УДК 004.89

ИНТЕЛЛЕКТУАЛЬНЫЙ АНАЛИЗ ПРОГРАММНЫХ АНОМАЛИЙ

В. В. Краснопрошин[1], А. И. Наумович[2]

[1] Д. т. н., профессор, заведующий кафедрой информационных систем управления Белорусского государственного университета, Минск, Беларусь

[2] Магистрант кафедры информационных систем управления Белорусского государственного университета, Минск, Беларусь

Реферат

В работе рассмотрена актуальная прикладная проблема, связанная с анализом аномальных ситуаций в программных логах на этапе исполнения программного продукта. Предложен подход к распознаванию аномальных событий, основанный на построении модели процесса исполнения программы в виде взвешенного орграфа, который отображает взаимосвязи между функциями программы и ее переходами. Разработан комплекс оригинальных алгоритмов предобработки, анализа и распознавания данных, которые в совокупности эффективно решают поставленную задачу.

Ключевые слова: программные логи, анамальная ситуация, формальная модель, нейронная сеть, классификатор

INTELLIGENT ANALYSIS OF SOFTWARE ANOMALIES

Krasnoproshin V. V., Naumovich A. I.

Abstract

The paper considers an actual applied problem related to the analysis of abnormal situations in program logs. An approach for recognizing anomalous events is proposed. The approach is based on constructing a program execution model in the form of a weighted digraph that displays the relationship between program functions and its transitions. A set of original algorithms for data preprocessing, analysis, and recognition has been developed. The algorithms effectively solve the stated problem.

Keywords: program logs, anomalous situation, formal model, neural network, classifier.

Введение

В условиях массовой цифравизации современной экономики быстро растут объемы программных продуктов, направленных на автоматизацию процессов обработки данных [1]. Корректность выполнения программ напрямую определяет прибыль предприятия, получаемую от средств автоматизации. К сожалению, не все ошибки могут быть обнаружены на стадии разработки программ и их тестирования. Многие из них могут проявиться уже на стадии исполнения программного продукта. Поэтому актуальной в настоящее время является задача обнаружения таких ошибок и аномалий.

Методы решения проблемы существуют и постоянно развиваются. Можно выделить два основных направления к ее решению. Первый (эмпирический) основан на анализе признакового описания всей системы и направлен на уменьшение объема данных, подвергаемых ручному анализу программного кода экспертами. Второй (аналитический) базируются на анализе признаковых описаний программных логов и различного рода цепочек событий.

Однако, несмотря на многочисленные попытки, проблема далека от ее окончательного разрешения. В данной работе исследуется один из возможных вариантов решения проблемы в рамках именно второго из указанных направлений.

Формализация проблемы и постановка задачи

Прежде чем перейти к рассмотрению проблемы, введем минимальный понятийный каркас для ее формального описания [2].

Пусть, например, задан некоторый программный лог $L=(l_1,l_2,...,l_n)$, состоящий из строк, разделенных символом-разделителем. Предположим, что для этого лога выделено множество шаблонов (или событий) E^* и построено отображение T_E^{-1}, ставящее в произвольной строке лога в соответствие с ее шаблоном. Предположим также, что с помощью удаления части строки лога, не входящей в шаблон, выполнен переход к сжатому его представлению $\bar{L}=(E_1,E_2,...,E_n)$. Здесь E_i – событие, шаблону которого соответствует строка l_i исходного программного лога.

Естественным структурным преобразованием формального представления лога является цепочка событий. Цепочкой длины k, начинающейся с события E_i, называется k подряд идущих в преобразованном логе событий. Обозначим такую цепочку через $s_i^k=(E_i,E_{i+1},...,E_{i+k})$, $k \geq 0$. Событие преобразованного лога принадлежит цепочке $E_j \in s_i^k$, если $i \leq j \leq i+k$.

На основании введенных понятий исследуемую проблему формально можно сформулировать следующим образом.

Постановка задачи

Пусть задан программный лог $L=(l_1,l_2,...,l_n)$, где l_i – i-я по порядку строка лога. Предположим, что с помощью «обратного» преобразования выполнен переход к формальному

его представлению $\bar{L}=\left(E_{i_1},E_{i_2},...,E_{i_n}\right)$, $E_j\in E^*$, на основании которого построено множество всевозможных цепочек $\left\{s_i^k\middle|\forall k,\forall i\right\}$.

Требуется построить алгоритм $A\colon L\times\bar{L}\times\left\{s_i^k\right\}\to L$, который на основе анализа входных данных выделял бы в программном логе аномальные события.

Решение

Процесс решения поставленной задачи предлагается реализовать в виде последовательного рассмотрения трех взаимозависимых подзадач: выделения шаблонов событий, анализа цепочек и выделения аномального события. При этом первые из них играют роль своеобразной предобработки данных.

Рассмотрим более подробно каждую из подзадач и опишем процесс их решения.

Выделение шаблонов событий

При отсутствии универсального (по отношению к любому набору данных) подхода предлагается использовать алгоритм, состоящий из набора процедур, зависящих от характеристик анализируемого программного лога.

В предлагаемом алгоритме первая процедура основана на идее выделения классов эквивалентности. Она реализуется над векторными представлениями с помощью модели Word2Vec [3] и последующей их кластеризацией. Процедура эффективна для конструкций естественного языка.

Вторая – на выделении в коллекции текстовых документов классов полудублей [4]. Хорошо работает в информационном поиске, когда строки лога, соответствующие одному событию, отличаются незначительно (длина параметрической части шаблона сравнительно небольшая).

Наконец, третья процедура основана на модели мешка слов. Строки лога рассматриваются как множество N-грамм, на основании которых строится их признаковое описание. Процедура обрабатывает любой программный лог и рекомендуется во всех остальных случаях.

Таким образом, рассмотренные выше алгоритмы позволяют для каждой строки программного лога получить шаблон ее события с помощью выделения классов эквивалентности и замены элементов классов их представителем.

Этап анализа цепочек необходим для сокращения числа исследуемых цепочек, составленных из некоторого числа подряд идущих шаблонов событий. Рассматриваются цепочки фиксированной длины, величина которой выбирается экспертом. При этом последняя должна быть достаточной для обнаружения зависимостей в переходах между событиями. Цепочки, выделенные алгоритмом для дальнейшего рассмотрения, назовем сомнительными или подозрительными на аномальность.

Модель

Поскольку цепочка (на содержательном уровне) отображает взаимосвязь между функциями программы и ее переходами, предлагается построить формальную модель процесса, отражающего эти взаимосвязи.

Введем необходимые для этого понятия и определения.

Пусть задан программный лог $V=\left(l_1,...,l_n\right)$. Будем говорить, что l_i предшествует l_j, $l_i\to l_j, l_i, l_j\in V$, если запись, соответствующая l_i встречается в логе не позднее записи, соответствующей l_j. Нетрудно видеть, что отношение $\to$ на множестве строк лога образует частичный порядок.

Используя данное отношение, построим множество $E=\left\{,i,j.\middle|l_{\perp}i\to l_{\perp}j\right\}$. Тогда пара $G=\left(V,E\right)$ задает орграф [5]. Нетрудно показать, что с помощью выделения событий на множестве строк лога можно построить классы эквивалентности. Это можно сделать путем замены вершин, входящих в один и тот же класс ее представителем, сохранив при этом все ребра. В результате будет построен соответствующий мультиграф G', который и отражает отношение между шаблонами и программными событиями.

Рассмотрим теперь G' и его преобразования в граф вызовов, необходимые для использования в процессе анализа цепочек.

Пусть в процессе выполнения функции f вызываются функции $a_1,...,a_k$. Каждая из этих функций запрашивает логгирование у модуля логгирования ровно один раз. Для простоты будем считать, что параметры событий отсутствуют.

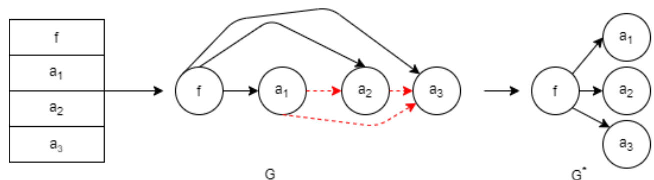

Рисунок 1 – Пример G для независимых вызовов внутри функции

Поскольку функция f последовательно вызывает $a_1,...,a_k$, то очевидно, что в программном логе событие f происходит не позже a_1, a_1 – не позже a_2 и так далее. Используя отношение предшествования, построим граф $G=G'$. Пример такого графа для $k=3$ представлен на рисунке 1. Красным пунктиром выделены дуги G', которые не могут существовать при последовательных вызовах функций a_1,a_2,a_3 функцией f. Построенный таким образом G^* называем графом вызовов.

В графе G также необходимо учитывать наличие дуг, искажающих реальную схему вызовов. Пусть, например, функция f вызывается внутри некоторого цикла (рисунок 2).

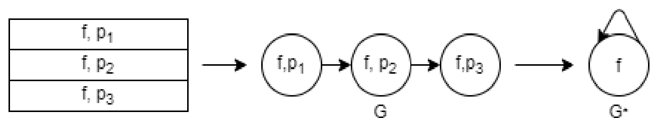

Рисунок 2 – Вызов в цикле

В этом случае строкам лог-файла будет соответствовать одно и то же событие. Будем считать, что строки лога различаются в параметрической части. На графе вызовов G^* вызов функции в цикле выражается в виде петли у соответствующей вершины. Отметим, что аналогично можно отображать рекурсивный вызов функции.

Предположим теперь, что некоторая функция f содержит условный переход $P\left(x\right)$. Если $P\left(x\right)=1$, выполняется функция a, иначе выполняется функция b. Отличительной особенностью условного перехода является отсутствие заранее определенной очередности вызова функций. В этом слу-

doi.org/10.36773/1818-1212-2020-123-5-21-25

чае условный переход от события f к событиям a, b предлагается заменить вероятностным переходом.

Пусть в итоге проведенных построений получен граф вызовов $G^* = (V^*, E^*)$. Вершины такого графа соответствуют событиям, а дуги – отношениям «вызова». Иными словами, если $(a, b) \in E^*$, то функция a в исходном коде может вызывать функцию b. На содержательном уровне введенные вероятности носят смысл «вероятности продолжения» выполнения ветви программы. Поэтому каждой дуге поставим в соответствие вероятность перехода.

Таким образом, в результате всего вышеописанного построена модель процесса в виде взвешенного графа вызовов. Нетрудно видеть, что она формально отражает взаимосвязи между элементами программы и переходами между ними. Опишем теперь кратко процесс построение модели в виде алгоритма.

Алгоритм

Он состоит из следующих основных шагов:

Шаг 0. Фиксируется ширина скользящего окна равной длине цепочки и последовательность событий $e_1, e_2, \ldots, e_m$.

Шаг 1. Извлекаются события, попавшие в окно. Пусть, например, $e_{i_1}, e_{i_2}, \ldots, e_{i_w}$ – извлеченные события для некоторой позиции окна и i – номер текущей цепочки.

Шаг 2. Для каждой цепочки строится подграф графа вызовов: $G^i = (V, E^i)$, где

$$E = \left\{ \left(e_{\perp}(i, j), e_{\perp}\left(i_{\perp}(j+1)\right) \right) + \left| \forall j \in [1, w-1] \right. \right\}.$$

Шаг 3. Выполняется объединение подграфов $G = U_i G^i$.

Шаг 4. Производится «редукция» дуг для графа G: множество дуг между двумя вершинами заменяется одной дугой с весом, равным числу замененных дуг. Пусть $\bar{G}$ – полученный граф, а $w(x, y)$ – функция веса ребра.

Шаг 5. Для каждой вершины выполняется нормировка весов выходящих из нее дуг: $\forall \upsilon \in V$, $\forall u \in N^+(\upsilon)$: $w^*(\upsilon, \mathrm{u}) = w(\upsilon, u) \Big/ \sum_{u \in N^+(\upsilon)} w(\upsilon, u)$. Полученная таким образом функция $w^*(x, y)$ задает требуемые вероятности.

Для оценки эффективности работы алгоритма определим количество времени и объемы памяти, необходимые для его выполнения.

Пусть, например, задан программный лог в сжатом представлении, состоящий из n событий. Число уникальных событий в логе равно $|V|$. Пусть также задана ширина окна w. Для хранения всего графа необходимо $O(|V| + |E|)$ памяти, где $|E|$ – число дуг в графе.

В рамках одной фиксированной позиции скользящего окна шириной w на подсчет дуг необходимо затратить $O(w)$ времени. Если сдвиг скользящего окна равен δ, то всего различных позиций скользящего окна порядка $O\left(\frac{n}{\delta}\right)$. Тогда суммарное время построения графа без нормализации составляет порядка $O\left(nw/\delta\right)$. При проведении нормализации должны быть просмотрена ребра, исходящие из вершины. Максимальное суммарное число вершин составляет $|V|$, исходящих ребер из вершины в худшем случае $|V| - 1$. В этом случае на подсчет нормировочного коэффициента необходимо затратить порядка $O(|V|)$ операций.

В итоге время нормировки ребер графа составляет порядка $O\left(|V|^2\right)$, а суммарное время построения графа вызовов – порядка $O\left(nw/\delta + |V|^2\right)$.

Перейдем теперь к процессу анализа цепочек. Для решения данной подзадачи предлагается алгоритм, который основан на использовании модели графа вызовов $G^* = (V, \mathrm{E})$. С целью повышения эффективности процесса анализа в рамках алгоритма предлагается использовать процедуры, которые бы максимально учитывали специфику конкретного процесса.

Пусть, например, задана цепочка событий $e_{i_1}, e_{i_2}, \ldots, e_{i_w}$, для которой в G^* были получены соответствующие переходные вероятности $p_1, \ldots, \mathrm{p}_{w-1}$. Здесь $p(a, b)$ – вероятность перехода из события a к событию b. Цепочку будем считать аномальной, если в ней существует событие e_{i_k} в этой цепочке, такое, что $e_{i_k} \neq \operatorname{argmax}_{a \in N^+\left(e_{i_{k-1}}\right)} p\left(e_{i_{k-1}}, a\right)$.

То есть цепочка, которая отклоняется в набольшей степени от вероятного пути в графе вызовов. На практике такие отклонения происходят достаточно часто. Поэтому сам факт отклонения нельзя считать высоко-информативным для обнаружения аномальных цепочек.

В связи с этим предлагается ввести так называемую меру аномальности. В простейшем случае она может зависеть только от переходных вероятностей и их очередности.

Процедуру, основанную на указанной мере аномальности, предлагается строить путем выбора наиболее подходящей функции из заранее фиксированного параметрического семейства. Предлагаемая процедура состоит из следующих основных шагов:

Шаг 1. Фиксируется параметрическое семейство функций, например, Ξ.

Шаг 2. Строится граф вызовов G^* и произвольный вектор начальных параметров $\theta^0 \in \Xi$.

Шаг 3. Для каждой цепочки:

Шаг 3.1. Формируется пакет нормальных и сомнительных переходов x, а также вектор целевых переменных y. Если переход x_i – сомнительный, то $y_i = 1$, иначе $y_i = 0$.

Шаг 3.2. Вычисляется среднеквадратичная ошибка на множестве: $MSE = \frac{1}{n} \sum_{i=1}^{n} (x_i - y_i)^2$.

Шаг 3.3. Выполняется градиентный шаг по параметру θ относительно функции ошибки: $\theta^{i+1} = \theta^i - \alpha \nabla L$.

Шаг 3.4. Переход к следующей цепочке.

В данной процедуре наиболее сложным шагом является выбор параметрического семейства функций. Для предсказания события, следующего за цепочкой, предлагается использовать нейронную сеть [6].

Предположим, что предшествующие алгоритмы в качестве входных данных получали вектор, составленный из пе-

doi.org/10.36773/1818-1212-2020-123-5-21-25

реходных вероятностей цепочки. При таком подходе, очевидно, не учитывается тип события и прочая информация. Предлагается дополнительно к входным данным использовать признаковые описания наблюдаемой строки программного лога. В этом случае на вход нейронной сети подается цепочка событий $e_i,\ldots,e_w$ и признаковые описания соответствующих строк $x_i,\ldots,x_w$. Возможный вариант архитектуры сети представлен на рисунке 3.

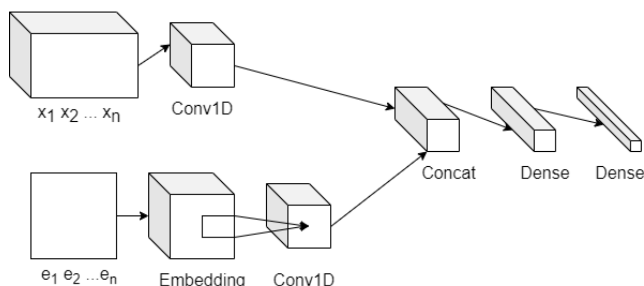

Рисунок 3 – Архитектура нейронной сети

Выбор весов слоев нейронной сети можно, например, выполнить с помощью метода градиентного спуска. Укажем вид функции потерь.

Поскольку нейронная сеть должна предсказывать следующее за цепочкой событие, ее выходом должно быть вероятностное распределение на множестве всех известных событий $P\left(e_{\perp}\left(w+1\right)+e_{\perp}1,\ldots,e_{\perp}w,x_{\perp}1,\ldots,x_{\perp}w\right)$. Непосредственной функцией потерь является кросс-энтропия между предсказанным распределением и вырожденным распределением (следующего наблюдаемого события).

Таким образом, в результате выполнения описанного выше процесса анализа будет построено множество всех сомнительных цепочек.

Выделение аномального события

Заключительным этапом решения задачи является локализация аномалии в сомнительной цепочке [7]. Для этого предлагается провести бинарную классификацию событий цепочки с учетом ее контекста и исчерпания многообразия программных логов (рисунок 4).

Таким образом, в результате заключительного этапа определяется в конечном итоге подмножество событий, которые классифицируются в подозрительной цепочке как аномальные.

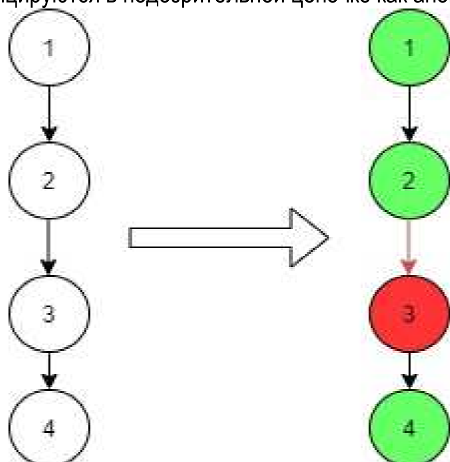

Рисунок 4 – Локализация аномального события

Замечание

Отличительной особенностью заключительного этапа является отсутствие меток классов для обучения классификатора. В данном случае проверка результатов классификации проводилась экспертом, который в итерационном режиме предсказаний и проверок настраивал классификатор. При этом эксперт проверял предсказания только на объектах, которые, по его мнению, расположены достаточно близко к разделяющей поверхности между классами. Такой прием позволил достаточно быстро аппроксимировать необходимую поверхность и значительно ускорить процесс обучения.

Пример

Рассмотрим возможности предложенного подхода на примере решения задачи по обработке программного лога системы HDFS [8]. Это распределенная файловая система, активно используемая Apache Hadoop [9]. Известно, что работоспособность указанной системы напрямую зависит от максимально быстрого определения момента выхода узлов кластера из строя и потери соединения между ними. Для выявления такого рода неисправностей предлагается использовать программные логи файловой системы HDFS.

Получение реальных данных определяется политикой конфиденциальности компаний и стоимостью аннотирования датасетов такого типа. Поэтому необходимые для экспериментов наборы данных, как и многие прочие, находящиеся в репозитории [10], были получены синтетическим путем в лаборатории. Однако условия генерации логов при этом были максимально приближены к реальности.

В ходе экспериментов из исходного набора логов было выделено 113 шаблонов событий. Для анализа цепочек событий использовалась нейронная сеть. Кроме того, был проведен дополнительный этап анализа, основанный на обучении нейронной сети экспертом.

После завершения обучения получены следующие результаты по выявлению аномальных: **precision 0.05, recall 1.0.**

Полученные результаты свидетельствуют о том, что все существующие в логах аномалии практически были обнаружены. Здесь следует отметить, что алгоритм допускал достаточно много так называемых «ложных срабатываний». Однако их значимость, как показала практика, гораздо ниже стоимости пропуска реальных аномалий. Поэтому полученные результаты можно считать весьма обнадеживающими. То есть результаты экспериментов подтвердили эффективность предложенного подхода и его пригодность к практическому использованию.

Заключение

В работе рассмотрена прикладная проблема, связанная с анализом аномальных ситуаций в программных логах на этапе их исполнения. Предложен подход к распознаванию аномальных событий, основанный на построении модели процесса исполнения программы в виде взвешенного орграфа. Модель отображает взаимосвязи между функциями программы и ее переходами. Разработан комплекс оригинальных алгоритмов предобработки, анализа и распознавания данных, которые в совокупности эффективно решают поставленную задачу.

Экспериментальные исследования, проведенные, в частности, на примере решения актуальной прикладной задачи – обнаружения аномалий в логах распределенной файловой системы HDFS, подтвердили эффективность предложенного подхода в целом.

doi.org/10.36773/1818-1212-2020-123-5-21-25

Список цитированных источников

1. Виссия, Х. Э. Р. М. Принятие решений в информационном обществе: учебное пособие / Х. Э. Р. М. Виссия, В. В. Краснопрошин, А. Н. Вальвачев. – Санкт-Петербург : Лань, 2019. – 228 с.
2. Krasnoproshin, V. Solution of applied problems: formalization, methodology and justification / V. Krasnoproshin, V. Obraztsov, H. Vissia // World Scientific Proceeding Series on Computer Engineering and Information Science. – Vol. 3 : Computational Intelligence in Business and Economics. – London : World Scientific, 2010. – P. 57–64.
3. Distributed Representation of Word and Phrases and their Compositionality [Electronic resource]. – Mode of access : https://papers.nips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf. – Date of access : 30.11.2020.
4. Detecting Near-Duplicates for Web Crawling [Electronic resource]. – Mode of access : https://static.googleusercontent.com/media/ research.google.com/ru//pubs/archive/33026.pdf. – Date of access : 01.12.2020.
5. Краснопрошин, В. В. Исследование операций : учебное пособие / В. В. Краснопрошин, Н. А. Лепешинский – Минск : БГУ, 2013. – 191 с.
6. Головко, В. А. Нейросетевые технологии обработки данных : учебное пособие / В. А. Головко, В. В. Краснопрошин. – Минск : БГУ, 2017. – 264 с.
7. Unsupervised log message anomaly detection [Electronic resource]. – Mode of access : https://www.sciencedirect.com/science/article/pii/S2405959520300643. – Date of access : 30.11.2020.
8. HDFS Architecture Guide [Electronic resource]. – Mode of access : https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. – Date of access : 01.12.2020.
9. Apache Hadoop [Electronic resource]. – Mode of access : https://hadoop.apache.org. – Date of access : 25.10.2020.
10. A large collection of system log datasets for AI-powered log analytics [Electronic resource]. – Mode of access : https://github.com/logpai/loghub. – Date of access : 20.10.2020.

References

1. Vissiya, X. E. R. M. Prinyatie reshenij v informacionnom obshchestve: uchebnoe posobie / X. E. R. M. Vissiya, V. V. Krasnoproshin, A. N. Val'vachev. – Sankt-Peterburg : Lan', 2019. – 228 s.
2. Krasnoproshin, V. Solution of applied problems: formalization, methodology and justification / V. Krasnoproshin, V. Obraztsov, H. Vissia // World Scientific Proceeding Series on Computer Engineering and Information Science. – Vol. 3 : Computational Intelligence in Business and Economics. – London : World Sci-entific, 2010. – R. 57–64.
3. Distributed Representation of Word and Phrases and their Compositionality [Electronic resource]. – Mode of access : https://papers.nips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf. – Date of access : 30.11.2020.
4. Detecting Near-Duplicates for Web Crawling [Electronic resource]. – Mode of access : https://static.googleusercontent.com/media/ research.google.com/ru//pubs/archive/33026.pdf. – Date of access : 01.12.2020.
5. Krasnoproshin, V. V. Issledovanie operacij : uchebnoe posobie / V. V. Krasnoproshin, N. A. Lepeshinskij – Minsk : BGU, 2013. – 191 s.
6. Golovko, V. A. Nejrosetevye tekhnologii obrabotki dannyh : uchebnoe posobie / V. A. Golovko, V. V. Krasnoproshin. – Minsk : BGU, 2017. – 264 s.
7. Unsupervised log message anomaly detection [Electronic resource]. – Mode of access : https://www.sciencedirect.com/science/article/pii/S2405959520300643. – Date of access : 30.11.2020.
8. HDFS Architecture Guide [Electronic resource]. – Mode of access : https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html. – Date of access : 01.12.2020.
9. Apache Hadoop [Electronic resource]. – Mode of access : https://hadoop.apache.org. – Date of access : 25.10.2020.
10. A large collection of system log datasets for AI-powered log analytics [Electronic resource]. – Mode of access : https://github.com/logpai/loghub. – Date of access : 20.10.2020.

Материал поступил в редакцию 17.12.2020