AND TECHNOLOGIES FOR PROCESSING LARGE VOLUMES OF DATA IN DECENTRALIZED SYSTEMS

The article presents an analytical review of methods and technologies for processing large amounts of data in decentralized systems. The key engineering trade-offs of data-intensive applications are considered, in particular, scalability, consistency, reliability, performance, and the ability to maintain operability under conditions of increasing volumes and data rate. The logic of building end-to-end data pipelines is explained, including data ingestion, storage, transformation, and delivery to consumers, taking operational constraints into account.

Special attention is paid to the practical evaluation of data processing systems through the characteristics of the expected workload and the analysis of the system behavior during scaling: how performance changes with fixed resources and how many resources need to be added to maintain target indicators. The difference between the evaluation criteria of batch systems, in particular bandwidth and task execution time, and online services, for which response support is key, is noted.

The structural components of modern decentralized data architectures are systematically described, including distributed storage systems, caching layers, indexing mechanisms, messaging brokers, stream processing engines, and batch analytics frameworks. Particular attention is devoted to technologies such as Apache Kafka, Apache Hadoop, Apache Spark, and Apache Flink. Their architectural models, processing semantics (at-most-once, at-least-once, exactly-once), and suitability for real-time versus batch workloads are comparatively analyzed. The role of these platforms in decentralized big data ecosystems is evaluated with respect to elasticity, state management, checkpointing, and fault tolerance.

The article concludes that effective processing of large-scale data in decentralized systems requires an integrated approach that combines architectural modularity, workload-aware optimization, and continuous performance evaluation. The synthesis of streaming and batch paradigms, along with adaptive resource management strategies, forms the technological foundation for resilient and scalable big data infrastructures.

Keywords: *Big Data, decentralized systems, distributed systems, data-intensive applications, scalability, fault tolerance, consistency, stream processing, batch processing, data pipelines, Apache Kafka, Hadoop, MapReduce, Apache Spark, Apache Flink.*

Стаття надійшла до редакції / Received 23.02.2026
Прийнята до друку / Accepted 13.03.2026
Опубліковано / Published 28.05.2026

Постановка проблеми у загальному вигляді та її зв'язок із важливими науковими чи практичними завданнями

Подальше вдосконалення і зростання кількості обробки великих даних у централізованих системах дедалі частіше стикаються з обмеженнями щодо масштабованості, стійкості до відмов, безпеки, конфіденційності та ефективності використання ресурсів. У зв'язку з цим значної актуальності набувають децентралізовані системи, які розподіляють обчислення, зберігання та управління даними між багатьма вузлами мережі. У зв'язку з цим, автор вбачає потребу в систематичному аналітичному огляді сучасних методів і технологій опрацювання великих обсягів даних у децентралізованих системах, їх порівнянні, класифікації та оцінюванні з точки зору ефективності, надійності, масштабованості та безпеки.

Розв'язання цієї проблеми має важливе значення як для розвитку теоретичних засад розподілених обчислень, так і для практичного впровадження децентралізованих рішень у сферах промисловості, фінансів,

енергетики, смарт-інфраструктури та цифрового врядування.

Аналіз досліджень та публікацій

Проєктування сучасних систем дедалі більше формується обмеженнями, пов'язаними з даними. Багато найскладніших інженерних компромісів станом на сьогодні обертаються навколо масштабованості, узгодженості, надійності, ефективності та зручності обслуговування, особливо зі зростанням обсягів даних, збільшенням кількості сутностей і зв'язків між ними, гетерогенністю даних та зростанням частоти їх оновлень. Водночас інженери повинні орієнтуватись в широкій екосистемі технологій – реляційних базах даних, сховищах даних NoSQL, розподілених файлових системах, фреймворках потокової та пакетної обробки, а також брокерах повідомлень – вибираючи комбінації, що відповідають робочому навантаженню та вимогам до коректної роботи програми.

Застосунок, який використовує або обробляє велику кількість даних (data-intensive), можна вважати такий, коли його основною проблемою є не складність обчислень, а кількість даних, їх складність або швидкість їх зміни [1]. Більшість систем, що використовують багато даних, зібрані зі спільних структурних блоків, які забезпечують повторювану функціональність: постійне сховище для подальшого отримання інформації – бази даних; кешування для повторного використання результатів ресурсозатратних операцій та зменшення затримки читання; пошук та індексування для підтримки запитів за ключовими словами та гнучкої фільтрації; обмін повідомленнями або потокове передавання для забезпечення асинхронного зв'язку та обробки на основі подій; та пакетна обробка для періодичного обчислення агрегатів або похідних наборів даних на основі великих історичних даних. Ці компоненти зазвичай об'єднуються в наскрізні конвеєри, які приймають, зберігають, перетворюють та обслуговують дані за різних операційних обмежень. Data-intensive застосунки відрізняються ступенем паралелізму та потребують ефективного використання локальності даних для підвищення продуктивності [2].

Практичний спосіб оцінки таких систем полягає в описі очікуваного навантаження, наприклад частоти запитів, обсягу даних, частоти запису та розподілу навантаження, а потім – в аналізі поведінки системи зі збільшенням навантаження. Цей аналіз ґрунтується на двох взаємодоповнюючих питаннях:

1. якщо робоче навантаженння збільшується, а ресурси залишаються фіксованими (процесор, пам'ять, операції вводу/виводу сховища, мережа), як знижується продуктивність;
2. якщо робоче навантаження збільшується, наскільки необхідно масштабувати ресурси, щоб зберегти цільову продуктивність.

Показники продуктивності залежать від стилю обробки: пакетні системи часто наголошують на пропускній здатності (кількість обчислювальних записів за секудну або час виконання завдання), тоді як онлайн-сервіси, орієнтовані на користувача, наголошують на часі відгуку (затримка між запитом і відповіддю). Виконання обробки даних безпосередньо там, де вони зберігаються може істотно зменшити проблему їхнього переміщення в data-intensive застосунках [3].

Зрештою, масштабування створює якісно різні проблеми залежно від того, чи є система без збереження стану (stateless), чи зі збереженням стану (stateful). Реплікація сервісів без збереження стану на кількох машинах зазвичай є простою, але масштабування систем даних із збереженням стану з одного вузла до розподіленого розгортання додає складності (реплікація, розділення, координація, обробка збоїв та управління узгодженістю). Як результат, поширеним шляхом є збереження стану на одному вузлі, доки вартість, вимоги до пропускної здатності або вимоги до високої доступності не змусять його розподілити, і тоді рішення щодо проєктування розміщення даних та узгодженості стають центральними для загальної архітектури. На відміну від сервісів без збереження стану, які легко перезапускати на іншому вузлі, станні мікросервіси під час міграції мають зберігати стан у пам'яті, що ускладнює перенесення контейнерів між хостами, особливо у великомасштабних застосунках під керуванням платформ оркестрації на кшталт Kubernetes [4]

Формулювання цілей статті

Метою роботи є: дослідження методів і технологій опрацювання великих обсягів даних у децентралізованих системах.

Виклад основного матеріалу

Проблеми великих даних природно породжують кілька ключових дослідницьких питань [6], зокрема:

1. як проєктувати масштабовані середовища обробки;
2. як забезпечити відмовостійкість за частих відмов компонентів;
3. як створювати рішення, що залишаються ефективними з точки зору затримки, пропускної здатності та використання ресурсів.

Ці питання стають особливо важливими в розподілених середовищах, де обсяг, швидкість та різноманітність даних збільшують ймовірність виникнення вузьких місць (bottlenecks), наприклад мережевих обмежень та обмежень вводу/виводу, та ускладнюють гарантію коректності та надійності. Огляди систем великих даних постійно підкреслюють масштабованість та відмовостійкість як першокласні обмеження проєктування, поряд з продуктивністю та економічною ефективністю.

Системи опрацювання великих даних з огляду на вимоги до оперативності зазвичай поділяють на дві групи: пакетні системи обробки великих даних та системи потокової обробки. Пакетну обробку також розглядають як роботу з історичними великими даними, тоді як потокові системи призначені для обробки потоку даних у реальному часі [7].

Більше того, багато традиційних підходів до зберігання та аналітики, спочатку розроблених для помірних, структурованих робочих навантажень, часто намагаються задовольнити вимоги масштабних,

неоднорідних, швидкозмінних даних. Як результат, сучасні архітектури великих даних спираються на розподілені моделі обробки та спеціалізовані платформи, наприклад системи виконання для великих кластерів, які явно включають механізми паралелізму та стійкості. До них належать повторне виконання завдань, реплікація, контрольні точки та служби координації. Наприклад MapReduce популяризував відмовостійку модель для обробки масивних даних шляхом автоматичного повтороного запуску невдалих завдань та керування розподіленим виконанням між кластерами. Паралельно, новіші аналізи підкреслюють, що досягнення ефективної відмовостійкості саме по собі є нетривіальним, оскільки методи забезпечення надійності можуть призвести до значних накладних витрат часу, сховища та обчислювальних ресурсів, що робить розробку «ефективної відмовостійкості» активним напрямком досліджень.

Apache Kafka – це розподілена платформа потокової передачі подій [8], розроблена для забезпечення високопродуктивної, відмовостійкої та масштабованої передачі потоків даних між незалежними програмними компонентами. Вона широко застосовується в сучасних ахітектурах з інтесивним використанням даних, де системи повинні безперервно, в режимі реального часу та з мінімальною затримкою обмінюватися великими обсягами інформації. Kafka дотримується моделі зв'язку «publish-subscribe», де одні сервіси публікують події в певні теми (topic), а інші сервіси виконують роль споживачів і підписуються та читають ці події незалежно. Такий підхід дозволяє роз'єднати виробників та споживачів даних, зменшуючи прямі залежності та підвищуючи гнучкість у розподілених архітектурах.

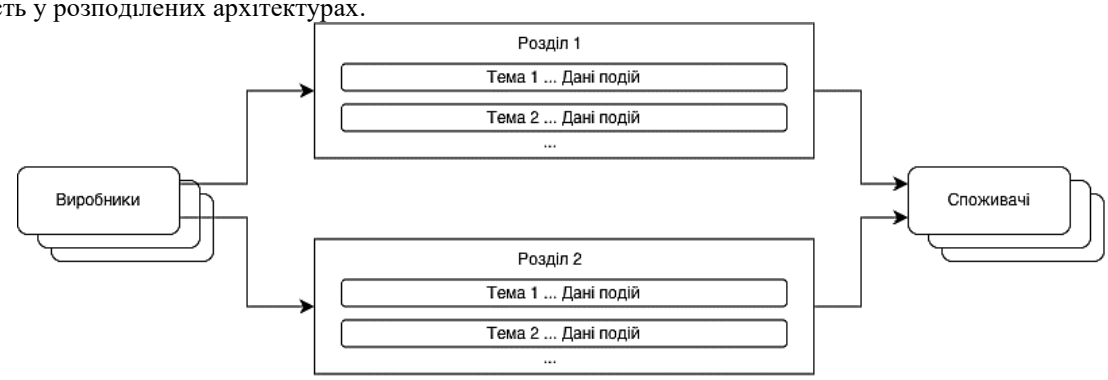

Рис. 1. Огляд архітектури Apache Kafka

Ключовою особливістю Kafka є модель зберігання на основі журналів (commit log). Події послідовно додаються до розділів тем, утворюючи незмінний журнал записів, який довго зберігається на диску. Теми поділяються на розділи, що забезпечує горизонтальну масштабованість та паралельну обробку. Упорядкування подій гарантується всередині окремого розділу, тоді як розділення дозволяє Kafka обробляти потоки розподілено між кількома брокерами (вузлами кластера). Крім того, Kafka підтримує реплікацію розділів між брокерами, що значно підвищує стійкість до збоїв вузлів та забезпечує високу доступність потокових даних. Споживачі відстежують свій прогрес за допомогою зміщень (offsets), що дозволяє відновлювати обробку після збоїв, відтворювати історичні дані або виконувати повторону обробк.

Apache Kafka часто використовується як основний компонент інфраструктури в децентралізованих системах великих даних, де джерела даних та вузли обробки розподілені між кількома сервісами, командами або локаціями. У таких середовищах Kafka діє як транспортний та буферний рівень, що забезпечує безперервне отримання та поширення подій від багатьох постачальників, зокрема мікросервісів, пристроїв IoT, користувацьких програм, систем моніторингу та транзакційних платформ. Оскільки зв'язок є асинхронним, Kafka зменшує тісний зв'язок між компонентами, підтримує стійку роботу за умов різких навантажень та мережевих затримок, а також покращує загальну стійкість системи, дозволяючи сервісам споживати дані незалежно, коли вони готові.

Apache Hadoop – це фреймворк, розроблений для розподіленог зберігання та обробки дуже великих наборів даних на кластерах стандартного обладнання [9]. Він дозволяє масштабоване керування даними та обчислення, розподіляючи як дані, так і робочі навантаження між кількома вузлами, що є важливим у середовищах великих даних, де централізовані системи стають неефективними або занадто дорогими. Архітектура Hadoop побудована навколо розподіленої файлової системи Hadoop (HDFS) для відмовостійкого зберігання та рівня керування ресурсами кластера (YARN) для планування та розподілу обчислювальних ресурсів. Великі набори даних зберігаються у вигляді блоків, реплікованих між вузлами, що дозволяє системі залишитися доступною навіть у разі збоїв.

Фунтаментальною моделлю обробки в екосистемі Hadoop є **MapReduce**, яка забезпечує пакетно-орієнтований підхід до паралельних обчислень. MapReduce поділяє обчислення на дві основні фази: крок map перетворює та фільтрує вхідні дані в проміжні пари ключ-значення, тоді як крок reduce агрегує та підсумовує ці проміжні результати для отримання кінцевих результатів [10]. Ця модель підтримує розподілене виконання в масштабі та особливо ефективна для великих завдань офлайн-аналітики, таких як обробка журналів, індексація, підрахунок та ETL (Extract, Trasnform, Load). У децентралізованих архітектурах великих даних Hadoop зазвичай використовується як основа озера даних (data lake), де різноманітні джерела даних консолідуються для довгострокового зберігання та масштабного історичного аналізу. У поєднанні з такими інструментами, як Hive для SQL-подібних запитів та HBase для розподіленого NoSQL-сховища, Hadoop підтримує надійні конвеєри

обробки та дозволяє децентралізованим командам і службам працювати зі спільними наборами даних, зберігаючи розподілене виконання, надійність і масштабованість.

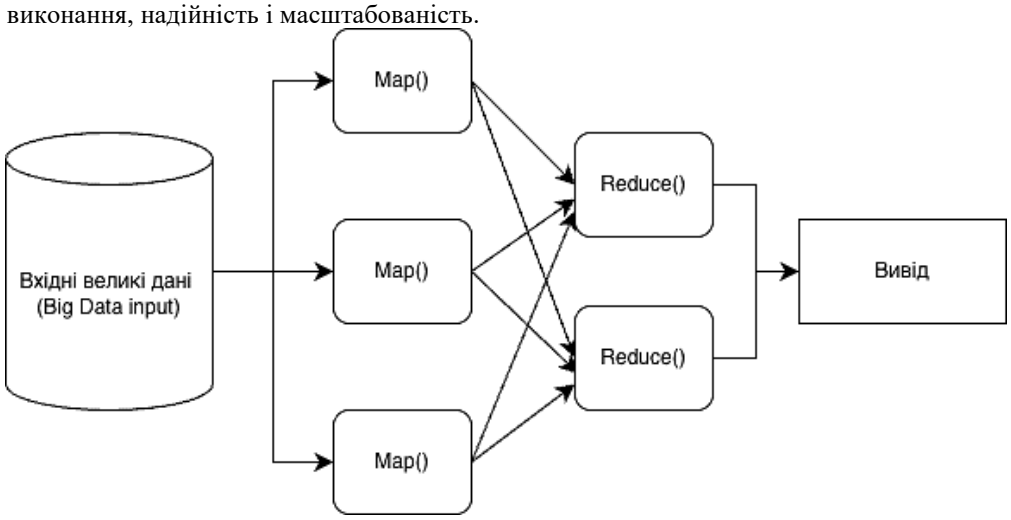

Рис. 2. Огляд MapReduce

Фреймворк Apache Spark застовується для розподілених обчислень, розроблений для швидкої та масштабованої обробки великих наборів даних. Він широко використовується в середовищах великих даних, оскільки підтримує як пакетну аналітику, так і потокову обробку в режимі реального часу в рамках єдиної архітектури [11]. Spark виконує робочі навантаження паралельно в кластері, використовуючи модель master-worker, та менеджер кластера, такий як YARN, Kubernetes, або вбудований автономний режим. Ключовою перевагою Spark є його можливість обробки в пам'яті, яка значно прискорює ітеративні обчислення та повторний доступ до даних порівняно з дисково-орієнтованими підходами, що робить його ефективним для складної аналітики, масштабних перетворень та конвеєрів машинного навчання.

Apache Spark надає структуровану модель програмування через високорівневі API, такі як DataFrames та Datasets, що дозволяє розробникам писати розподілену логіку обробки даних за допомогою SQL-подібних операцій, тоді як фреймворк автоматично оптимізує виконання за допомогою свого оптимізатора запитів. Для випадків потокового використання Spark Structured Streaming дозволяє обробляти дані, що безперервно надходять, майже в реальному часі, часто інтегруючись з такими системами, як Apache Kafka, для прийому даних. У децентралізованих архітектурах великих даних фреймворк зазвичай застосовується як розподілений шар обробки, який об'єднує дані з кількох джерел і виконує перетворення, агрегації, виявлення аномалій та розробку функцій у кластерах. Його підтримка бібліотек, таких як Spark SQL, MLlib для машинного навчання та GraphX для графової аналітики, дозволяє організаціям створювати комплексні аналітичні рішення, де окремі служби та команди можуть обробляти спільні набори даних, зберігаючи масштабованість, відмовостійкість та ефективне використання ресурсів у розподілених середовищах.

Apache Flink – це розподілена платформа для обробки потоків з відкритим кодом, розроблена для високопродуктивної аналітики з низькою затримкою даних, що постійно надходять [12]. На відміну від традиційних механізмів пакетної обробки, Flink побудований з архітектурою потокової обробки, де як робочі навантаження в реальному часі, так і пакетні, преставлені у вигляді потоків даних. Це робить Flink добре пристосованим для сучасних середовищ великих даних, які потребують негайної реакції на події, такі як виявлення шахрайства, моніторинг, персоналізація та аналітика промислової телеметрії. Flink працює на розподілених кластерах та інтегрується з менеджерами ресурсів, такими як Kubernetes та YARN, забезпечуючи масштабоване виконання на кількох вузлах з ефективним паралелізмом.

Основною можливість Flink є підтримка обробки потоків з урахуванням стану, де система зберігає результати з часом (стан) та оновлює їх у міру надходження нових подій. Це дозволяє виконувати розширені обчислення, такі як агрегації на основі вікон, виявлення шаблонів подій та аналіз сеансів. Flink також підтримує оброку в часі подій, що дозволяє отримувати правильні результати навіть тоді, коли дані надходять із затримкою або не в порядку – поширена проблема в децентралізованих системах, де події можуть затримуватись мережами або створюватись географічно розподіленими джерелами. Для надійності Flink забезпечує механізми контрольних точок та відновлення стану, які допомагають гарантувати послідовну обробку у разі збоїв. У децентралізованих архітектурах великих даних Flink часто використовується разом з Apache Kafka як основа для прийому та обміну повідомленнями, тоді як Flink виконує перетворення, збагачення та виявлення аномалій у реальному часі перед записом результатів в аналітичні бази даних, озера даних або мікросервіси нижче за течією. Apache Flink має спільні риси з Spark, він пропонує хорошу продуктивність обробки складних структур великих даних, таких як графи.

Висновки з даного дослідження
і перспективи подальших розвідок у даному напрямі

Виконаний огляд дозволяє зробити висновок, що для сучасних систем, що вимагають обробки великої кількості даних ключовими є інженерні компроміси між масштабованістю, узгодженістю, надійністю,

ефективністю та зручністю супроводу за умов росту обсягів, гетерогенності та частоти оновлення даних. Визначено, що більшість систем опрацювання великих даних будуються з повторюваних структурних блоків, зокрема сховища, кешування, обміну повідомленнями, потокової обробки та пакетної обробки. Ці блоки об'єднуються в наскрізні конвеєри приймання, зберігання, перетворення та надання даних.

Узагальнено практичний підхід до оцінювання продуктивності, коли спершу описується очікуване навантаження, далі аналізується деградація продуктивності при фіксованих ресурсах і потреба в масштабуванні ресурсів для збереження цільових показників. Обґрунтовано, що для децентралізованих архітектур визначальним є горизонтальний розподіл навантаження в межах шару (шардинг/розділення даних), який забезпечує паралелізм і балансування, тоді як вертикальний розподіл відповідає функціональній декомпозиції.

На прикладі оглянутих технологій показано їхні ролі в сучасних конвеєрах даних. Kafka розглянуто як подієвий транспорт і буфер із розділами, реплікацією та зміщеннями для відновлення і повторної обробки. Hadoop і MapReduce подано як основу для довготривалого зберігання та пакетної обробки. Spark описано як універсальний обчислювальний шар для пакетної й потокової аналітики з акцентом на in-memory обробку. Flink представлено як платформу для stateful stream processing з event time та контрольними точками, яку часто застосовують у зв'язці з Kafka для задач реального часу.

Література

1. Kleppmann M. *Designing data-intensive applications: the big ideas behind reliable, scalable, and maintainable systems*. Sebastopol, CA: O'Reilly Media, 2017. 616 p.

2. Stavrinides G. L., Karatza H. D. Scheduling data-intensive workloads in large-scale distributed systems: trends and challenges // arXiv preprint. 2025. arXiv:2510.25362. DOI: 10.48550/arXiv.2510.25362.

3. Singh G. Designing, modeling, and optimizing data-intensive computing systems // arXiv preprint. 2022. arXiv:2208.08886. DOI: 10.48550/arXiv.2208.08886.

4. Dinh-Tuan H., Jiang J. Optimizing stateful microservice migration in Kubernetes with MS2M and forensic checkpointing // arXiv preprint. 2025. arXiv:2509.05794. DOI: 10.48550/arXiv.2509.05794.

5. van Steen M., Tanenbaum A. S. *Distributed systems*. 3rd ed. [S. l.]: distributed-systems.net, 2017. 623 p.

6. Chen M., Mao S., Liu Y. Big data: a survey // *Mobile Networks and Applications*. 2014. Vol. 19, No. 2. P. 171–209. DOI: 10.1007/s11036-013-0489-0.

7. Zheng T., Chen G., Wang X., Chen C., Wang X., Luo S. Real-time intelligent big data processing: technology, platform, and applications // arXiv preprint. 2021. arXiv:2111.11872. DOI: 10.48550/arXiv.2111.11872.

8. Zelenin A., Kropp A. *Apache Kafka in action: from basics to production*. 1st ed. Shelter Island, NY: Manning Publications Co., 2025. 384 p.

9. White T. *Hadoop: the definitive guide: storage and analysis at internet scale*. 4th ed. Sebastopol, CA: O'Reilly Media, 2015. 756 p.

10. Dean J., Ghemawat S. MapReduce: simplified data processing on large clusters // *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI'04)*. 2004. P. 137–150.

11. Armbrust M. et al. Structured streaming: a declarative API for real-time applications in Apache Spark // *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. 2018. P. 601–613.

12. Carbone P., Katsifodimos A., Ewen S., Markl V., Haridi S., Tzoumas K. Apache Flink™: stream and batch processing in a single engine // *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*. 2015. Vol. 36, No. 4. P. 28–38.

References

1. Kleppmann M. *Designing data-intensive applications: the big ideas behind reliable, scalable, and maintainable systems*. Sebastopol, CA: O'Reilly Media, 2017. 616 p.

2. Stavrinides G. L., Karatza H. D. Scheduling data-intensive workloads in large-scale distributed systems: trends and challenges // arXiv preprint. 2025. arXiv:2510.25362. DOI: 10.48550/arXiv.2510.25362.

3. Singh G. Designing, modeling, and optimizing data-intensive computing systems // arXiv preprint. 2022. arXiv:2208.08886. DOI: 10.48550/arXiv.2208.08886.

4. Dinh-Tuan H., Jiang J. Optimizing stateful microservice migration in Kubernetes with MS2M and forensic checkpointing // arXiv preprint. 2025. arXiv:2509.05794. DOI: 10.48550/arXiv.2509.05794.

5. van Steen M., Tanenbaum A. S. *Distributed systems*. 3rd ed. [S. l.]: distributed-systems.net, 2017. 623 p.

6. Chen M., Mao S., Liu Y. Big data: a survey // *Mobile Networks and Applications*. 2014. Vol. 19, No. 2. P. 171–209. DOI: 10.1007/s11036-013-0489-0.

7. Zheng T., Chen G., Wang X., Chen C., Wang X., Luo S. Real-time intelligent big data processing: technology, platform, and applications // arXiv preprint. 2021. arXiv:2111.11872. DOI: 10.48550/arXiv.2111.11872.

8. Zelenin A., Kropp A. *Apache Kafka in action: from basics to production*. 1st ed. Shelter Island, NY: Manning Publications Co., 2025. 384 p.

9. White T. *Hadoop: the definitive guide: storage and analysis at internet scale*. 4th ed. Sebastopol, CA: O'Reilly Media, 2015. 756 p.

10. Dean J., Ghemawat S. MapReduce: simplified data processing on large clusters // *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI'04)*. 2004. P. 137–150.

11. Armbrust M. et al. Structured streaming: a declarative API for real-time applications in Apache Spark // *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. 2018. P. 601–613.

12. Carbone P., Katsifodimos A., Ewen S., Markl V., Haridi S., Tzoumas K. Apache Flink™: stream and batch processing in a single engine // *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*. 2015. Vol. 36, No. 4. P. 28–38.