УДК 004.031
DOI: 10.17586/0021-3454-2024-67-4-345-351

# МЕТОДЫ БАЛАНСИРОВКИ НАГРУЗКИ В ИНФОРМАЦИОННЫХ СИСТЕМАХ

Е. Д. Архипцев*, Н. С. Мокрецов

*Санкт-Петербургский государственный электротехнический университет „ЛЭТИ“ им. В.И. Ульянова (Ленина), Санкт-Петербург, Россия*
**lokargenia@gmail.com*

**Аннотация.** Обсуждается проблема балансировки нагрузки в крупных информационных системах. В условиях высокой нагрузки информационных систем, через которые проходят большие объемы данных, и увеличения числа пользователей, становится критически важным эффективное распределение нагрузки между ресурсами системы. Рассматриваются алгоритмы балансировки нагрузки, функционирование которых зависит от подхода (централизованного или распределенного) к построению архитектуры информационной системы. Описаны разные типы архитектуры построения информационных систем с выделением особенностей механизмов балансировки нагрузки для них. Демонстрируются результаты натурного эксперимента на виртуальных серверах по оценке эффективности алгоритмов балансировки нагрузки. Вычислительные характеристики серверов задавались разными и ожидаемое время выполнения запроса разыгрывалось случайным образом, что делает условия эксперимента близкими к реальным.

***Ключевые слова:*** *высоконагруженная система, балансировка нагрузки, архитектура системы, распределение ресурсов, алгоритмы балансировки, натурный эксперимент*

**Ссылка для цитирования:** *Архипцев Е. Д., Мокрецов Н. С.* Методы балансировки нагрузки в информационных системах // Изв. вузов. Приборостроение. 2024. Т. 67, № 4. С. 345—351. DOI: 10.17586/0021-3454-2024-67-4-345-351.

## METHODS OF LOAD BALANCING IN HIGHLY LOADED SYSTEMS

**E. D. Arkhiptsev*, N. S. Mokretsov**

*St. Petersburg Electrotechnical University, St. Petersburg, Russia*
**lokargenia@gmail.com*

**Abstract.** The problem of load balancing in large information systems is discussed. In conditions of high load of information systems caused by big data and an increase in the number of users, effective distribution of the load between system resources becomes critical. Existing load balancing algorithms are considered, considering a centralized or distributed approach to building the architecture of an information system. A description of different architectures for building information systems is given, highlighting the features of load balancing mechanisms for them. The results of a full-scale experiment on virtual servers to evaluate the effectiveness of load balancing algorithms are presented. The computing characteristics of the servers were set differently and the expected request execution time was played out randomly, which makes the experiment close to real conditions.

**Keywords:** highly loaded system, load balancing, system architecture, resource allocation, balancing algorithms, full-scale experiment

**For citation:** Arkhiptsev E. D., Mokretsov N. S. Methods of load balancing in highly loaded systems. *Journal of Instrument Engineering.* 2024. Vol. 67, N 4. P. 345—351 (in Russian). DOI: 10.17586/0021-3454-2024-67-4-345-351.

**Введение.** Важным этапом разработки высоконагруженных информационных систем является выбор механизма балансировки нагрузки.

Основная цель балансировки нагрузки заключается в обеспечении гарантированной безотказности и быстродействия информационной системы, которая достигается решением следующих задач:

— эффективное использование вычислительных ресурсов;

— обеспечение гарантированной пропускной способности системы;

— обеспечение гарантированного времени отклика системы;

— предотвращение перегрузки вычислительных ресурсов.

Независимо от того, является ли высоконагруженная информационная система распределенной или централизованной, балансировка нагрузки необходима — механизм ее реализации будет соответствовать архитектуре системы. Среди основных типов архитектуры информационных систем можно выделить:

— монолитную архитектуру [1];

— сервисно-ориентированную архитектуру (SOA) [2];

— микросервисную архитектуру [3];

— событийно-ориентированную архитектуру (EDA) [4].

При монолитной архитектуре все части приложения, включая пользовательский интерфейс, бизнес-логику и слой данных, развертываются и запускаются в одном процессе или на одном сервере [5].

При централизованном подходе один узел или группа узлов управляют передачей служебной информации, на основании которой происходит сбалансированное распределение нагрузки в системе. Недостаток такого подхода заключается в возможной перегрузке узла с централизованным принятием решения — так называемой точки отказа [6].

Остальные типы архитектуры — сервисно-ориентированная, микросервисная, событийно-ориентированная — представляют собой распределенные решения. Компоненты приложения при сервисно-ориентированной архитектуре представлены в виде автономных служб, взаимодействующих посредством стандартизированных протоколов. В случае микросервисной архитектуры приложение состоит из множества небольших автономных сервисов, каждый из которых решает конкретную бизнес-задачу. При событийно-ориентированной архитектуре компоненты системы могут обмениваться событиями для управления и обрабатывания изменений в системе [7—10].

При использовании распределенного подхода все участвующие в реализации функций балансировки нагрузки узлы обмениваются служебной информацией, на основании которой принимается решение с учетом собственных ресурсов узла [11].

Балансировщики нагрузки необходимы крупным компаниям, таким как банки, страховые и торговые корпорации, телекоммуникационные компании (системы биллинга, хостинга, всевозможные веб-сервисы и социальные сервисы). Эти компании используют сложные бизнес-приложения (ERP, CRM-системы и др.), и их деятельность напрямую зависит от надежности инфраструктуры [12, 13].

В настоящей статье с помощью натурного эксперимента оценивается эффективность известных алгоритмов балансировки нагрузки.

**Алгоритмы балансировки нагрузки.** Каждая архитектура информационной системы использует собственный балансировщик нагрузки. Определяется список доступных для балансировщика нагрузки серверов, которым при установке назначаются уникальные идентификаторы процесса (PID) [14].

Пусть $L = \{p_1, p_2, ..., p_n\}$ является списком из $n$ серверов, где $p_i$ — $i$-й сервер. Когда запрос пользователя отправляется на обслуживание в информационную систему, алгоритм балансировки нагрузки выбирает сервер (т.е. его PID), который будет использован для выполнения запроса. Во время работы выбор PID будет зависеть от алгоритма балансировки нагрузки [15]. Приведем известные алгоритмы балансировки нагрузки.

*Алгоритм случайного результата* (*Randomized*) [16]. Согласно алгоритму, новый запрос пользователя отправляется на случайно выбранный узел — сервер из имеющейся совокупности ресурсов. Таким образом, случайным образом выбирается натуральное число $x$ в диапазоне $[1, n]$ и $p_i = p_x$, где $p_i$ служит выходным значением (PID) сервера в алгоритме балансировки, а $p_x$ — элементом на позиции $x$ в списке $L$.

При случайном распределении нагрузки с течением времени ресурсы будут использоваться примерно одинаково, поскольку вероятность выбора каждого узла одинакова. Для алгоритма не требуется задавать какую-либо связь или извлекать информацию о состоянии серверов с целью балансирования нагрузки.

*Алгоритм циклического перебора* [17]. Согласно алгоритму, запрос отправляется следующему доступному узлу в очереди. При инициализации балансировщика выбирается случайное целочисленное значение $x$, которое определяет первоначальную балансировку нагрузки, а далее номер сервиса определяется по итерационному алгоритму.

Пусть $k$ — число запросов, прошедших через балансировщик нагрузки, $r$ — количество ожидающих запросов, тогда сначала определяется результирующее число балансировки $y = k + r$, а потом номер сервиса $i$ для обработки запроса $i = (y \bmod n) + 1$. Операция mod — нахождение остатка от деления — позволяет циклически выбирать элементы из списка $L$.

Циклический перебор не учитывает времени выполнения запроса, что приводит к повышению времени отклика, неэффективному использованию и распределению ресурсов. Эта проблема становится актуальной при различных значениях времени выполнения запроса или мощности серверов.

*Алгоритм min-max* [18] определяет минимальное время выполнения актуальных запросов и направляет запрос с наибольшим временем выполнения на узел с минимальным временем выполнения. Алгоритм основан на предположении, что сервер с меньшей рабочей нагрузкой имеет более высокую вероятность получения запроса с меньшим временем вычисления, чем сервер с более высокой рабочей нагрузкой. При выборе сервера балансировщик нагрузки вычисляет вероятность распределения зависимости от загруженности сервера и определяет PID из этого распределения

$$P(i) = \frac{a_i}{\sum_{i=1}^{n} a_i},$$

где $P(i)$ — вероятность выбора сервера $p_i$; $a_i = 1 - w(p_i)$ — весовой коэффициент распределения нагрузки $i$-го узла; $w(p_i)$ — рабочая нагрузка сервера $p_i$.

**Обсуждение результатов.** Для экспериментального сравнения алгоритмов балансировки нагрузки была развернута система, состоящая из трех виртуальных серверов с разной производительностью.

Любой запрос пользователя сначала направляется на балансировщик нагрузки — программный контроллер, предназначенный для последовательного выполнения следующих задач:

1) прием запросов от клиентов;
2) выбор сервера для обработки запроса клиента;
3) перенаправление запроса на выбранный сервер;
4) прием ответа от сервера на клиентский запрос;
5) перенаправление ответа сервера на запрос клиенту
6) мониторинг загруженности серверов информационной системы.

Каждый запрос представляет собой сессию пользователя, которая состоит из последовательности операций, необходимых для его выполнения. Поэтому время выполнения запросов задавалось случайным образом в виде функции распределения или функции плотности вероятностей и оценивалось как период между моментом поступления запроса на вход балансировщика нагрузки и моментом появления ответа на этот запрос на выходе сервера.

Сессия пользователя информационной системы может быть представлена временной диаграммой, отражающей выполнение задания запроса на сервере (рис. 1). Диаграмма состоит из периодов, когда вычислительные ресурсы сервера заняты выполнением задания и

интервалов времени, в течение которых процесс находится в состоянии ожидания освобождения ресурса.

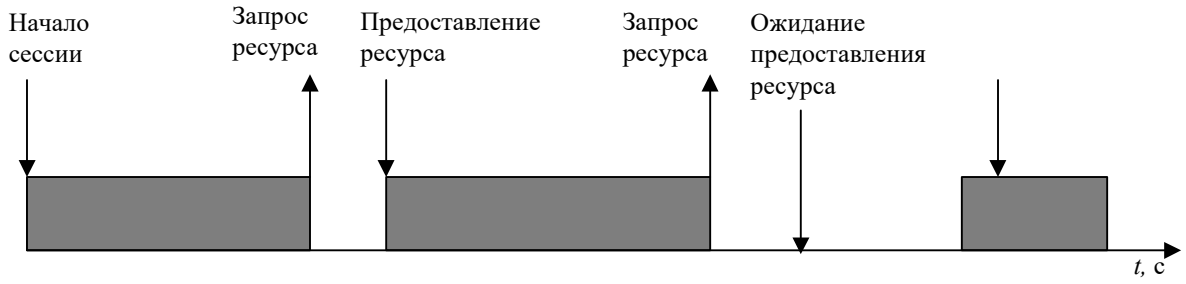


*Рис. 1*

На рис. 2 приведены результаты балансировки нагрузки по рассмотренным выше алгоритмам (*d* — число запросов, отклоненных серверами из-за большой нагрузки; производительность наименее мощного сервера оценивалась в 20 баллов, самого мощного — в 30 баллов).

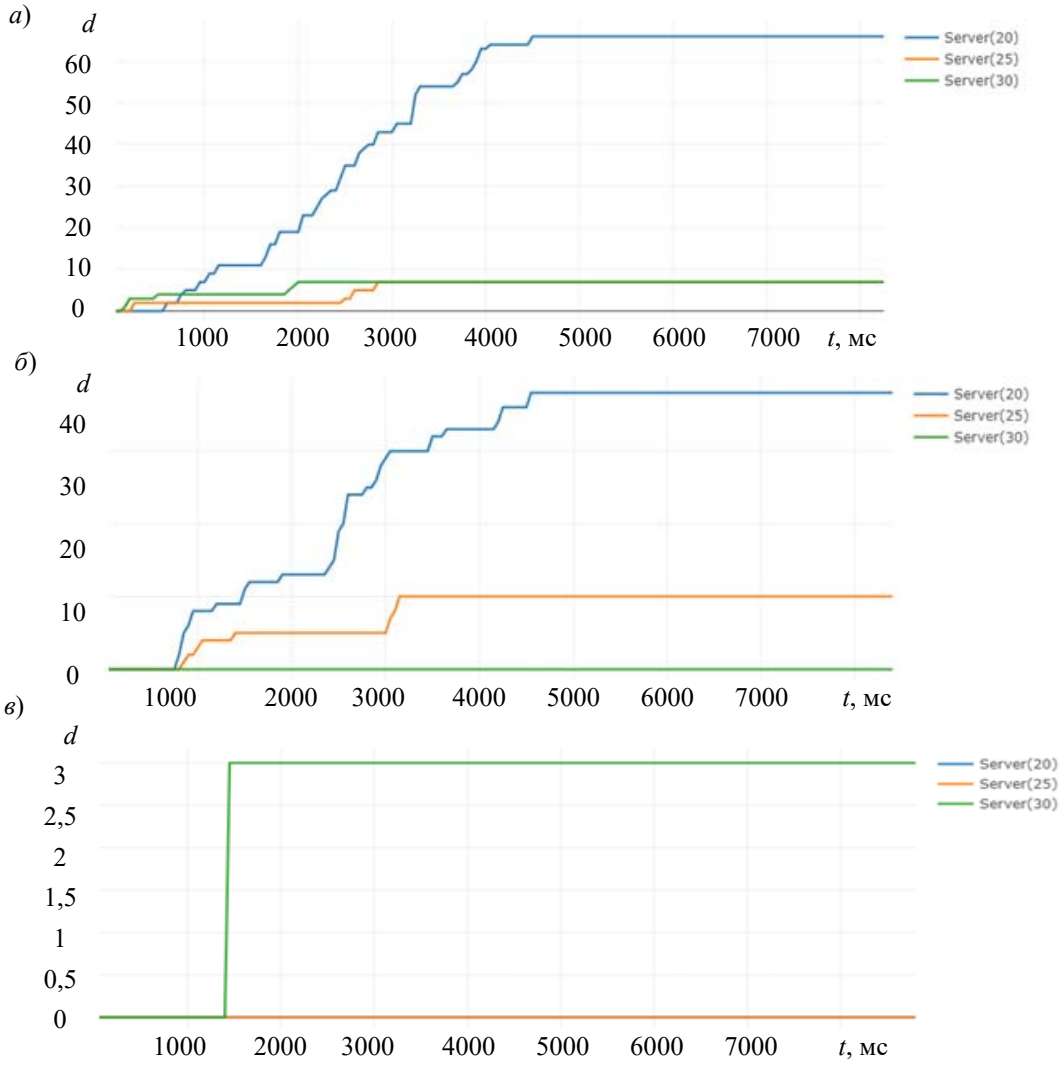


*Рис. 2*

На рис. 2, *а*, где проиллюстрирована балансировка нагрузки по алгоритму случайного результата, первый сервер — низкопроизводительный (синяя кривая) — показал ожидаемый результат. Однако второй сервер (оранжевая кривая) лучше справлялся с нагрузкой, по сравнению с самым мощным сервером (зеленая кривая). Относительно результатов эксперимента балансировки нагрузки по алгоритму случайного результата можно сделать следующий вывод: несмотря на незначительное время вычисления PID сервера, выбранного для обработ-

ки запроса, этот алгоритм малоэффективен, когда различается производительность серверов и запросы требуют разного времени выполнения.

Из графика для балансировки нагрузки по алгоритму циклического перебора (рис. 2, *б*) видно, что число отказов зависит от мощности сервера. Однако этот алгоритм не учитывает время, затраченное на выполнение запросов, что приводит к простою мощного сервера и отказу в обработке запросов на низкопроизводительном сервере.

Как видно из рис. 2, *в*, при балансировке нагрузки с применением алгоритма min-max число отброшенных запросов выровнялось на каждом сервере, а их общее число уменьшилось. Поскольку алгоритм учитывает текущее состояние серверов, это позволило равномерно распределять нагрузку по серверам.

**Заключение.** Рассмотрены типы архитектуры информационных систем и методы балансировки нагрузки, применяемые для распределения запросов пользователей по серверам системы с учетом требований к времени обработки и потерям.

Приведены особенности основных алгоритмов распределения нагрузки, таких как случайный результат, циклический перебор и min-max. По результатам натурного эксперимента на виртуальных серверах сделаны следующие выводы по эффективности алгоритмов балансировки нагрузки:

— алгоритм на базе случайного результата обеспечивает наименьшее время принятия решения о распределении нагрузки;

— алгоритм min-max показывает самое долгое время принятия решения о распределении нагрузки, поскольку решение зависит от состояния каждого сервиса системы;

— алгоритмы циклического перебора и случайного результата не учитывают текущей загруженности серверов и с увеличением нагрузки запросы пользователя начинают получать отказы в обслуживании;

— в условиях, близких к реальным, при которых различается производительность серверов и для обслуживания запросов требуется разное время обработки, алгоритмы случайного результата и циклического перебора оказались неэффективны;

— алгоритм min-max, благодаря своим особенностям вычисления загруженности серверов, справляется с балансировкой нагрузки лучше алгоритмов случайного результата и циклического перебора.

## СПИСОК ЛИТЕРАТУРЫ

1. *Амиров С. Н.* Особенности разработки выоконагруженных систем // International Journal of Open Information Technologies. 2020. Т. 8, № 8. С. 32—47.

2. *Подольный В. П.* Архитектура высоконагруженных систем. Системы сбора информации, распределенные системы управления, системы реального времени. М.: САМ Полиграфист, 2022. 160 с.

3. *Мычко С. И.* Микросервисная архитектура // Информационные технологии. 2019. С. 166—168.

4. *Беллемар А.* Создание событийно-управляемых микросервисов. СПб: BHV, 2022. 320 с.

5. *Радостаев Д. К., Никитина Е. Ю.* Стратегия миграции программного кода из монолитной архитектуры в микросервисы // Вестник Пермского университета. Математика. Механика. Информатика. 2021. Вып. 2(53). С. 65—68. DOI: 10.17072/1993-0550-2021-2-65-68.

6. *Мартин Р.* Чистая архитектура. Искусство разработки программного обеспечения. СПб: Питер, 2018. 352 с.

7. *Богатырев В. А., Лисичкин Д. Э.* Оптимизация периодичности инициализации контроля на основе дублированных вычислений // Программные продукты и системы. 2019. № 2. С. 214—220. DOI: 10.15827/0236-235X.126.214-220.

8. *Сергеева И. И., Белильщикова А. А.* Сервисно-ориентированная архитектура (SOA): Опыт внедрения // Научные Записки ОрелГИЭТ. 2012. № 1. С. 440—444.

9. *Татарникова Т. М., Архипцев Е. Д., Кармановский Н. С.* Определение размера кластера и числа реплик высоконагруженных информационных систем // Изв. вузов. Приборостроение. 2023. Т. 66, № 8. С. 646—651. DOI: 10.17586/0021-3454-2023-66-8-646-651.

10. *Ньюмен С.* Создание микросервисов. СПб: Питер, 2018. 304 с.

11. *Татарникова Т. М., Архипцев Е. Д.* Алгоритм контроллера нечеткой логики для размещения файлов в системе хранения данных // Науч.-техн. вестн. информационных технологий, механики и оптики. 2023. Т. 23, № 6. С. 1171—1177. DOI: 10.17586/2226-1494-2023-23-6-1171-1177.

12. *Бороздин Н. М.* Исследование и анализ практических преимуществ микросервисной архитектуры для современных веб-приложений // Инновационные научные исследования в современном мире. 2023. С. 279—284.

13. *Татарникова Т. М., Архипцев Е. Д.* Определение числа реплик распределенного хранения больших данных // Междунар. конф. по мягким вычислениям и измерениям. 2023. Т. 1. С. 305—308.

14. *Malavika R., Valarmathi M. L.* Adaptive Server Load Balancing in SDN Using PID Neural Network Controller // Computer Systems Science & Engineering. 2022. Vol. 42, N 1. P. 229—243. DOI: 10.32604/csse.2022.020947.

15. *Zagarese Q.* et al. Improving data-intensive EDA performance with annotation-driven laziness // Science of Computer Programming. 2015. Vol. 97. P. 266—279. DOI: 10.1016/j.scico.2014.03.007.

16. *Mitzenmacher M.* The power of two choices in randomized load balancing // IEEE Transact. on Parallel and Distributed Systems. 2001. Vol. 12, N 10. P. 1094—1104. DOI: 10.1109/71.963420.

17. *Kaur S.* et al. Round-robin based load balancing in Software Defined Networking // 2nd Intern. Conf. on Computing for Sustainable Global Development (INDIACom). New Delhi, India, 2015. P. 2136—2139.

18. *Maqsood Z. S. Kh., Ali T., Bilal M., Madani K., Khan S., ur Rehman A.* A Load Balanced Task Scheduling Heuristic for Large-Scale Computing Systems // Computer Systems Science and Engineering. 2019. N 1. P. 1—12. DOI: 10.32604/csse.2019.34.079.

***Сведения об авторах***

***Евгений Дмитриевич Архипцев*** — аспирант; Санкт-Петербургский государственный электротехнический университет „ЛЭТИ“ им. В.И. Ульянова (Ленина), кафедра информационных систем; E-mail: lokargenia@gmail.com

***Никита Сергеевич Мокрецов*** — аспирант; Санкт-Петербургский государственный электротехнический университет „ЛЭТИ“ им. В.И. Ульянова (Ленина), кафедра информационных систем; E-mail: nikitamokrecov6374@gmail.com

Поступила в редакцию 04.12.23; одобрена после рецензирования 08.12.23; принята к публикации 08.02.24.

**REFERENCES**

1. Amirov S.N. *International Journal of Open Information Technologies*, 2020, no. 8(8), pp. 32–47. (in Russ.)
2. Podolny V.P. *Arkhitektura vysokonagruzhennykh sistem. Sistemy sbora informatsii, raspredelennyye sistemy upravleniya, sistemy real'nogo vremeni* (Architecture of Highly Loaded Systems. Information Collection Systems, Distributed Control Systems, Real-Time Systems), Moscow, 2022, 160 p. (in Russ.)
3. Mychko S.I. *Information Technology*, 2019, pp. 166–168. (in Russ.)
4. Bellemare A. *Building Event-Driven Microservices*, O'Reilly Media, 2020, 324 p.
5. Radostev D.K., Nikitina E.Yu. *Bulletin of Perm University. Mathematics. Mechanics. Computer Science*, 2021, no. 2(53), pp. 65–68, DOI: 10.17072/1993-0550-2021-2-65-68. (in Russ.)
6. Martin R.C. *Clean Architecture. A Craftsman's Guide to Software Structure and Design*, 2017.
7. Bogatyrev V.A., Lisichkin D.E. *Software & Systems*, 2019, no. 2, pp. 214–220. DOI: 10.15827/0236-235X.126.214-220. (in Russ.)
8. Sergeeva I.I., Belilshchikova A.A. *Scientific Notes of the Oryol State Institute of Economics and Trade*, 2012, no. 1, pp. 440–444. (in Russ.)
9. Tatarnikova T.M., Arkhiptsev E.D., Karmanovskiy N.S. *Journal of Instrument Engineering*, 2023, no. 8(66), pp. 646–651, DOI: 10.17586/0021-3454-2023-66-8-646-651. (in Russ.)
10. Newman S. *Building Microservices*, O'Reilly, 2015.
11. Tatarnikova T.M., Arkhiptsev E.D. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2023, no. 6(23), pp. 1171–1177, DOI: 10.17586/2226-1494-2023-23-6-1171-1177. (in Russ.)
12. Borozdin N.M. *Innovatsionnyye nauchnyye issledovaniya v sovremennom mire* (Innovative Scientific Research in the Modern World), Materials of the XIII International Scientific and Practical Conference, Ufa, 2023, pp. 279–284. (in Russ.)
13. Tatarnikova T.M., Arkhiptsev E.D. *Mezhdunarodnaya konferentsiya po myagkim vychisleniyam i izmereniyam* (Inter-

national Conference on Soft Computing and Measurement), 2023, vol. 1, pp. 305–308. (in Russ.)
14. Malavika R., Valarmathi M.L. *Computer Systems Science & Engineering*, 2022, no. 1(42), pp. 229–243, DOI: 10.32604/csse.2022.020947.
15. Zagarese Q. et al. *Science of Computer Programming*, 2015, vol. 97, pp. 266–279, DOI: 10.1016/j.scico.2014.03.007.
16. Mitzenmacher M. *IEEE Transactions on Parallel and Distributed Systems*, 2001, no. 10(12), pp. 1094–1104, DOI: 10.1109/71.963420.
17. Kaur S. et al. *2nd Intern. Conf. on Computing for Sustainable Global Development (INDIACom)*, New Delhi, India, 2015, pp. 2136–2139.
18. Maqsood Z.S.Kh., Ali T., Bilal M., Madani K., Khan S., ur Rehman A. *Computer Systems Science and Engineering*, 2019, no. 1, pp. 1–12, DOI: 10.32604/csse.2019.34.079.

***Data on authors***

***Evgeny D. Arkhiptsev*** — Post-Graduate Student; St. Petersburg Electrotechnical University, Department of Information Systems; E-mail: lokargenia@gmail.com

***Nikita S. Mokretsov*** — Post-Graduate Student; St. Petersburg Electrotechnical University, Department of Information Systems; E-mail: nikitamokrecov6374@gmail.com

Received 04.12.23; approved after reviewing 08.12.23; accepted for publication 08.02.24.