УДК 004.75
DOI https://doi.org/10.32782/2663-5941/2025.1.2/05

Вакалюк Т.А.
Державний університет «Житомирська політехніка»

Лобанчикова Н.М.
Державний університет «Житомирська політехніка»

Янчук В.М.
Державний університет «Житомирська політехніка»

Фаррахов О.В.
Центр інформаційно-аналітичного та технічного забезпечення моніторингу об'єктів атомної енергетики Національної академії наук України

Жиляєв Є.В.
Державний університет «Житомирська політехніка»

ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ ПРОТОТИПУ ДОДАТКУ ВИЯВЛЕННЯ АНОМАЛІЙ ШЛЯХОМ ВІДСТЕЖЕННЯ ПОДІЙ У РОЗПОДІЛЕНИХ КОМП'ЮТЕРНИХ СИСТЕМАХ

В статті представлено теоретичне обґрунтування та розробку прототипу додатку для виявлення аномалій у розподілених комп'ютерних системах шляхом відстеження подій. Досліджено сучасні підходи до моніторингу та аналізу аномалій у складних розподілених системах, зокрема з використанням технологій Spring Cloud Sleuth, Apache Kafka та Apache Spark. Розроблено архітектуру прототипу, що базується на модульному підході та забезпечує гнучкість і масштабованість рішення. Визначено та систематизовано ключові метрики для моніторингу, які включають ідентифікаційні метрики (Span ID, Trace ID, назва сервісу), метрики швидкодії (час відгуку, завантаження ЦПУ та пам'яті) та метрики помилок (http-статус, теги помилок). Особливу увагу приділено вибору та обґрунтуванню алгоритмів виявлення аномалій. На основі аналізу існуючих підходів обрано алгоритми на базі кластеризації, зокрема KMeans, що забезпечують високу швидкодію та ефективне використання пам'яті при обробці даних у режимі реального часу. Визначено та охарактеризовано три основні типи аномалій для моніторингу: помилки та винятки, порушення визначених порогових значень та збільшення часу відгуку запитів. Запропоновано інноваційний підхід до аналізу першопричин аномалій з урахуванням архітектурного контексту, що дозволяє не лише виявляти аномалії, але й визначати їх джерело в складній розподіленій системі. Розроблено механізм обробки даних у режимі реального часу з використанням Apache Spark, що забезпечує швидку та ефективну обробку великих обсягів даних від різних сервісів. Особливістю запропонованого рішення є можливість паралельного використання різних алгоритмів виявлення аномалій та їх легкої заміни або модифікації завдяки модульній архітектурі. Теоретичні засади створюють підґрунтя для практичної реалізації ефективної системи виявлення аномалій у розподілених комп'ютерних системах. Запропоноване рішення може бути використане для підвищення надійності та ефективності роботи складних розподілених систем.

Ключові слова: *розподілені комп'ютерні системи, виявлення аномалій, відстеження подій, Spring Cloud Sleuth, Apache Kafka, Apache Spark, кластеризація, KMeans, аналіз першопричин, моніторинг у реальному часі.*

Постановка проблеми. Сучасні розподілені комп'ютерні системи стають все більш складними та масштабними, що ускладнює процес моніторингу їх стану та виявлення потенційних проблем. Особливо гостро постає питання швидкого виявлення аномалій у роботі таких систем, оскільки навіть незначні збої можуть призвести до суттєвих втрат як з технічної, так і з бізнес точки зору. Традиційні методи моніторингу часто виявляються неефективними через велику кількість даних, які потрібно аналізувати в режимі реального часу, та складність взаємозв'язків між різними компонентами системи. Особливо це стосується мікросервісної архітектури, де кожен сервіс може мати

власні особливості роботи та патерни поведінки. Крім того, існуючі рішення часто не враховують контекст архітектури системи при виявленні аномалій, що може призводити до неправильної інтерпретації проблем та ускладнювати процес їх усунення. Тому розробка ефективного підходу до виявлення аномалій шляхом відстеження подій у розподілених комп'ютерних системах з урахуванням архітектурного контексту є актуальною науково-практичною задачею.

Аналіз останніх досліджень і публікацій. Питанням виявлення аномалій у розподілених комп'ютерних системах присвячено значну кількість наукових праць. Зокрема, Бовензі (Bovenzi) та ін. [1] запропонували статистичний алгоритм для виявлення аномалій в режимі онлайн для критично важливих програмних систем, який враховує динамічну природу таких систем. Пітакрат (Pitakrat) та ін. [2] розробили підхід до ієрархічного прогнозування відмов з урахуванням архітектури, що дозволяє більш точно визначати причини аномалій у складних системах.

Важливий внесок у дослідження методів виявлення аномалій з використанням сучасних технологій зробили Солаймані (Solaimani) та ін. [3], які запропонували статистичний метод онлайн-виявлення аномалій з використанням Apache Spark для обробки гетерогенних даних від різних джерел. Їхня робота демонструє ефективність використання технологій розподіленої обробки даних для задач моніторингу.

Значний прогрес у розробці інфраструктури для відстеження подій у розподілених системах представлено в роботі Сіджелман (Sigelman) та ін. [5], де описано систему Dapper, яка стала основою для багатьох сучасних рішень, включаючи Spring Cloud Sleuth [4]. (Захарія) Zaharia [10] у своєму дисертаційному дослідженні представив архітектуру для швидкої та загальної обробки даних у великих кластерах, що лягла в основу Apache Spark [6]. Подальший розвиток цієї технології, включаючи MLlib [7] та Spark Streaming [8], створив потужну екосистему для обробки даних у режимі реального часу та машинного навчання.

Аналіз досліджень показує, що незважаючи на значний прогрес у розробці методів та інструментів для виявлення аномалій, залишається актуальною проблема створення комплексного рішення, яке б поєднувало ефективні алгоритми виявлення аномалій з урахуванням архітектурного контексту системи та можливістю роботи в режимі реального часу.

Постановка завдання. Метою даної роботи є опис теоретичних засад розробки прототипу додатку для виявлення аномалій шляхом відстеження подій у розподілених комп'ютерних системах з урахуванням архітектурного контексту, що забезпечує моніторинг та аналіз у режимі реального часу.

Виклад основного матеріалу. Мета полягає у виявлені аномалій на основі даних від системи відстеження подій. Більше того, виявлені аномалії мають розглядатись в контексті архітектури, щоб визначити, які з аномалій надходять від сервісу, що їх викликав, а які є лише наслідком поширення помилки від іншого сервісу. На першому етапі дані отримуються за допомогою системи відстеження подій, після чого дані готуються відповідно до потреб алгоритмів виявлення аномалій. Ці алгоритми виявлять аномалії серед вхідних даних, як тільки вони стають доступними, тобто обробляються дані передаються та обробляються в режимі онлайн. Згодом, виявлені аномалії додатково опрацьовуються за допомогою з точки зору аналізу першопричини, щоб пріоритезувати ті аномалії, що найімовірніше є першопричиною.

Рішення-прототип має охоплювати усі ці етапи. Однак, в даній імплементації фокус буде зосереджений на підготовці даних для обробки, виявлені аномалій та їх аналізу в контексті залежностей сервісу задля визначення першопричини аномалії. Саме вилучення даних із сервісу та візуалізація результатів будуть реалізовані на мінімально-базовому рівні.

Також, по своїй суті прототип має передбачатися як основа для подальших досліджень та/або покращень, а тому має враховувати можливості для подальших змін з мінімальними зусиллями. Для цього рішення має бути реалізовано через модульний підхід.

Перш за все, моделі та алгоритми для виявлення аномалій мають бути взаємозамінними. Навіть використання різних алгоритмів паралельно з прийнятними результатами має бути можливим. Для досягнення цього, інфраструктурою, яка буде зв'язувати різні частини рішення буде реалізовано за допомогою асинхронною системи повідомлень. Якщо при паралельній роботі алгоритмів, вони відмітять один і той самий запит як аномальний, аналіз першопричини має бути здатним виявляти та об'єднувати такі дублюючи результати. Також, сам процес обробки має бути окремим модулем, для того щоб вибір різного інструментарії в якості системи відстеження подій, або ж вибір іншого підходу для збору інформації про залежності для

аналізу першопричину не впливало на роботу інших частин рішення. Єдиним має бути лише формат повідомлення у якому частини прототипу будуть комунікувати між собою, самій ж модулі можна буде заміняти.

Інструментарій рішення для відстеження подій створено на основі Spring Cloud Sleuth. Базові налаштування буде розширено за допомогою додаткового коду, щоб отримати до додаткової інформації, що може бути корисна для виявлення аномалій.

Дані від системи відстеження подій будуть записуватися у відповідний розділ системи Kafka у форматі JSON. Звідти їх буде отримавати модель для виявлення аномалій, реалізований на основі Apache Spark. Після отримання даних із Kafka дані будуть аналізуватися за допомогою декількох різних алгоритмів для виявлення аномалій, кожен з яких є незалежним від інших.

Алгоритм може мати, а може і не мати тренувальну фазу, під час якої буде мати доступ до усіх історичних даних у Kafka. Кожен алгоритм має фазу виявлення, коли він аналізує точку даних та оцінює чи є вона аномальною чи ні. Після виявлення аномалій, кожен з алгоритмів записує дані про них в загальний для всіх алгоритмів розділ Kafka, внаслідок чого може виникнути дубльовані дані.

Другий модуль, також реалізований на основі Apache Spark, відповідає за аналіз першопричини. Він зчитує дані про аномальні точки даних із Kafka та після їх аналізу має надати першопричину для певного набору аномалій. Це робиться через аналіз аномалії у контексті залежностей сервісу де вона виникла.

Нарешті, прототип також матиме просту візуалізацію для демонстрації результатів в зрозумілий для людини спосіб. Візуазізація буде реалізована у базовому вигляді.

Spring Cloud Sleuth [4] є фреймворком для реалізації відстеження подій у розподілених системах в контексті Spring Cloud Applications. Однією з «гарантій» цього проєкту є отмання даних про події без жодної постобробки та якомога швидше, а тому це є хорошим вибором для реалізації в якій одна з цілей є виявлення аномалій в режимі онлайн.

Spring Cloud Sleuth використовує ту саму термінологію, що і робота по системі Dapper [5]. Використовуються такі терміни як «трейс (trace = слід)» та «спен (span = діапазон)». Єдиною складністю тут є те що частина інформації про спен події записується на одному сервісі, а інша частина на іншому. Рішенням цієї проблеми є те, що усі сервіси будуть збирати інформацію про події за певний проміжок часу, а потім надсилати їх до системи обміну повідомленнями в форматі JSON. Такий об'єкт буде містити інформацію як про сервіс, що його надіслав, так і про спен подій за сторони саме цього сервісу. Однак, іншим викликом є те, що оскільки інформація про події буде посилатися як з сервісу що надсилає запит, так і з сервісу, що обробляє запит, необхідно буде об'єднати ці об'єкти.

Apache Spark [6] – це об'єднана структура для обробки даних у розподілених комп'ютерних

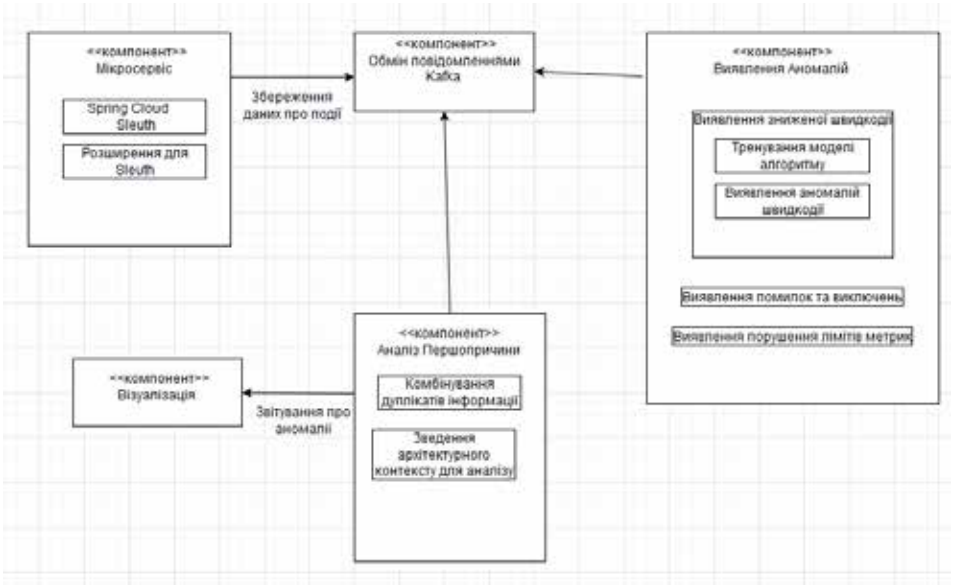

Рис. 1. Загальний огляд архітектури

системах. Проект розпочався у 2009 році в Каліфорнійському університеті в Берклі. Відтоді він став найбільш активним проектом з відкритим кодом, нараховуючи понад 1000 учасників та використовується у більше ніж 1000 компаніях. Також, він використовується в багатьох поточних академічних дослідницьких проектах які досліджують потоки даних та виявлення аномалій, наприклад [3].

Apache Spark підтримує кілька мов програмування, таких як Java, Scala та Python. Головною абстракцією даних у цьому фреймворку є RDD (Resilient Distributed Datasets = Стійкі Набори Даних у розподіленій системі). Це колекції об'єктів, які можуть бути розподілені між кількома кластерами даних для швидкої паралельної обробки завдань. Основні операції для обробки даних що підтримуються є map, filter та group by.

Для розширення базового функціоналу Apache Spark, яка в основному націлена на обробку великих наборів даних, різні додаткові бібліотеки мають бути включені до проекту. Spark SQL дозволяє користувачеві використовувати SQL-запити відносно RDD, як один із методів структурування даних. Spark MLlib [7] містить алгоритми машинного навчання, які використовують фреймморк Spark для швидкої обробки та можуть бути легко інтегровані в програми на основі Spark. Останньою бібліотекою для розширення можливостей фрейморку є Spark Streaming [8]. Вона забезпечує розподілену обробку та маніпуляцію потоків даних.

Не зважаючи на те що Apache Spark за своїм дизайном більше розрахований для роботи з великими об'ємами даних, він також може надавати швидкодію у нішевих сферах, таких як використання потоків та машинне навчання. Захарія (Zaharia) [10] порівняв Apache Spark з Apache Storm [9], іншим проектом з відкритим кодом, що спеціалізується на роботі з потоками. Згідно його результатів, Apache Spark показав щонайменше вдвічі більшу швидкодію при обробці завдань.

Отже, з готовою основою для роботи з потоками даних да імплементованими алгоритмами машинного навчання, Apache Spark має в своєму розпорядженні увесь інструментарій необхідний для цієї практичної роботи, а саме для виявлення аномалій у вхідному потоці даних. Також, важливо відзначити, що цей фреймворк має у своєму розпорядженні навіть більше додаткових інструментів, в тому числі для роботи з Big Data, що дозволить легше адаптувати рішення до можливих нових викликів та вимог у майбутньому.

Основною причиною вибору Apache Kafka [11] є обмеження, що випливають із вибору Apache Spark та Spring Cloud Sleuth. Spring Cloud Sleuth підтримує вивід даних до Apache Kafka або RabbitMQ [13]. З іншого боку Spark Streaming надає вбудовану підтримку зчитування даних з Apache Kafka [12], в той час як підтримка RabbitMQ наявна лише через сторонні бібліотеки. Крім того, Apache Kafka пропонує можливість зберігати дані на додаток до функціоналу в якості серві для обміну повідомленнями. Навіть у блозі компанії Pivotal, автору фреймворку Spring та команда підтримки RabbitMQ, Apache Kafka згадується як хороший вибір для сценаріїв де потребується «потік від сервісу A до B без складної маршрутизації та з максимальною пропускною здатністю», що є як раз сценарієм для рішення в рамках цієї роботи. Саме ці фактори роблять Apache Kafka підходящим вибором для асинхронної системи обміну повідомленнями.

Як було описано в архітектурі, в рішенні-прототипі для збору метрики планується використовувати Spring Cloud Sleuth. Єдина метрика, яка збирається за замовчення, це час відгуку, а точніше, тривалість запиту.

За допомогою додаткового налаштування фреймворку можна також отримати середнє значення Завантаження ЦП та Завантаження пам'яті у JVM (віртуальна машина для Java). Ця метрика збирається за замовченням у JVM- платформі і потрібно лише передати їх до Sleuth. Такі налаштування базуються на можливостях моніторингу зсередини JVM і від розробника вимагається лише додати кілька рядків коду в конфігурації мікросервісу. Нижчу показано все, що потрібно зробити для кожного сервісу в класі конфігурації. Після ін'єкції BeanFactory та Spring Cloud Sleuth Tracer потрібно додатково зареєструвати три додаткові компоненти. Ці компоненти імпортуються в якості додаткових залежностей.

На додаток до того, що згадується у літературі, Spring Cloud Sleuth також надає додаткові можливості, які варто розглянути, адже серед них є метрики які можуть допомогти прослідити конкретний виклик методу на конкретному сервісі. До них входить інформація про URI (unique resource identifier = унікальний ідентифікатор ресурсу) запиту, назва сервісу, IP-адреса та порт.

Щодо http-запитів Spring Cloud Sleuth надає моржливість відслідковувати http-метод та http-статус. Більш того, у випадку помилки, до метрик додається тег помилки. Ну і нарешті, надаються ідентифікаційні номера, які дозволяють прослід-

ковувати конкретний виклик і те як він проходить через різні сервіси, вони називаються Span ID(span = діапазон) та Trace ID(trace = слід).

Отже, в залежності від алгоритмів та потреб, планується отримати наступні метрики, що наведені у таблиці 1.

Прототип за своїм дизайном має виявляти наступні три типи аномалій: помилки та винятки; порушення визначених порогових значень (наприклад, використання ресурсів ЦП або ресурсів пам'яті системи); збільшення часу відгуку запитів порівняно з визначеним «нормальним» рівнем.

Помилка – це відхилення стану сервісу від очікуваного. Зазвичай це призводить до виключення або повідомлення з кодом помилки. В ідеальному світі, програма має бути протестована, усі можливі причини помилок виявлені і усунуті.

Визначені порогові значення особливо корисні для моніторингу метрик, які вимірюється у відсотках. З уже визначених метрик, виведених у таблиці 1, такими є завантаження ресурсів ЦП та завантаження ресурсів пам'яті.

Третій тип аномалій це збільшений час відгуку, порівняно з «нормальним» очікуваним рівнем. Це є важливим показником, адже збільшений час відгуку може негативно вплинути на успіх бізнесу, що підтримує веб- додаток. Тому необхідно встановити базове очікуване значення для часу відгуку. Аномаліями будуть вважатись усі запити, відповідь на які буде потребувати більше часу, ніж задане порогове значення.

З виявленням таких типів аномалій, можна виявити наступні сценарії збоїв системи: сервіс аварійно завершує роботу через помилку під час виконання; сервіс не повертає відповідь на відгук вчасно; сервіс має збільшений час відгуку порівняно з очікуваним, що означає недостатню швидкодію сервісу.

Перший сценарій спричинює помилку, або ж виняток, тобто одну із визначених вище аномалій. Виявляючи їх, цей сценарій вирішується.

В системі під спостереженням, запит, який потребує занадто багато часу, призведе до помилки в зв'язку з таймаутом. Отже, цей сцена-

```
@Configuration
public class MvcConfig extends WebMvcConfigurerAdapter {

    Tracer tracer;

    @Autowired
    BeanFactory beanFactory;

    Tracer tracer() {
        if (this.tracer == null) {
            this.tracer = this.beanFactory.getBean(Tracer.class);
        }
        return this.tracer;
    }

    @Bean
    public DemoTraceHandlerInterceptor demoTraceHandlerInterceptor(BeanFactory beanFactory) {
        return new DemoTraceHandlerInterceptor(beanFactory, tracer());
    }

    @Bean
    public SpanAdjuster demoSpanAdjuster() {
        return new demoSpanAdjuster();
    }

    @Bean
    DemoFilter demoFilter() {
        return new DemoFilter(tracer());
    }
}
```

Рис. 2. Код, необхідний сервісу для більшої якості метрик

Таблиця 1

Метрики для використання у прототипі

Ідентифікаційні метрики	Метрики швидкодії	Метрики помилок
Span ID Trace ID Назва сервісу URL запиту IP-адреса Порт http-метод	Час відгуку Завантаження ЦПУ Завантаження пам'яті	http-статус Тег помилки

рій збою системи також охоплюється виявленням конкретним помилок. Якщо ж інша система знає як справитись з таймаутом, повертаючи запит із дуже довгою тривалістю, це охоплюється виявленням збільшення часу відгуку системи. Єдиний випадок, який не може виявлятися системою, це коли нічого не повідомляється. Якщо немає звітованого спену, система не зможе нічого виявити.

Щоб проілюструвати корисність спостереження за визначеними пороговими значенням, буде використаний сценарій витоку пам'яті. У цьому сценарії завантаження ЦПУ збільшується, коли використання ресурсів пам'яті JVM перевищує певний поріг. Це пов'язано з більшою активністю системи garbage collector (збирача сміття), яка звільнює пам'ять від об'єктів, що не використовуються. Системний адміністратор може встановити порогове значення і бути проінформованим, як тільки показник увійде у критичну відмітку для певного сервісу.

Моніторинг мікросервісу для виявлення аномалій створює необхідність визначення певних вимог. По-перше, завдання моніторингу мікросервісу в режимі онлайн потребує здійснення цього процесу в реальному часі [1]. Більш того, має бути враховано, що якщо система піднімає занадто багато помилкових тривог, то вона, ймовірно, буде ігноруватися, або вимкнена оператором. А тому, надійний алгоритм виявлення, який також враховує можливість моніторингу у реальному часі є базовою вимогою.

Також, у програмному середовищі, яке часто змінюється, як мікросервісна програма, що знаходиться у стадії активної розробки, необхідно щоб і модель виявлення аномалій могла часто адаптуватися до нових вимог [2]. Коли стан системи під час роботи починає сильно відхилятися від стану систему, на якому була натренована модель виявлення аномалій, точність виявлення аномалій неодмінно знижується [1]. Тому, додатковою важливою вимогою є застосування алгоритмів які здатні справлятися зі змінним середовищем.

Відстеження помилок і таймаутів не потребує алгоритмів пов'язаних з машинним навчанням чи статистикою, принаймні у наведеному прикладі використання. Коли виникає помилка, Spring Cloud Sleuth додає тег помилки до поточного спену. Крім того, спени http-запитів отримують тег з http-статусом, що повертається. Таким чином, фільтрація спенів http-запитів з тегами помилки або ж спени запитів з http-статусом 500 (Внутрішня помилка серверу) в результаті надасть потік спенів, де запити або в результатів отримали таймаут, або ж викликали виключення, яке ніяк не обробляється, в програмі. Це можна порівняти з підходом на основі закономірностей, однак ніякого тренування алгоритму не відбувається. Буде виявлятися лише закономірність, що буде визначена як аномальна.

При спостереженні аномалій швидкодії є три метрики, такі як завантаження ресурсів пам'яті JVM, завантаження ресурсів ЦПУ та тривалість запиту. Щоб якнайкраще використати доступну інформацію, ці показники можна обробляти різними по-різному.

Завантаження ресурсів пам'яті JVM це є метрика, що може використовуватися як ранній індикатор того, що щось може піти не так. Оскільки процес прибирання зайвих об'єктів проходить все частіше, ресурсів пам'яті залишається все менше, може бути корисно налаштувати порогове значення використання ресурсів пам'яті та пов'язане з цим повідомлення.

Оскільки завантаження ЦПУ вимірюється у відсотках, найлегший спосіб виявляти аномалії через визначення порогового значення, вихід за яке розцінюється як аномалія.

Порівняно з завантаженням ресурсів пам'яті та ресурсів ЦПУ, порогове значення для тривалості відгуку сервісу визначити набагато складніше. Оскільки очікувана тривалість відгуку може сильно відрізнятись, залежно від конкретного використання та функції, визначення порогового значення завчасно не є варіантом. Необхідно вивести деяке базове значення, та прийнятне порогове значення виходячи із того, що є очікуваним сценарієм роботи сервісу.

Оскільки система прототипу спрямована на виявлення аномалій в режимі онлайн, надзвичайно важливо, щоб алгоритми були здатні дуже швидко обробляти вхідні дані, щоб своєчасно надавати результати та мати можливість обробляти велику кількість таких даних. Крім того, слід враховувати, що кожна точка даних поводиться по-різному. Тому цілком ймовірно, що для виявлення аномалій може знадобитися зберігати натреновану модель алгоритму для кожного сервісу окремо. Це вимагає, щоб алгоритм зберігав інформацію, необхідну для кожного сервісу якомога компактніше.

Система розробляється таким чином, що декілька різних алгоритмів можуть виконуватись паралельно. Це означає, що нові алгоритми можуть бути розміщені паралельно із початково реалізованими алгоритмами або ж повністю замінити їх. Оскільки система розроблена задля

подальшого розвитку, обрані алгоритми не мають бути досконалими, але мають бути здатні ефективно виконувати завдання з виявлення збільшення часу відгуку сервісу. Крім того, алгоритм має бути здатним виявляти аутлаєрів. Оскільки аналіз першопричини базується на припущенні, що аномалії передбачені на рівні спену. Таким чином, неможливо використовувати алгоритми, які виявляються аномальні часові ряди, оскільки тоді буде втрачена інформації на рівні спену, яка необхідна для аналізу першопричини.

В контексті інструментарію для реалізації рішення прототипу, який буде описаний в більших деталях у наступному розділі, важливо щоб обраний алгоритм мав підтримку бібліотеки Spark MLlib. Алгоритми на основі щільності можуть бути викреслені, оскільки вони не мають достатньої швидкодії при прогнозуванні аномалій. Те саме можна сказати про алгоритмі на основі дистанції між точками даних.

Всі ці вимоги залишають вибір між алгоритмами на основі моделювання часових проміжків та алгоритмами на основі кластеризації. Перша категорія не має підтримки бібліотеки Spark MLlib. Однак, щодо другої категорії, MLlib містить реалізацію алгоритму KMeans, який має додаткові сильні сторони, такі як висока швидкодія виявлення аномалій, та малий розмір даних, що мають зберігатися на сервісах. А тому, саме алгоритми на основі кластеризації будуть обрані для прототипу рішення.

Висновки. У роботі представлено теоретичні аспекти розробки прототипу додатку для виявлення аномалій у розподілених комп'ютерних системах. В результаті проведеного дослідження розроблено архітектуру прототипу, що базується на модульному підході з використанням сучасних технологій (Spring Cloud Sleuth, Apache Kafka, Apache Spark), що забезпечує гнучкість та масштабованість рішення. Визначено ключові метрики для моніторингу, які включають ідентифікаційні метрики (Span ID, Trace ID, назва сервісу тощо), метрики швидкодії (час відгуку, завантаження ЦПУ та пам'яті) та метрики помилок (http-статус, теги помилок). Обґрунтовано вибір алгоритмів на основі кластеризації, зокрема KMeans, для виявлення аномалій, що забезпечує високу швидкодію та ефективне використання пам'яті при обробці даних у режимі реального часу. Визначено три основні типи аномалій для моніторингу: помилки та винятки, порушення визначених порогових значень та збільшення часу відгуку запитів, що дозволяє комплексно оцінювати стан системи. Запропоновано підхід до аналізу першопричин аномалій з урахуванням архітектурного контексту, що дозволяє визначати джерело проблеми в складній розподіленій системі.

Розроблені теоретичні засади забезпечують можливість подальшого розвитку та вдосконалення системи, зокрема через додавання нових алгоритмів виявлення аномалій та розширення функціональності аналізу першопричин. Отримані результати створюють теоретичне підґрунтя для практичної реалізації системи виявлення аномалій, що може бути використана для підвищення надійності та ефективності роботи розподілених комп'ютерних систем.

Список літератури:

1. Bovenzi A., Brancati F., Russo S., Bondavalli A. A Statistical Anomaly-Based Algorithm for On-line Fault Detection in Complex Software Critical Systems. *Computer safety, reliability, and security: SAFECOMP 2011.* Berlin: Springer, 2011. P. 128-142.

2. Pitakrat T., Okanovic D., van Hoorn A., Grunske L. An Architecture-Aware Approach to Hierarchical Online Failure Prediction. *12th International ACM SIGSOFT Conference on Quality of Software Architectures.* Venice, Italy, 2016. P. 211-220.

3. Solaimani M., Iftekhar M., Khan L., Thuraisingham B. Statistical technique for online anomaly detection using Spark over heterogeneous data from multi-source VMware performance data. *IEEE International Conference on Big Data.* Washington, DC, USA, 2014. P. 1086-1094.

4. Spring Cloud Sleuth GitHub Repository. URL: https://github.com/spring-cloud/spring-cloud-sleuth (дата звернення: 02.01.2025).

5. Sigelman B. H., Barroso L. A., Burrows M. et al. Dapper, a Large-Scale Distributed Systems Tracing Infrastructure: Google Technical Report. 2010. URL: https://static.googleusercontent.com/media/research.google.com/de//pubs/archive/36356.pdf (дата звернення: 02.01.2025).

6. Apache Spark – Lightning-Fast Cluster Computing. Apache Foundation. URL: https://spark.apache.org/ (дата звернення: 02.01.2025).

7. MLlib: Main Guide – Spark Documentation. Apache Foundation. URL: http://spark.apache.org/docs/latest/ml-guide.html (дата звернення: 02.01.2025).

8. Spark Streaming Documentation. Apache Foundation. URL: http://spark.apache.org/streaming/ (дата звернення: 02.01.2025).

9. Apache Storm Documentation. Apache Foundation. URL: http://storm.apache.org/ (дата звернення: 02.01.2025).

10. Zaharia M. An Architecture for Fast and General Data Processing on Large Clusters: дис. … PhD: Electrical Engineering and Computer Sciences. University of California. Berkeley, 2014. 198 p.

11. Apache Kafka Documentation. Apache Foundation. URL: https://kafka.apache.org/ (дата звернення: 02.01.2025).

12. Spark Streaming + Kafka Integration Guide. Apache Foundation. URL: http://spark.apache.org/docs/latest/streaming-kafka-0-10-integration.html (дата звернення: 02.01.2025).

13. RabbitMQ – Messaging that just works. Pivotal. URL: https://www.rabbitmq.com/ (дата звернення: 02.01.2025).

14. Laptev N., Amizadeh S., Flint I. Generic and Scalable Framework for Automated Time-series Anomaly Detection. *Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* Sydney, Australia, 2015. P. 1159-1168.

Vakaliuk T.A., Lobanchykova N.M., Yanchuk V.M., Farrakhov O.V., Zhyliaev Ye.V. THEORETICAL ASPECTS OF PROTOTYPE APPLICATION DEVELOPMENT FOR ANOMALY DETECTION BY EVENT TRACKING IN DISTRIBUTED COMPUTER SYSTEMS

The article presents the theoretical justification and development of a prototype application for detecting anomalies in distributed computer systems by tracking events. Modern approaches to monitoring and analysing anomalies in complex distributed systems are investigated, mainly using Spring Cloud Sleuth, Apache Kafka and Apache Spark technologies. A prototype architecture based on a modular approach has been developed to ensure the flexibility and scalability of the solution. The key metrics for monitoring are identified and systematised, including identification metrics (Span ID, Trace ID, service name), performance metrics (response time, CPU and memory utilisation), and error metrics (HTTP status, error tags). Special attention is paid to the selection and justification of anomaly detection algorithms. Based on the analysis of existing approaches, clustering-based algorithms, particularly KMeans, have been selected to provide high performance and efficient memory usage in real-time data processing. Three main types of anomalies for monitoring are identified and characterised: errors and exceptions, violation of certain thresholds, and increased query response time. An innovative approach to analysing the root causes of anomalies, taking into account the architectural context, is proposed, which allows not only the detection of anomalies but also the determination of their source in a complex distributed system. A mechanism for real-time data processing using Apache Spark has been developed, which ensures fast and efficient processing of large amounts of data from various services. A unique feature of the proposed solution is the possibility of parallel use of different anomaly detection algorithms and their easy replacement or modification due to the modular architecture. The theoretical foundations create the basis for the practical implementation of an effective anomaly detection system in distributed computer systems. The proposed solution can improve the reliability and efficiency of complex distributed systems and serve as a basis for further research and improvements in anomaly monitoring and analysis.

Key words: *distributed computer systems, anomaly detection, event tracking, Spring Cloud Sleuth, Apache Kafka, Apache Spark, clustering, KMeans, root cause analysis, real-time monitoring.*