UDC 004.9:004.4 DOI https://doi.org/10.35546/kntu2078-4481.2025.3.2.12

D. S. VOVCHENKO
Postgraduate Student at the Computer Systems Software Department
National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute"
ORCID: 0009-0008-1806-5159

L. M. OLESHCHENKO
Candidate of Technical Sciences,
Associate Professor at the Computer Systems Software Department
National Technical University of Ukraine
"Igor Sikorsky Kyiv Polytechnic Institute"
ORCID: 0000-0001-9908-7422

DEVELOPMENT OF A NODE BALANCING ALGORITHM USING MACHINE LEARNING AND DYNAMIC WEIGHT CALCULATION

The paper presents a method and software solution for node balancing in multi-node systems using machine learning techniques and dynamic weight computation. The relevance of this research is determined by the bottleneck effect that arises during big data processing, when one node becomes overloaded while others remain underutilized, resulting in inefficient resource usage. The aim of the research was to design a software system capable of real-time load distribution across nodes to maintain stable performance under high data volumes. The proposed algorithm combines an online machine learning model, which estimates query complexity based on input parameters (length, number of headers, type of operations, presence of JOIN clauses), with a dynamic weight calculation mechanism that regulates node load. A distinctive feature is the ability of the system to learn directly while processing requests, eliminating the need for prior data preparation and reducing deployment time. The research implemented the Python River library, which provides tools for online learning. To evaluate the efficiency of the algorithm, a simulated three-node system was tested with a dataset of 10,000 queries. The proposed method provides an average efficiency improvement of 2–5 % due to a more balanced load distribution between nodes compared to the baseline algorithms. The experimental results demonstrated that the proposed method achieved efficiency comparable to classical approaches (Round Robin, Random, Join Shortest Queue), while ensuring a more balanced query distribution due to complexity-aware processing. Comparative analysis confirmed that even under controlled conditions, the algorithm reached the performance level of baseline methods, whereas in real-world scenarios, with more heterogeneous request characteristics, its advantages are expected to be more significant. Future work will focus on improving testing conditions, deploying the algorithm in real distributed environments, and decoupling the machine learning logic from the overall architecture to enhance flexibility and scalability.

Key words: *software, node balancing algorithms, multi-node systems, Python, online machine learning model.*

Д. С. ВОВЧЕНКО
аспірант кафедри програмного забезпечення комп'ютерних систем
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
ORCID: 0009-0008-1806-5159

Л. М. ОЛЕЩЕНКО
кандидат технічних наук,
доцент кафедри програмного забезпечення комп'ютерних систем
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
ORCID: 0000-0001-9908-7422

РОЗРОБКА АЛГОРИТМУ БАЛАНСУВАННЯ ВУЗЛІВ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЙ МАШИННОГО НАВЧАННЯ ТА ДИНАМІЧНОГО ОБЧИСЛЕННЯ ВАГИ

У статті представлено метод та програмне забезпечення для балансування вузлів у багатовузлових системах з використанням технологій машинного навчання та динамічного обчислення ваги. Актуальність дослідження зумовлена проблемою виникнення «вузьких місць» при обробці великих даних, коли один із вузлів перевантажується,

тоді як інші залишаються недовантаженими, що призводить до неефективного використання ресурсів. Метою дослідження є створення програмної системи, здатної в режимі реального часу здійснювати ефективний розподіл навантаження між вузлами та забезпечувати стабільну продуктивність при високих обсягах даних. Запропонований алгоритм поєднує онлайн-модель машинного навчання, яка оцінює складність запитів на основі їх характеристик (таких як довжина, кількість заголовків, тип операцій, наявність JOIN у SQL-запитах), та механізм динамічного обчислення ваги для контролю рівня завантаження вузлів. Важливою особливістю є здатність системи навчатись безпосередньо під час обробки запитів, що усуває потребу у попередній підготовці даних та скорочує час розгортання. У процесі дослідження було використано Python-бібліотеку River, яка надає необхідні інструменти для реалізації онлайн-моделей. Для перевірки роботи запропонованого алгоритму було проведено експериментальне моделювання системи з трьома вузлами та тестування на наборі із 10 тисяч запитів. Запропонований метод у середньому забезпечує покращення ефективності на 2–5 % завдяки більш збалансованому розподілу навантаження між вузлами порівняно з базовими алгоритмами. Отримані результати показали, що запропонований метод демонструє ефективність, співставну з класичними алгоритмами (Round Robin, Random, Join Shortest Queue), однак забезпечує кращий розподіл запитів між вузлами завдяки врахуванню їх складності. Порівняльний аналіз підтвердив, що навіть у контрольованих умовах алгоритм здатен досягати рівня ефективності базових підходів, а в реальних сценаріях, де характер запитів значно варіюється, його переваги можуть виявитися більш суттєвими. Подальші дослідження спрямовані на удосконалення тестового середовища, інтеграцію алгоритму в реальні розподілені системи та відокремлення логіки машинного навчання від загальної архітектури для підвищення гнучкості та масштабованості.

Ключові слова: *програмне забезпечення, алгоритми балансування вузлів, системи з багатьма вузлами, Python, онлайн-модель машинного навчання.*

Problem statement

The amount of information on the Internet is constantly growing and at the same time the need for its timely processing is growing. There are many systems that pass through them large volumes of data every day, which are called big data. Due to such a flow, the question inevitably arises of faster processing of incoming information. There are quite a few ways to scale such systems, and the simplest of them would be to increase the resources involved in this process. This approach is simple, however, it has its drawbacks, such as the need for additional physical components to use these resources and their high cost. There are many opinions that the best option is to distribute the system into several nodes [1]. This approach is characterized by the fact that one large system is divided into smaller subsystems, called nodes, each of which is capable of processing the incoming flow of information. The number of these systems can be different and depends solely on the task at hand. With this approach, the incoming data flow is divided into groups, which are then transmitted to the appropriate nodes for further processing according to a certain principle. The correct distribution of this data between the nodes determines how efficiently the processing will be carried out.

If the data is incorrectly distributed, the so-called “bottleneck effect” occurs. It is characterized by the fact that one of the nodes is overloaded with the amount of data, while the others may not be used at all. This leads to uneven use of resources and delays in the functionality. To prevent this situation, appropriate algorithms are implemented in systems with several nodes, the purpose of which is to control the distribution itself. There are many ways to build these algorithms, starting from the simplest ones using basic Round-Robin algorithms or minimum connections, to more complex ones with dynamic weight calculation. The most modern versions of these algorithms can be called those that use machine learning (ML) when analyzing the input stream. Despite all the variety and number of proposed algorithms, discussions regarding their effectiveness do not stop [2] and it is still impossible to say which of these approaches is better.

Most modern algorithms have a large number of shortcomings that do not allow them to be fully called effective. Among such shortcomings are: lack of testing, ignoring certain system parameters and lack of emphasis on big data. The last of these three aspects becomes more relevant with the gradual increase in the volume of data. In this regard, the task was to develop a distribution algorithm that would take into account all the existing shortcomings of existing solutions and show its effectiveness in comparison with them.

The task of this research is to create a node balancing algorithm that will focus on processing big data. This goal is planned to be achieved through the use of existing methods of dynamic weight calculation and the application of machine learning technologies. The effectiveness of the created algorithm will be assessed by comparing such values as response speed, request processing frequency and average request processing time with existing algorithms.

Related research

By their logic, all node balancing algorithms can be divided into two subtypes: algorithms using additional hardware and algorithms based on software. Each of these two types has its advantages and disadvantages. For example, when using additional hardware, better efficiency is observed compared to software, however, this approach is more expensive and more complex.

Most balancing algorithms are based on a software approach precisely because of its simplicity and relative efficiency compared to cost [3]. That is why this research focuses on the software-based balancing algorithm.

By their structure, software-based algorithms can also be divided into two categories: static and dynamic. Static algorithms are simpler in their construction and involve deciding the order of data transmission to nodes before the system starts and do not have the ability to change this order during operation. Static algorithms include the basic Round Robin and Least Connections.

Dynamic algorithms are distinguished by the ability to change the order of distribution of input data to nodes, which is why they show greater efficiency than static algorithms when tested. One of the basic examples of such an approach is the algorithm with dynamic weight calculation based on minimal connections [4]. The principle of this algorithm is that a basic weight is assigned to each node of the system from the very beginning. Based on this value, the system decides how to adjust the flow of information to this node. This flow can be reduced if the node shows signs of overload, or vice versa increased if the node is not involved in anything. This weight can be dynamically changed during the operation of the system based on the obtained characteristics of the node itself. These characteristics can vary from one algorithm to another, however, among the most used ones are: load on the node, CPU usage, response speed, and number of connections. Depending on these values, the weight of each node increases or decreases.

Despite all this, it is worth noting that frequent weight changes are not a positive sign, since this can negatively affect the operation of the system itself. In this regard, in such algorithms there is a certain threshold value, and if it shows a slight change in the weight during the calculation, then changes will not be made and the system will continue to work in the previous configuration.

Using dynamic weight calculation, we can also control the characteristics of the system itself. For example, if during normal operation of the system the weight value of each node is always high, then based on this we can conclude that a new node needs to be added. The opposite statement is also valid, if the weight is always low, then to save resources we can remove one of the nodes.

An example of a more specific dynamic algorithm is an algorithm that uses the distance between the client and the nodes [5]. By itself, a system with several nodes is considered a single whole, however, some nodes may be physically located on resources that are more or less distant from the final recipient or client. The value of this distance also has an impact on the efficiency of the system. The difference of this algorithm from the previous one is manifested at the stage before the start of the entire system. At this moment, the distance value is calculated for each of the system nodes and based on it, an array of these nodes is created with the order of the distance from smaller to larger. The input data array is also divided into arrays with the number of requests inside equal to the number of nodes, after which the requests from this array are sent to the corresponding node that has an identical number in their array. After that, the principle of operation of the algorithm is similar to the previous one, during the work the weight value is calculated and the data transmission principle is corrected based on it. In addition to algorithms with different options for calculating the weight, there are also more complex approaches. For example, an algorithm that takes into account the estimated time of execution of requests [6]. This is achieved by using an online ML model. The advantage of this particular model is that, compared to others, the result of its processing does not need to be stored in memory, which is very important for the overall use of resources, especially in systems with many nodes.

The principle of operation of this algorithm is quite simple, the input data set is divided into groups, one of which is transmitted to the system itself, and the other goes to the ML model. This model, based on the data of the received requests and their purpose, calculates the approximate time of their execution. Knowing this time, the requests already fall on the appropriate node for them, which allows them to be processed efficiently.

Proposed algorithm

To solve the problems with the above algorithms, this article proposes to create a new node balancing algorithm using machine learning technologies and dynamic weight calculation.

The existing problems of existing methods include:

- lack of emphasis on big data;
- lack of testing;
- fixation only on certain resulting parameters.

The first problem does not require additional explanation, many algorithms were created decades ago at a time when the amount of data on the Internet was not large. This has led to the fact that such algorithms are not adapted to processing big data, they often exhibit the effect of a “bottleneck” when a queue of incoming requests is formed that cannot be processed on time. As a result, the request processing time, average throughput, and other efficiency parameters suffer.

For new algorithms, the problem with big data is not so relevant, however, they are also characterized by other shortcomings, the main of which is the lack of testing. The new proposed algorithms are not widely used and therefore their testing results are not as extensive as the old approaches. Also, recently, quite a few new possible algorithms with improved approaches have been proposed, however, their existence remains only theoretical and they are usually compared only with basic algorithms such as Round Robin and Least Connections, which does not give a complete picture of their effectiveness in real application.

In addition, these algorithms are characterized by an emphasis on limited parameters. This means that only some of the many resulting parameters are analyzed during testing. These parameters include: memory or CPU usage, system bandwidth, average request processing time, number of requests processed per unit of time, etc. In poor testing, only a few of these parameters are looked at and based on them, a conclusion is drawn about the effectiveness of the algorithm, while completely ignoring the others. In this regard, it is possible that the algorithm is quite efficient in terms of processing time and the number of input requests, but at the same time requires excessive memory usage.

To overcome these shortcomings, the proposed method plans to combine a machine learning algorithm and dynamic weight calculation. The use of machine learning technologies will help to improve the focus on big data, since in theory this will simplify the process of accepting the input data stream and allow the system to focus on its processing. The approach with dynamic weight calculation has long proven its effectiveness and many tests conducted on it gave positive results. The use of dynamic calculation will allow to maintain a balance in the resulting parameters in which the efficiency of some variables does not improve at the expense of the subsidence of others.

The schematic representation of the proposed algorithm is presented in Fig. 1.

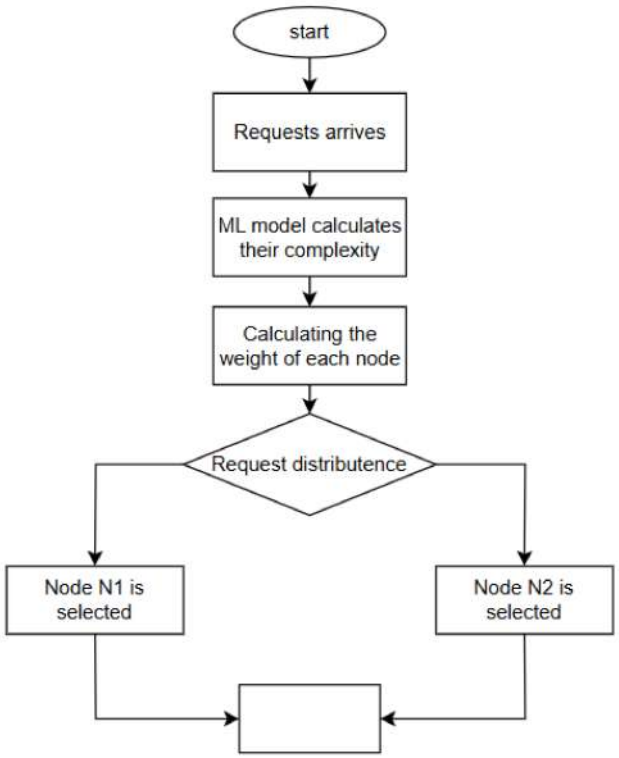

Fig. 1. Proposed algorithm workflow

The algorithm begins with the arrival of incoming requests, which are immediately processed by ML model to estimate their computational complexity. Based on this analysis, the system calculates the weight of each available node to reflect its current capacity and workload.

The requests are then distributed dynamically according to these calculated weights, ensuring that the load is balanced across the nodes. Depending on the results, a request is assigned to the most suitable node (e.g., Node N1 or Node N2), after which the process is repeated for subsequent requests, enabling continuous adaptation to workload fluctuations.

Research results

To implement the method, we first need to choose a programming language, and for this we should consider three options: C#, Java, and Python.

The main advantage of C# is its integration with Windows Server / Azure services, which provides better opportunities for deploying a multi-node system. This, in turn, will simplify the implementation itself and further testing and support. C# also has its own ML.NET machine learning library. The disadvantages of this language are platform binding and limited features of the built-in ML.NET. The C# language is quite dependent on its Windows Server / Azure infrastructure, and because of this, any implementation on other platforms is quite complicated [7], and the capabilities in the field of ML are less developed compared to Python.

Like C#, the Java language has advanced support for multi-node systems, however, the Java language allows focus on big data, this is achieved thanks to the existing Hadoop, Spark, Kafka, ZooKeeper frameworks. Java has support

for ML libraries, however, their limitations are the main disadvantage when choosing this language. The undoubted advantage of the Python language is its integrated libraries for working with machine learning. It provides the widest range of capabilities when creating and testing the developed model. Unlike the other two languages, the possibility of implementing and supporting a system with many nodes is less limited, but still present. The disadvantage is that, apart from the ML model, the language is not suitable for large systems.

None of the proposed languages can satisfy all the conditions of the proposed method, and therefore choosing only one is not optimal. In this regard, for the full implementation of the algorithm, it is planned to divide it into two components: the ML model itself and the balancer between nodes. Since the Python language has the best conditions for implementing the model, it is on it that the model will be built. To implement the balancer, it is worth using one of the two languages, C# or Java. In this article, attention is focused exclusively on the ML model, so the issue of implementing the balancer will not be considered in detail.

As mentioned, within the framework of this article, a ML model was developed in Python. This was achieved thanks to the built-in river library, which has all the necessary tools for implementing a ML model. In its structure, the algorithm uses an online learning model, which allows optimizing the learning process itself, since it does not require additional preparation time before starting work, unlike offline models. The choice of an online ML model is due to previous research in this field [8]. The principle of operation of the model is based on the expected processing time of the input query. To predict it, the model receives the necessary parameters of the query itself, such as its length, number of headers, type of operation, presence of JOIN in database queries, etc. Based on this information, the model selects a suitable node from the available ones, transmits the query to it, and at the same time uses its actual execution time for further training. Thanks to this, the system can simultaneously process the input stream and learn without additional preparation.

To test proposed algorithm, a three-node system has been simulated and a ten-thousand test set of various input queries has been created. The results of the created algorithm and the results of the basic algorithms are presented in the Table 1.

Table 1

Execution time comparison of baseline and proposed algorithms

Algorithm	Median query execution time	Execution time of the longest queries
Round Robin	195ms	728ms
Random	195ms	730ms
Join Shortest Queue	206ms	768ms
Proposed algorithm	206ms	763ms

The proposed algorithm provides an average efficiency improvement of 2–5 % due to a more balanced load distribution between nodes compared to the baseline algorithms. The median query execution time means that half of the queries were executed within this time, and the execution time of the longest queries was calculated based on the 5 % of the longest queries. According to this result, the basic RoundRobin algorithm showed the best performance, and the usual random distribution is not far behind and at first glance it may seem that the developed algorithm is not efficient, but this is not the case and there are several factors that indicate this.

First of all, the tests were conducted with a simulated data set on an emulated system with many nodes, that is, the entire infrastructure is controlled and unpredictable circumstances cannot arise in it. Under such conditions, the efficiency of the basic algorithms is always high, since they do not require additional calculations and can provide an instant result. Even under such conditions, the developed algorithm showed that it is able to achieve the level of efficiency of the basic ones. Under real conditions, the result of the algorithm may differ significantly from the current one.

Secondly, it is worth paying attention to the statistics of the distribution of queries between nodes. The best algorithm RoundRobin in terms of time showed an equal distribution of 3334, 3333 and 3333 requests for the first, second and third node respectively. Under real conditions, such a distribution can lead to delays on one of the nodes because it received longer requests compared to the others, which will negatively affect the overall performance of the entire system. When distributed by the developed algorithm, the first, second and third nodes received the following number of requests: 3356, 3256 and 3387, respectively. The reason for the discrepancy is that this algorithm took into account the objective duration of requests and the current queue on the nodes, which led to a more correct distribution. In general, the test results showed an equivalent efficiency of the algorithm compared to the baseline under simulated conditions, which in the future, when transferred to real conditions, may show the efficiency of the algorithm, since the baseline approaches have many shortcomings.

In Ukraine, where many sectors such as transportation, healthcare, energy, defense, and public administration are increasingly adopting digital platforms, the ability to process large amounts of information in real time without failures or bottlenecks is vital for efficiency and national competitiveness. For instance, in public transport systems or energy networks, even minor optimization of query distribution can lead to faster decision-making, more accurate forecasting, and better user experience, while in government digital services such improvements directly influence citizens' trust

and satisfaction. Since Ukrainian enterprises and institutions often operate under resource constraints, the ability of the algorithm to adaptively account for the complexity of incoming requests and dynamically redistribute loads offers a more sustainable and cost-effective alternative to simply scaling hardware infrastructure.

Conclusions and future work

Multi-node systems are becoming a common solution for optimal computing of large amounts of data, however, despite the availability of many solutions, they still have pressing problems with efficient query processing and working with big data. Three software-based distribution algorithms were considered. Most of them achieve a certain efficiency by changing the formula for calculating the weight of each node. The difference lies in the parameters used, such as the number of minimum connections or the physical distance to the node.

An algorithm was also considered that uses a machine learning model to predict the expected duration of an incoming request, based on which it was transferred to the appropriate node. Based on the algorithms studied, it was proposed to develop an own method that combines an online machine learning model with dynamic calculation of node weights. A prototype of this method was developed and tested in simulated conditions. As a result, the algorithm showed similar results to the basic approaches of the RoundRobin approach, which, due to the conditions, can speak about its effectiveness in a real experiment.

As a further work, it is planned to supplement the algorithm by improving the process of dynamic weight calculation. Also, for this algorithm, testing is planned in real conditions using a full system with many nodes and a reliable data set. Analysis of the results of the algorithm on such an infrastructure should show its effectiveness in comparison with the basic algorithms. The next step will be to compare the developed algorithm with modern options and analyze them when working with big data.

References

1. Hunter S. W., Smith W. E. (1999). Availability Modeling and Analysis of a Two Node Cluster. *5th Intl. Conference on Information Systems, Analysis and Synthesis*, Orlando, FL.

2. Yong Meng Teo, Ayani R. (2001). Comparison of Load Balancing Strategies on Cluster-based Web Servers. *Simulation*. 77 (5–6), pp. 185–195. https://doi.org/10.1177/00375497010770050

3. Belgaum M. R., Musa S., Alam M. M. and. Su'ud M. M. (2020). A Systematic Review of Load Balancing Techniques in Software-Defined Networking. *IEEE Access*, vol. 8, pp. 98612–98636. https://doi.org/10.1109/ACCESS.2020.2995849

4. Pan Z., Jiangxing Z. (2017). Load Balancing Algorithm for Web Server Based on Weighted Minimal Connections. *Journal of Web Systems and Applications*. Vol. 1. P. 1–8. DOI: https://dx.doi.org/10.23977/jwsa.2017.11001

5. Pei-rui J., Li-min M., Yu-zhou S., Yang-tian-xiu H. (2017). A client proximity based load balance algorithm in web sever cluster. *2nd International Conference on Wireless Communication and Network Engineering*. P. 317–322.

6. Omori M., Nishi H. (2018). Request Distribution for Heterogeneous Database Server Clusters with Processing Time Estimation. *International Conference on Industrial Informatics (INDIN)*, Porto. P. 278–283. DOI: 10.1109/INDIN.2018.8471931

7. Marcin Jamro. C# Data Structures and Algorithms. Second Edition. Published by Packt Publishing Ltd., in Birmingham, UK. 2024. – 349 p.

8. Okano H., Yamaguchi F., Takagiwa K., Nishi H. (2014). Traffic-based Flow Cache Port Separate Mechanism for Network processor. *The Institute of Electoronics, Information and Communication Engineers Technical Report*, vol. 114, no. 18, pp. 69–74.

Дата першого надходження рукопису до видання: 29.09.2025
Дата прийнятого до друку рукопису після рецензування: 24.10.2025
Дата публікації: 28.11.2025