Научная статья

УДК 004.89
DOI 10.25205/1818-7900-2023-21-1-62-72

Разработка системы выявления аномалий на основе распределенной трассировки логов

Даниил Александрович Худяков

Новосибирский государственный университет
Новосибирск, Россия

khudaikoff@gmail.com, https://orcid.org/0009-0005-2747-8535

Аннотация

Разработчики программных систем должны оперативно реагировать на сбои, чтобы избежать репутационных и финансовых потерь для своих заказчиков. Поэтому важно своевременно обнаруживать поведенческие аномалии в работе программных систем. На данный момент активно развиваются различные средства для автоматического мониторинга работы систем, однако главным инструментом для анализа сбоев являются логи. Логи содержат информацию о работе системы в различных точках исполнения. Современные системы часто имеют распределенную микросервисную архитектуру, что значительно усложняет задачу анализа логов. Логи таких систем собираются централизованно из разных микросервисов, образуя огромный поток информации, которую очень сложно анализировать вручную. Однако проблему идентификации логов, относящихся к конкретному запросу в систему, решает распределенная трассировка, использование которой открывает широкие возможности для внедрения автоматического анализа. Уже существует множество решений для обнаружения аномалий в логах, однако они не используют преимущества распределенной трассировки. Статья посвящена решению задачи обнаружения поведенческих аномалий в работе распределенных программных систем на основе автоматического анализа трассировок логов. Решение основано на синтезе методов машинного обучения. Цепочки логов проходят предобработку, а также очистку с использованием методов процессной аналитики. Далее производится векторизация и кластеризация сообщений логов. После чего для анализа отклонений в последовательностях обработанных логов применяется сеть долгой краткосрочной памяти (LSTM). В результате проведенной работы был разработан и протестирован прототип системы обнаружения аномалий.

Ключевые слова

микросервисная архитектура, анализ логов, обнаружение аномалий, распределенная трассировка, машинное обучение, процессная аналитика

Для цитирования

Худяков Д. А. Разработка системы выявления аномалий на основе распределенной трассировки логов // Вестник НГУ. Серия: Информационные технологии. 2023. Т. 21, № 1. С. 62–72. DOI 10.25205/1818-7900-2023-21-1-62-72

Development of Anomaly Detection System Based on Distributed Log Tracing

Daniil A. Khudyakov

Novosibirsk State University
Novosibirsk, Russian Federation

khudaikoff@gmail.com, https://orcid.org/0009-0005-2747-8535

Abstract

Software system developers must respond quickly to failures in order to avoid reputational and financial losses for their customers. Therefore, it is important to detect behavioral anomalies in the operation of software systems in a timely manner. At the moment, various tools for automatic monitoring of systems are being actively developed, but logs are the main tool for analyzing failures. Logs contain information about the operation of the system at various points of execution. Modern systems often have a distributed microservice architecture, which significantly complicates the task of analyzing logs. Logs of such systems are collected centrally from different microservices, forming a huge flow of information that is very difficult to analyze manually. However, the problem of identifying logs related to a specific request to the system is solved by distributed tracing, the use of which opens up wide opportunities for the introduction of automatic analysis. There are already many solutions for detecting anomalies in logs, but they do not take advantage of distributed tracing. The article is considered to solving the problem of detecting behavioral anomalies in the work of distributed software systems based on automatic analysis of log traces. The solution is based on the synthesis of machine learning methods. Log traces are preprocessed and cleaned using process mining methods. Next, vectorization and clustering of log messages is performed. After that, a long short-term memory network (LSTM) is used to analyze deviations in the sequences of processed logs. As a result of the work performed, a prototype of the anomaly detection system was developed and tested.

Keywords

micro service architecture, log analysis, anomaly detection, distributed tracing, machine learning, process mining

For citation

Khudyakov D. A. Development of Anomaly Detection System Based on Distributed Log Tracing. *Vestnik NSU. Series: Information Technologies*, 2023, vol. 21, no. 1, pp. 62–72. (in Russ.) DOI 10.25205/1818-7900-2023-21-1-62-72

Введение

Большинство существующих программных систем подвержены различным сбоям из-за сложности и неизбежных недостатков программного и аппаратного обеспечения системы. Такие сбои приводят к снижению надежности, высоким финансовым затратам и могут повлиять на критически важные приложения. Таким образом, потеря контроля не допускается ни для одной системы или инфраструктуры, поскольку качество обслуживания имеет большое значение. Автоматизация обнаружения аномалий значительно помогает командам разработчиков сократить время, затрачиваемое на поиск сбоев. Это подразумевает обнаружение и распознавание паттернов, которые не соответствуют ожидаемому поведению системы. Необходимым условием для устранения аномалий [1], возникающих в любой системе, является наличие наблюдаемых данных [2] о ее работе. Наиболее полезными системными данными являются логи. Основное назначение логов – записывать состояние системы и важные события в различных критических точках выполнения. Логи являются основой многих решений, которые помогают решить проблему автоматизации обнаружения аномалий с помощью машинного обучения [3, 4].

В отличие от большинства проблем и задач машинного обучения, обнаружение аномалий касается непредсказуемых, неопределенных и редких событий, что приводит к определенным трудностям [5–7]. Современные программные системы часто имеют распределенную архитектуру, основанную на концепции микросервиса, которая обеспечивает быструю, частую и надежную доставку больших и сложных приложений. Эта парадигма в разработке программного обеспечения обеспечивает гибкость систем, но также значительно увеличивает их сложность

[8, 9]. Микросервисная архитектура предполагает разбиение системы на слабо связанные, независимо развертываемые и масштабируемые компоненты, отвечающие за узкий набор функций. Сами компоненты, как правило, представляют собой многопоточные приложения, способные обрабатывать множество запросов параллельно. Однако логи распределенных систем собираются централизованно из разных микросервисов в единое хранилище. В результате происходит их сильное перемешивание, из-за чего становится невозможным понять, например, какие логи относятся к конкретному запросу пользователя или внутренней задаче [10]. В результате анализ логов распределенных систем значительно усложняется.

Главной особенностью сбора логов распределенных систем является их автоматическое объединение в цепочки, благодаря использованию распределенной трассировки[1]. В этой статье под цепочкой (трассой) подразумевается группа логов, сгенерированных по одному пользовательскому запросу. Цепочки позволяют избавиться от неопределенности в логах, которая затрудняет их анализ. Однако существующие решения не используют преимущества распределенной трассировки, поэтому их применение в распределенных системах может иметь крайне ограниченный результат. Основной целью данного исследования является разработка нового решения, использующего возможности распределенной трассировки логов для решения проблемы автоматизации обнаружения аномалий.

В первом параграфе статьи описывается процесс сбора логов распределенных систем. Второй параграф посвящен описанию существующих решений для поиска аномалий на основе анализа логов. В третьем параграфе описывается общая архитектура предлагаемого решения. Четвертый параграф детализирует описание процесса предварительной обработки данных. В пятом параграфе описывается принцип работы сети LSTM и особенности ее настройки. В шестом параграфе приведены результаты работы над прототипом решения, протестированном на сгенерированных синтетических данных. В статье также обсуждаются дальнейшие направления исследований.

1. Распределенная трассировка

Для того чтобы с логами систем, имеющих микросервисную архитектуру, можно было работать, применяется распределенная трассировка – диагностическая методика, используемая для профилирования и мониторинга приложений, построенных с использованием архитектуры микросервисов. Распределенная трассировка открывает широкие возможности для анализа логов. В частности, разработчики и инженеры технической поддержки при ручном разборе инцидентов могут видеть контекст ошибок, что упрощает поиск причины их происхождения. Распределенная трассировка реализуется во многих современных популярных фреймворках. Например, инструмент Spring Cloud Sleuth[2] для Spring Framework. Схема его работы изображена на рис. 1.

Spring Cloud Sleuth осуществляет распределенную трассировку отдельных запросов к системе, прослеживая путь потока исполнения при обработке каждого запроса, т. е. при его прохождении через микросервисы. В процессе обработки запроса логи помечаются, благодаря чему появляется возможность легко их сгруппировать. Такую группу логов можно назвать цепочкой или трассой. Каждой цепочке присваивается идентификатор – **traceId**. Внутри цепочки логи также группируются, если внутри системы происходят запросы между микросервисами. После каждого такого запроса генерируется новый идентификатор **spanId**, который присваивается всем логам до следующего внутреннего запроса.

[1] A Quick Introduction to Distributed Tracing. URL: https://newrelic.com/sites/default/files/2021-08/quick-introduction-to-distributed-tracing.pdf

[2] Spring Cloud Sleuth. URL: https://cloud.spring.io/spring-cloud-sleuth/reference/html/

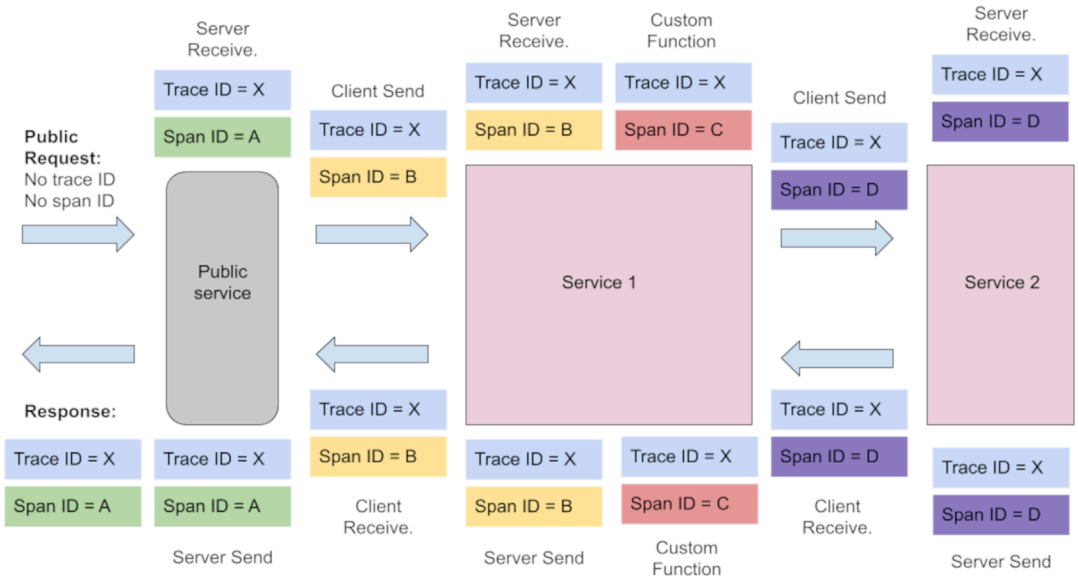

Рис. 1. Схема работы Spring Cloud Sleuth
Fig. 1. How Spring Cloud Sleuth operates

Далее рассмотрим процесс сбора и хранения логов. На данный момент в промышленной разработке повсеместно используется набор технологий под аббревиатурой ELK[3]. В данный набор входят такие продукты, как Elasticsearch, Logstash и Kibana. ELK предоставляет возможность собирать логи различных систем, хранить и визуализировать их. Elasticsearch – это распределенный поисковый и аналитический движок на базе Apache Lucene, а также хранилище данных. Logstash – это инструмент, предназначенный для сбора, фильтрации и последующего перенаправления логов систем в их конечное хранилище. Kibana – это инструмент визуализации логов, который полезен разработчикам и инженерам технической поддержки для разбора инцидентов в работе систем. Logstash позволяет гибко кастомизировать логирование, однако большинство разработчиков старается придерживаться хотя бы минимально необходимого набора атрибутов для **каждого лога**:

1) timestamp – время возникновения лога
2) level – DEBUG, INFO, и т.д.
3) applicationName – название микросервиса
4) username – инициатор запроса
5) traceId – id цепочки
6) spanId – id спана
7) message – сообщение лога
8) METHOD – название запроса в системе

Именно на основе данного набора атрибутов и будет основываться работа представленного в статье решения.

2. Существующие решения

Далее перейдем к краткому обзору существующих подходов к детектированию аномалий в логах и оценим их применимость к логам распределенных систем. Исследования в области детектирования аномалий можно разделить на два крупных направления.

[3] What is the ELK stack? URL: https://aws.amazon.com/what-is/elk-stack/

Использование статистических методов машинного обучения для анализа так называемых, message count векторов (MCV) [11, 12]. MCV аналогичен понятию мешка слов из обработки естественного языка. MVC сопоставляется цепочке логов, прошедшей стадию препроцессинга, и отображает количество произошедших в ней сообщений того или иного типа. Для анализа MVC используются различные подходы. Один самых популярных – это использование метода главных компонент (PCA). У данного подхода имеется ряд минусов, которые мешают его использованию в распределенных системах. Рассмотрим их:

1) Логи нестабильны. Системы постоянно обновляются, появляются новые логи, а старые выходят из использования. При этом размерность MCV соответственно должна изменяться. Модель, соответственно, должна переобучаться с новыми параметрами из-за чего использование статических методов не подходит для задачи онлайн-детектирования.

2) Не учитывается контекстная информация. При анализе используется только частота появлений тех или иных сообщений в последовательности логов. Контекстная информация при этом никак не учитывается, из-за чего не удается определить степень важности отдельного сообщения в рамках контекста.

Другой подход основан на использовании методов глубокого машинного обучения. Главное направление в этой области – это анализ цепочек логов с помощью глубоких нейронных сетей долгой краткосрочной памяти (LSTM) [13, 14]. Предсказание при этом подходе является вероятностным и базируется на анализе цепочки логов заданной длины, предшествующей рассматриваемому логу, называемой окном. Лог считается аномальным, если он не попадает в заданное число наиболее вероятных, предсказанных моделью.

Однако прямое использование одного из существующих решений на основе LSTM-сетей применительно к логам распределенных может не дать хороших результатов в силу следующих причин:

1) Использование готовых парсеров логов может не сработать. Необходимо учитывать специфику происхождения логов. Часто более производительным и точным вариантом препроцессинга будет использование собственного решения, адаптированного, например, для сообщений в формате JSON или XML.

2) Не решается проблема перемешивания логов. Если брать логи напрямую в порядке их записи в хранилище, то сформированные окна логов, на основе которых производится анализ, потенциально будут содержать сообщения из разных запросов, от разных пользователей, что лишает их смысла.

3) Проблема выбора ширины окна. Крайне важным гиперпараметром моделей на основе LSTM-сетей является ширина окна рассматриваемых логов. Однако цепочки логов имеют самую разную длину. И решение, использующее одну заданную ширину окна, смогло бы обнаружить лишь некоторые из закономерностей в логах, что сильно снижает его точность.

Перечисленные проблемы создают предпосылки для создания нового решения для детектирования аномалий в логах, которое могло бы учитывать специфику сбора логов распределенных систем.

3. Архитектура решения

Далее рассмотрим общую архитектуру предлагаемого решения. На рис. 2 схематически изображена схема его работы. На вход в систему поступает множество «сырых» логов, каждый из которых имеет описанную структуру. Далее на основе **traceId** производится формирование цепочек логов, после чего цепочки группируются в зависимости от «активности», т. е. в зависимости от типа запроса в систему (определяется на основе поля **method** первого лога в цепочке). Далее для каждой активности последовательно производится построение модели процесса, очистка от шума, векторизация и последующая кластеризация сообщений отдельных логов в цепочках. После чего преобразованные цепочки поступают на вход LSTM-модели для ее обучения.

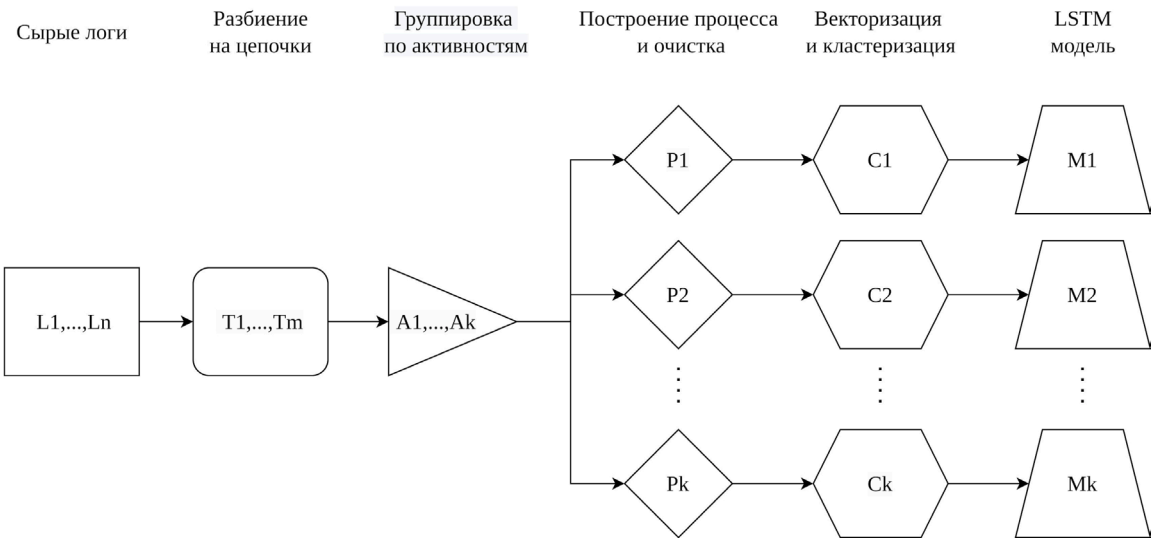

Рис. 2. Процесс обучения системы детектирования аномалий
Fig. 2. Process of training the anomaly detection system

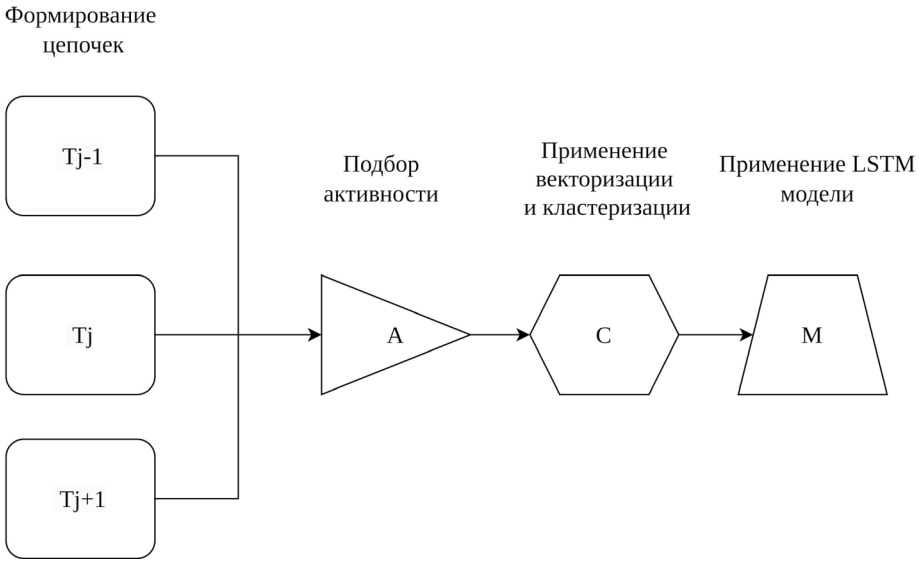

Рис. 3. Процесс предсказания аномалий
Fig. 3. Anomaly prediction process

Процесс предсказания аномалий для новых данных изображен на рис. 3. Из множества новых рассматриваемых логов выделяются цепочки, после чего каждая цепочка анализируется отдельно. Для цепочки подбирается активность, она обрабатывается с использованием ранее обученных моделей векторизации и кластеризации. И далее преобразованная цепочка поступает на вход LSTM-модели, которая делает предсказание для каждого отдельного взятого лога.

4. Предобработка данных

После группировки цепочек для каждой активности производится построение модели процесса. Как описывалось выше, принимающий запрос от пользователя микросервис внутри системы может вызывать запросы у других микросервисов, которые, в свою очередь, также

могут вызывать запросы по цепочке. Таким образом, формируется цепочка вызовов запросов в системе. Важно, что элементы множества цепочек, относящихся к одной активности (представляющих один тип запроса в систему), могут сильно различаться с точки зрения последовательности внутренних запросов т. к. последовательность может зависеть от параметров запроса. Построив все варианты последовательностей внутренних запросов для активности, мы можем получить описание процесса данной активности. В частности, такой процесс можно изобразить, построив взвешенный ориентированный граф (directed flow graph), где вершины – это названия запросов в системе, а ребра обозначают последовательный вызов пары запросов, вес каждого ребра – количество последовательных вызовов. Такой граф интересен с точки зрения применения к нему методов процессной аналитики (process mining).

При дальнейшем обучении LSTM-модели важно, чтобы в тренировочной выборке присутствовали только сильные закономерности, для этого нам нужно попытаться избавиться от шума. Для этого было решено использовать алгоритм Inductive Miner Infrequent [15] из библиотеки pm4py[4]. Применение этого алгоритма к взвешенному ориентированному графу в результате дает дерево переходов, на соответствие которых, мы можем проверить каждую цепочку, относящуюся к активности. Данный алгоритм позволяет конфигурировать параметр границы шума (noise_threshold), его настройка приводит к изменению дерева переходов т. к. из него выпадают редко происходящие переходы. Таким образом, выбрав значение параметра границы шума и построив дерево переходов, мы можем отбросить часть цепочек логов, относящихся к активности, которые не могут быть выровнены по построенному дереву переходов. В итоге данный метод был использован для очистки от шума тренировочной выборки.

Далее после формирования тренировочной выборки для каждой активности производится очистка сообщений всех логов от параметров. Для этого из сообщений логов исключатся все цифры, специальные символы, а также ряд известных заранее паттернов. В частности, известно, что современные веб-сервисы для передачи данных чаще всего используют сообщения в форматах JSON и XML. Можно сказать, что данные целиком являются параметрами, поэтому также должны быть исключены из сообщения лога, что и было сделано в рамках реализации решения.

После очистки сообщений происходит их векторизация. Для этого применяется алгоритм Doc2Vec [16]. Выбор этого алгоритма обоснован тем, что в результате его применения получаются векторы заданной размерности, которую можно задать небольшой, что ускорит дальнейшую кластеризацию. Другим важным аргументов является то, что полученную в результате применения алгоритма модель можно интеративно дообучать без полного пересчета (в отличие от, например, TF-IDF), что крайне важно для реализации онлайн-детектирования аномалий, когда система постоянно подкрепляется новыми данными. После векторизации сообщения логов кластеризуются, чтобы в результате преобразовать последовательности сообщений логов в последовательности чисел (каждому логу в результате кластеризации назначается числовая метка). Для кластеризации используется алгоритм KMeans. Число соседей подбирается путем рассмотрения длин цепочек активности, рассматриваются варианты в диапазоне от минимальной до максимальной длины цепочки. Лучший вариант выбирается на основе расчета коэффициента силуэта.

5. Детектирования аномалий

После завершения этапа препроцессинга цепочки, представленные числовыми последовательностями, используются для обучения нейронной сети LSTM. LSTM (long short-term memory) нейронная сеть – это рекуррентная нейронная сеть, способная к обучению долговременным зависимостям. Такие модели хорошо подходят для анализа событий, продолжающихся во времени. Кроме того, данная нейронная сеть обучается без учителя. Такой вариант

[4] pm4py documentation. URL: https://pm4py.fit.fraunhofer.de/documentation

обучения является единственно возможным вариантом при работе с логами распределенных систем, так как произвести их ручную разметку, когда логов становится огромное количество, невозможно.

Гиперпараметром модели является ширина окна **h**, т. е. количество событий (в нашем случае логов), предшествующих рассматриваемому событию, на основе которых делается предсказание. LSTM-модель обучается искать закономерности в последовательностях, после чего выдает вероятность для рассматриваемого события быть каждым из известных событий. После чего задается порог **k** (также гиперпараметр), и если рассматриваемое событие не попадает в список **k** наиболее вероятных предсказанных событий, то оно считается аномальным.

6. Результаты

В соответствии с описанным алгоритмом был реализован прототип системы детектирования аномалий. Также для его тестирования был разработан алгоритм генерации синтетических данных, результатом работы которого являются тренировочная и тестовая выборки, сгенерированные на основе линейной последовательности логов, представленной графом, и в которую специальным образом внесены отклонения, которые считаются аномалиями. Для построения графа использовалась библиотека NetworkX[5]. Пример последовательности, для которой производились расчеты, изображен на рис. 4. Выборки генерируются путем обхода графа. Тренировочная выборка генерируется по узлам, которые не считаются аномальными (на рисунке выделены зеленым), в тестовой же выборке часть примеров генерируется с проходом по узлам, представляющим аномалии (на рисунке выделены красным). Для генерации сообщений логов использовалась нейронная сеть gpt-neo[6].

Параметры генерации данных для тестирования отображены в табл. 1.

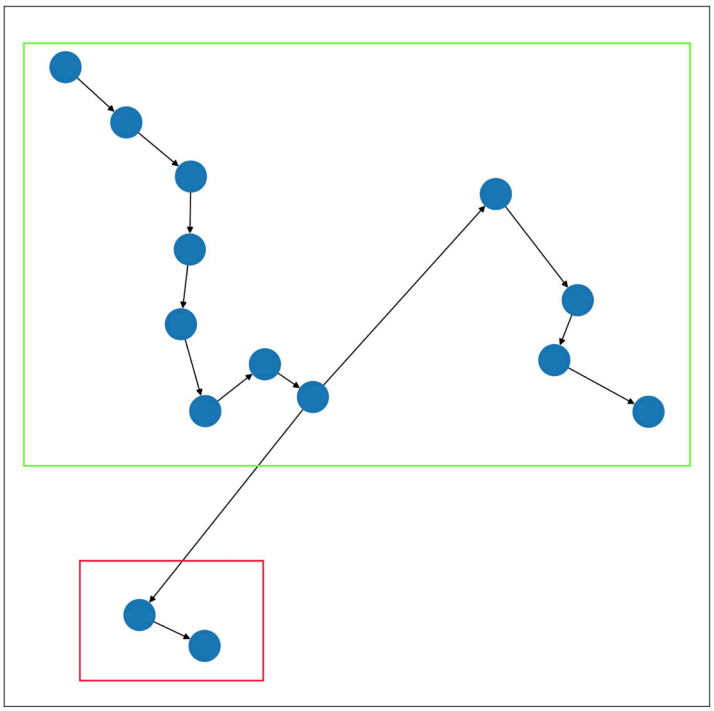

Рис. 4. Граф для генерации выборок
Fig. 4. Graph for generating samples

[5] NetworkX. URL: https://networkx.org/
[6] GPT Neo. URL: https://github.com/EleutherAI/gpt-neo

Таблица 1

Параметры генерации выборок

Table 1

Sample Generation Options

Размер тренировочной выборки	300
Размер тестовой выборки	100
Процент аномальных примеров в тестовой выборке	25%
Длина цепочки логов	12
Количество спанов	3
Максимальная длина сообщения лога	8

Таблица 2

Результаты тестирования

Table 2

Test Results

Точность	**Полнота**	**F1 – мера**
0.969	0.64	0.77

Результаты расчета метрик качества классификации (мы решаем задачу бинарной классификации) на тестовой выборке отображены в табл. 2. Как можно увидеть, прототип хорошо справляется с поиском аномалий. Высокая точность (precision) показывает, что система хорошо умеет определять именно аномальные логи, однако результаты могут быть улучшены с точки зрения полноты (recall).

Заключение

Микросервисная архитектура современных программных систем усложняет задачу обнаружения сбоев, главным инструментом для поиска которых являются логи. Распределенная трассировка логов открывает большие возможности для их анализа. Однако существующие решения для автоматического обнаружения аномалий в логах не используют преимущества распределенной трассировки. В данной работе предлагается новое решение для поиска аномалий в логах распределенных систем, имеющих микросервисную архитектуру. В ходе работы была спроектирована архитектура системы выявления аномалий на основе логов, собранных с использованием распределенной трассировки. В соответствии с архитектурой был реализован прототип. Для тестирования прототипа был реализован алгоритм генерации синтетических размеченных данных. Тестирование прототипа на сгенерированных тестовых данных показало, что система хорошо справляется с поиском аномалий в логах с точки точности прогнозирования, однако алгоритм может быть улучшен с точки зрения полноты классификации аномалий.

В дальнейшем планируется протестировать работу системы на более сложных синтетических данных, в которые будет вноситься больше шума, свойственного реальным системам. Также планируется работа над улучшением метрик качества классификации аномалий, в том числе путем настройки гиперпараметров моделей. Главным же направлением развития системы будет ее доработка для реализации возможности итеративного дообучения, что необходимо для решения задачи онлайн-детектирования аномалий. После доработок система будет способ-

на автоматически дообучаться на новых данных через каждый заданный промежуток времени, таким образом адаптируясь под изменения системы, являющейся источником логов.

Список литературы/References

1. **Chandola V., Banerjee A., Kumar V.** Anomaly Detection: A Survey. ACM Computing Surveys (CSUR) 41.3, 2009. p. 1–58. DOI: 10.1145/1541880.1541882
2. **Sridharan C. D**istributed Systems Observability: A Guide to Building Robust Systems. O'Reilly Media, 2018.
3. **Sultana N., Chilamkurti N., Peng, W., Alhadad R.** Survey on SDN Based Network Intrusion Detection System Using Machine Learning approaches. Peer-to-Peer Networking and Applications, 2018. p. 1–9. DOI: 10.1007/s12083-017-0630-0
4. **Pang G., Shen C., Cao L., van den Hengel A.** Deep Learning for Anomaly Detection: A Review, 2020. arXiv: 2007.02500. DOI: 10.1145/3439950
5. **Palchunov D. E., Yakhyaeva G. E.** Integration of Fuzzy Model Theory and FCA for Big Data Mining. SIBIRCON 2019 – International Multi-Conference on Engineering, Computer and Information Sciences, Proceedings, 2019. p. 961–966. DOI: 10.1109/sibircon48586.2019.8958216
6. **Yakhyaeva G. E.** Application of Boolean Valued and Fuzzy Model Theory for Knowledge Base Development. SIBIRCON 2019 - International Multi-Conference on Engineering, Computer and Information Sciences, Proceedings, 2019. p. 868–871. DOI: 10.1109/sibircon48586.2019.8958245
7. **Palchunov D. E.,. Tishkovsky D. E,. Tishkovskaya S. V, Yakhyaeva G. E.** Combining logical and statistical rule reasoning and verification for medical applications. Proceedings – 2017 International Multi-Conference on Engineering, Computer and Information Sciences, SIBIRCON 2017, 2017, p. 309-313. DOI: 10.1109/sibircon.2017.8109895
8. **Esposito C., Castiglione A., Choo K. R.** Challenges in Delivering Software in the Cloud as Microservices. IEEE Cloud Computing 3.5, 2016, p. 10–14. DOI: 10.1109/mcc.2016.105
9. **Gunawi H. S., Hao M., Leesatapornwongsa T., Patana-anake T., Do T., Adityatama J., Eliazar K. J., Laksono A., Lukman J. F., Martin V.** What Bugs Live in the Cloud? A Study of 3000+ Issues in Cloud Systems. Proceedings of the ACM Symposium on Cloud Computing, 2014. p. 1–14. DOI: 10.1145/2670979.2670986
10. **Beschastnikh I., Wang P., Brun Y., Ernst M. D.** Debugging distributed systems: Challenges and options for validation and debugging. Communications of the ACM, vol. 59, no. 8, Aug. 2016. p. 32-37. DOI: 10.1145/2927299.2940294
11. **Xu W., Huang L., Fox A., Patterson D., Jordan M. I.** Detecting large-scale system problems by mining console logs. Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (SOSP '09). Association for Computing Machinery, New York, NY, USA, 2009. p. 117–132. DOI: 10.1145/1629575.1629587
12. **Vaarandi R.** A data clustering algorithm for mining patterns from event logs. Proc. 3rd IEEE Workshop IP Oper. Manage, Oct. 2003. p. 119–126. DOI: 10.1109/ipom.2003.1251233
13. **Du M., Li F., Zheng G., Srikumar V.** Deeplog: Anomaly detection and diagnosis from system logs through deep learning. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS), 2017. p. 1285-1298. DOI: 10.1145/3133956.3134015
14. **Zhang X., Li Z., Chen J.** Robust log-based anomaly detection on unstable log data. Proceedings of the 2019 27th ACM Joint Meeting, Tallinn, Estonia. 26–30 August 2019. p. 807–817. DOI: 10.1145/3338906.3338931
15. **Leemans S.J.J., Fahland D., van der Aalst W.M.P.** Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour. Business Process Management Workshops. Lecture Notes in Business Information Processing, vol 171. Springer, Cham, 2013. DOI: 10.1007/978-3-319-06257-0_6

16. **Le Q., Mikolov T.** Distributed Representations of Sentences and Documents. 31st International Conference on Machine Learning, ICML 2014, 2014. DOI: 10.18653/v1/s17-1003

Информация об авторе

Худяков Даниил Александрович, студент НГУ

Information about the Authors

Daniil A. Khudyakov, Student, Novosibirsk State University (Novosibirsk, Russian Federation)
ResearcherID: IAN-8563-2023

Статья поступила в редакцию 05.04.2023;
одобрена после рецензирования 19.04.2023; принята к публикации 19.04.2023
The article was submitted 05.04.2023;
approved after reviewing 19.04.2023; accepted for publication 19.04.2023