UDC 004.657
DOI: 10.25140/2411-5363-2021-2(24)-75-82

Igor Boyarshin, Anna Doroshenko, Pavlo Rehida

REQUEST BALANCING METHOD FOR INCREASING THEIR PROCESSING EFFICIENCY WITH INFORMATION REPLICATION IN A DISTRIBUTED DATA STORAGE SYSTEM

The article describes a new method of improving efficiency of the systems that deal with storage and providing access of shared data of many users by utilizing replication. Existing methods of load balancing in data storage systems are described, namely RR and WRR. A new method of request balancing among multiple data storage nodes is proposed, that is able to adjust to input request stream intensity in real time and utilize disk space efficiently while doing so.

Keywords: *load balancing; data replication; scheduling.*
Fig.: 5. Table: 1. References: 7.

Urgency of the research. With increased popularity of remote computing and shared access to remote data, the overall load on the data storage nodes that contain this data increases accordingly. As a result, one way of improving the performance of the system as a whole is the use of additional storage nodes that replicate the data to which shared access is desired. This allows for distribution of user requests among these storage nodes, effectively increasing the overall system processing rate. The problem of user request distribution is solved with the help of a load balancer that employs a particular algorithm [1] in order to perform this distribution. This article describes one such load balancing algorithm that can be used in systems with multiple users and multiple date storage nodes.

Target setting. Load balancer acts as a single and main node that is responsible for the reception of all user requests that arrive into the system and their subsequent distribution among available data storage nodes, as well as solving the problem of optimal data placement among these nodes that would result in the maximum request processing intensity by the system. At the same time, as the amount of user data is substantial, a tradeoff between the total memory used on the data storage nodes and the resulting request processing intensity must be found. Efficient data placement among the storage nodes directly impacts the overall performance of the system. That is why the goal of the load balancing algorithm proposed in this article is twofold: on the one hand, to provide sufficient rate of user request processing, and on the other hand, to minimize the total memory used on all data storage nodes.

Analysis of existing research and publications. There is a number of algorithms used extensively that deal with efficient balancing of load in the system, namely Round Robin [2, 3] and Weighted Round Robin [4], that are both essentially an improvement of FCFS. There is also some research that concerns itself with different adaptations of Round Robin that additionally utilize information about the actual performance of different nodes: one such algorithm is Weighted Least Connections [5, 6]. Article [7] describes a modified load distribution planning prediction algorithm that is an improvement over Round Robin cyclic planning algorithm.

The Round Robin algorithm at its core uses the notion of a time quota that is constant. The algorithm allows the current process to run for exactly this time period, and if the process is unable to finish during this time, then it is placed to the end of the queue. In the constant of a load balancer the Round Robin algorithm chooses the storage node to assign a request to in a circular fashion and subsequently, thus each storage node performs an equal amount of work on average.

The Weighted Round Robin algorithm works on the same principal as the Round Robin, but it also considers the differences between the processed being executed. The processes are assigned different weights, where a larger weight denotes that this process shall be given a larger time quota to execute. This introduces the notion of process priority to the system. In the context of load balancing, the Weighted Round Robin algorithm distributes the requests among the storage nodes in such a way as to accomplish the same ratio of resulting processed request count as the ratio of corresponding weight coefficients of the storage nodes.

Uninvestigated parts of the general problem. Although there exist methods of load distribution in systems with constant request processing intensity, the approaches to load distribution for systems where the processing intensity can be varied and adjusted in real time with accordance to the need are poorly investigated.

Article objective. The objective of the article is to describe a new load balancing algorithm that is more optimal than known alternatives with regards to delivering a sufficiently high request processing intensity of the system, while reaching the lowest possible total memory consumption on all the data storage nodes. A detailed analysis of the performance of this algorithm compared to similar algorithms is conducted.

General overview of the system. The system that is modelled is comprised of m users $u_1..u_m$ that generate k requests $q_1..q_k$ that arrive to the load balancer, which is responsible for assigning them for processing to one of n data storage nodes $s_1..s_n$. The general system architecture is shown in Fig. 1.

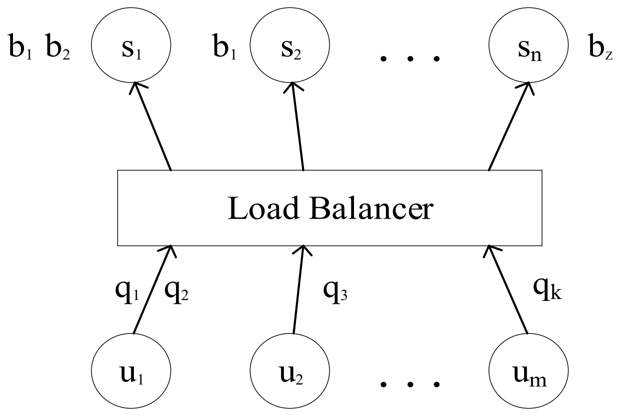

Fig. 1. General system architecture

The system stores z projects $B=\{b_1..b_z\}$ that are shared between users. A project represents a logical unit of data that the system operates with. Each project is stored on one or more data storage nodes. Users access these projects by the means of requests Q – each request q_i is an act of accessing a particular project b_j in order to work with it.

User model overview. Each user has a certain set of c projects that the user will work with $P=\{p_1..p_c\}$ that is a subset of all projects in the system: $P \subset B$. In order to create a more realistic simulation model each user is assigned one of the following behavior strategy models:

1. Random strategy. The project for request is chosen at random from set P of this user with equal probability of each project being selected.
2. Averaged sequential strategy. Each user works with each project from P for a particular time, while maintaining equal pick rate in the long term. A current project p_i as well as the repetition count for this project r are chosen at random. The next r requests from this user will be to project p_i. After r requests new values for p_i and r are generated.
3. Favorite project strategy. Each user has a favorite project p_k that is chosen at random from projects set P of this user. Project for each request is chosen at random, but with probability=0.7 the favorite project will be chosen.
4. Two favorite projects strategy. This strategy works the same way as the strategy with one favorite project, but instead of one there are two favorite projects p_k and p_m, with probability of either favorite project being selected equal to 0.35.

Data storage node overview. Each data storage node s_h stores a particular set of projects $W_h \subset P$ that can change over time. Storage node s_h is capable of processing user request q_i if and only if the project p_j of this request is stored on this node: $p_j \in W_h$, and also provided that this node is not busy processing another request at this time. Each node s_h has its own request processing time τ_h.

In order to measure the total memory consumption of the system on all storage nodes a new metric g is introduced. This metric is defined as the ratio between the total magnitude sum of sets W_h of all storage nodes and the total count of possible projects being stored in the system:

$$g = \sum_{i=1}^{n} \frac{W_i}{(z*n)}.$$

Load balancer model overview. As an input the load balancer takes a Poisson stream with intensity λ of requests $q_1..q_k$. The responsibilities of the load balancer include the following:

- Temporary storage of incoming requests in an internal queue.
- Selection of node s_h for the processing of request q_i to project p_m from internal request queue. This assumes that node s_h is free and contains data for project p_m: $p_m \in W_h$.
- Making decision about replication. The term «replication» here means an act of additional storing data for project p_m on node s_h, provided that this node did not contain data for this project prior to replication: $p_m \notin W_h$.
- Recording and storage of request processing statistics that is further used while making decisions about replication.

The principal job of the load balancer is to find an optimal placement of projects set B among data storage nodes S with potential replication in such a fashion as to reach a high enough processing intensity of requests Q by the system to avoid its overloading, while at the same time minimizing the total memory consumption on all data storage nodes S, which is directly proportional to the total sum of powers of sets W of nodes S, and can be measured using metric g.

Proposed load balancer algorithm. In order to temporary store incoming requests an internal queue is used that works in the following way: new requests are added to the end of the queue, while search for requests to be assigned to nodes S is done from the start of the queue. This way, maximum load of available data storage nodes is reached at each moment and also the maximum waiting time in queue is reduced, because requests with higher waiting time are located closer to the head of the queue, thus having a higher priority for assignment.

In order to make decision about replication the load balancer analyzes waiting time in queue to all the projects. The load balancer makes the decision about replication and an additional storage of project p_m on node s_h with current project set W_h if the following conditions are satisfied:

- Data storage node s_h does not contain data for this project yet: $p_m \notin W_h$.
- Last v waiting time deltas for project p_m are greater than zero (which implies continuous increase in waiting time in queue).
- The last replication for project p_m happened more than v/2 requests to project p_m ago. This condition is required to allow the system to react to the last change that happened (that manifests itself in changes in waiting time) before (maybe) performing subsequent replication.

In order to select node s_h to replicate project p_m to, a subset R of nodes that do not yet contain data for project p_m is constructed: $R \subset S$: $\forall\, r_i \in R$: $p_m \notin W_i$ of node r_i and $\forall\, s_i \in S$, $s_i \notin R$: $p_m \in W_i$ of node s_i. If the resulting set is empty ($R = \emptyset$), then subsequent replication of this project is not possible. Otherwise a node is selected from R: $r_i \in R$, that has the least value of metric *worktime*, which is calculated using formula (1):

$$\text{worktime}(r_i) = (\text{amount of processed requests on node } r_i) * \tau_i$$

The *worktime* metric for a node is defined as the product of the amount of requests processed on that node and the time τ_i it takes to process one request, and is a measure of total work done on that node. This way, between two nodes with equal *worktime*, the faster node will have a higher processed request count, as its τ is smaller. Consequently, close values of *worktime* signify approximately the same load of nodes. And so when performing a replication, the node with least *worktime* value is selected.

If request to project p_m arrives into the system for the first time, and so there is no node s_h, project set W_h of which contains project p_m, then this project is registered in the system using the same replication algorithm as described above.

System simulation. In order to determine the efficiency of the proposed load balancing algorithm, a program for simulating the described system was created using the Rust programming language. The system has been simulated with the following parameters:

- Total project count z=20;
- User count m=10;
- Data storage nodes count n=7 with their respective request processing times (in milliseconds) T={31, 41, 71, 95, 121, 131, 262};
- Total request count to simulate k=2048;
- Amount of projects that every user is interested in c=8;
- Amount of waiting time in queue deltas upon which the decision about replication is made v=6.

In order to define input request stream intensity λ, the maximum possible input processing intensity of the system μ is introduced. The maximum possible input processing intensity of the system is defined as the sum of all individual intensities of all the nodes:

$$\mu = \sum_{i=1}^{n} \mu_i = \sum_{i=1}^{n} \frac{1}{\tau_i}.$$

For the simulation the value of input request intensity was set to λ=0.85 * μ, which means that the input stream is a Poisson stream with intensity that comprises 85% of the maximum possible input processing intensity of the system.

Based on the conducted simulation the graphics of waiting time in queue for the whole system as well as for each project separately are constructed. Requests that led to replication are denoted with vertical red lines. Graphics for the mentioned above simulation parameters for waiting time in queue for the whole system as well as for the first project are depicted on figures 2 and 3, respectively.

The resulting value of total memory usage metric for this simulation was g=35/140=0.25.

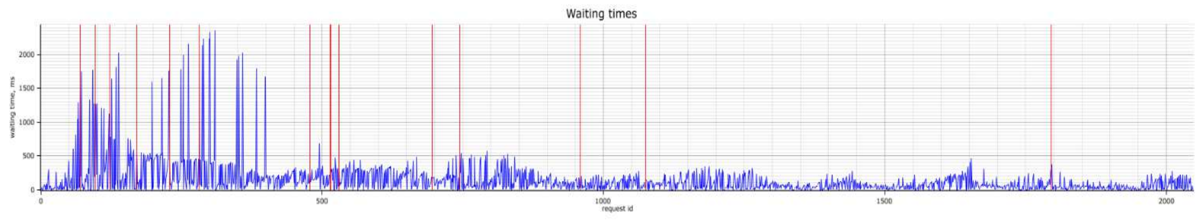

Fig. 2. Request waiting time in queue for the whole system

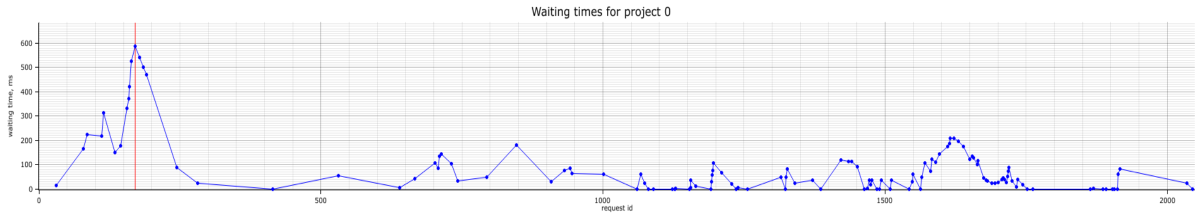

Fig. 3. Request waiting time in queue for the first project

As can be seen from the graphics, at first the waiting times vary significantly and are substantial, but as the system performs the necessary replication and adjusts to the input intensity, the values are stabilized and reduced, which implies that the input processing intensity of the system reaches the input request intensity.

As can be clearly seen from Fig. 3, the only replication for this project happened exactly after v=6 subsequent increases of its waiting time in queue. This replication led to a significant improvement in waiting time at once. It can also be seen that as there were no more v=6 subsequent increases in waiting time for this project, no more replication happened until the end of the simulation.

Comparison of proposed algorithm with other. In order to estimate the effectiveness of the proposed load balancer algorithm, it is tested against two other algorithms, namely Round Robin and Weighted Round Robin. As these two algorithms do not perform any replication and work with static values of request processing intensity, in order to be able to properly test them against the proposed algorithm, the simulations for the two algorithms is performed with fixed values of replication factor, which means that a fixed amount of replication is performed for them prior to the beginning of the simulation. The replication factor is defined as the amount of replication of the data for each project in the system as a whole. For example, the replication factor of 2 means that there are exactly two replication of the data of each project on some two distinct nodes in the system.

The Round Robin algorithm selects the node to assign the next incoming request for a particular project in a circular queue fashion among the nodes that contain data for this project. The Weighted Round Robin algorithm works the same way as Round Robin, but instead of uniform distribution of requests for each project among its data storage nodes, it does so in a weighted fashion, where the respective weights are the values of request processing intensities of each of the nodes.

The simulation results for different values of input request intensities for all three algorithms, as well as for different replication factors for Round Robin and Weighted Round Robin, are presented in table. The Bal denotes the proposed load balancer algorithm, the RR-2, RR-3 and WRR-2, WRR-3 denote the Round Robin and Weighted Round Robin algorithms with replication factors 2 and 3, respectively. For each of the algorithms the simulation is conducted with different values of input request intensity. In order to evaluate the performance of the algorithms the following metrics were selected: average waiting time in queue (denoted as *avg time*) and total memory consumption ratio (the *g* metric). As the amount of projects in the simulation *z* is equal 20, and the amount of data storage nodes *n* is equal 7, that means that the values of *g* for Round Robin and Weighted Round Robin with different values of replication factor are known beforehand and are constant.

Table – The simulation results

λ=0.25μ	Avg time, ms	g	λ=0.5μ	Avg time, ms	G	λ=0.75μ	Avg time, ms	g
Bal	8.0	24/140		12.7	28/140		27.5	41/140
RR-2	19.5	40/140		4273.5	40/140		21780.7	40/140
WRR-2	6.6	40/140		56.6	40/140		4947.5	40/140
RR-3	16.6	60/140		2191.8	60/140		39866.0	60/140
WRR-3	4.6	60/140		26.6	60/140		260.9	60/140
λ=0.9μ			λ=μ					
Bal	50.3	51/140		442.3	61/140			
RR-2	34181.7	40/140		41973.9	40/140			
WRR-2	12384.0	40/140		32626.9	40/140			
RR-3	39985.3	60/140		50843.5	60/140			
WRR-3	6733.8	60/140		14507.2	60/140			

As can be seen from the Table 1, for low input request intensity all the algorithms show approximately the same value of average waiting time in queue, the minimum being reached with the Weighed Round Robin algorithm with the maximum value of simulated replication factor of 3. Despite the fact that the proposed algorithm shows slightly worse resulting values of average waiting time in queue, it does so with almost twice as small replication factor, compared to other algorithms.

However, as the input request intensity is increased, the tendency is clear that the proposed algorithm is able to reach significantly lower values of waiting time in queue, compared to other algorithms, and it does so resulting in only marginally higher replication factor. As can be seen from the table, due to the better placement of projects among the data storage nodes, the proposed algorithm is able to reach a substantially better performance of the system as a whole, and it does so in dynamically in real time, adjusting to the given input request intensity.

Significant reduction of average request waiting time in queue for the proposed algorithm compared to others can be explained in the following way. While other algorithms have a fixed sequence of nodes that they assign incoming requests to, even if they do so considering the respective processing intensity of the nodes (WRR), the proposed algorithm makes decisions about assignment in real time based on the current workload of the nodes, which in turn prevents their excessive overloading as well as reduces their idle time.

In order to further investigate the features of the proposed algorithm, a simulation for it is performed with the same input parameters, but with two small modification of the base algorithm:

1. A simpler internal FIFO queue is used;
2. No replication is done.

The resulting waiting time in queue graphics are shown in figures 4 and 5, respectively.

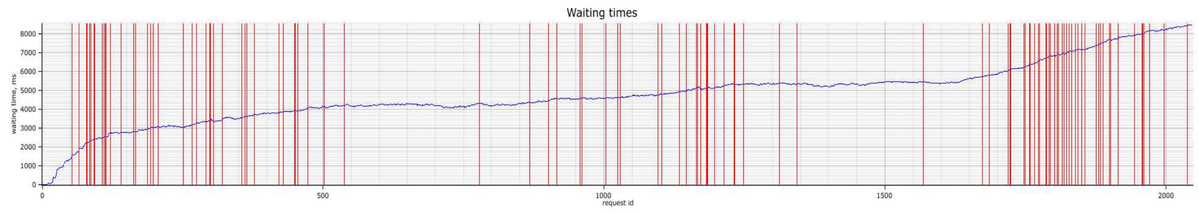

Fig 4. Waiting time in queue with FIFO type of queue

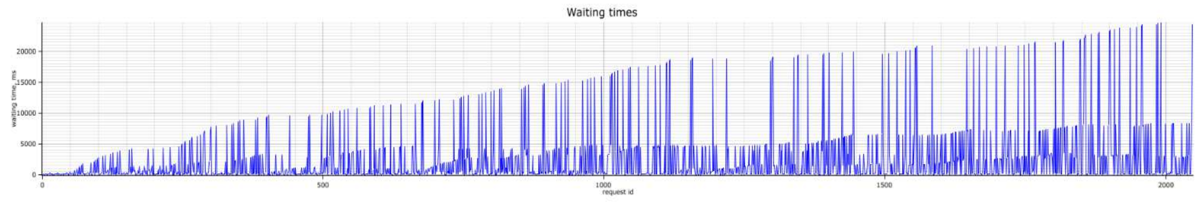

Fig 5. Waiting time in queue with no replication done

As can be seen from Fig. 4, the use of FIFO queue clearly stabilizes the waiting time of requests in queue, but the value itself continuously rises despite the large amount of replication that the system does, and the resulting request processing intensity of the system is still not high enough to process all the incoming requests, and thus the system is overloaded.

As can be seen from Fig. 5, the lack of replication leads to insufficient request processing intensity of the system, and as a result, the value of waiting time in queue rises continuously. Two distinct trends in waiting times can also be spotted: peaks values of waiting time, that correspond to more popular projects, and somewhat lower values, that correspond to less popular projects.

Conclusions. The article described and investigated a new method of increasing the efficiency of systems for data storage and processing requests of many users using replication. For this, a new load balancing algorithm was introduced, that distributes the incoming user requests among data storage nodes, adjusting to the intensity of the incoming requests in real time using replication in such a fashion as to reach an optimal disk utilization on data storage nodes.

A simulation of the described load balancing algorithm was performed, as well as its efficiency comparison with such of existing balancing algorithms, namely Round Robin and Weighted Round Robin. The proposed algorithm showed similar performance and used less disk space compared to other algorithms for low input request intensity, and significantly better performance for high input request intensity with only marginally higher disk utilization.

The proposed load balancing algorithm can be used in systems with many users for adjusting to input request intensity in real time in order to provide high efficiency and optimal disk space usage.

References

1. Comparing Load Balancing Algorithms (n.d.). https://www.jscape.com/blog/load-balancing-algorithms.
2. Hyytiä, E., & Aalto, S. (2016). On Round-Robin routing with FCFS and LCFS scheduling. *Performance Evaluation*, *97*, 83-103.
3. Ishwari, S.R., & Deepa, G. (2012). A priority based Round Robin CPU scheduling algorithm for real time systems. *International Journal of Innovations in Engineering and Technology (IJIET)*, 1(3), 11.
4. Wang, W., & Casale, G. (2014). Evaluating Weighted Round Robin load balancing for cloud web services. *Institute of Electrical and Electronics Engineers (IEEE)*, 393-400.
5. Choi, D., Chung, K.S., & Shon, J. (2010). An improvement on the weighted least-connection scheduling algorithm for load balancing in web cluster systems. *In Grid and distributed computing, control and automation* (pp. 127-134). Springer, Berlin, Heidelberg.
6. Singh, G., & Kaur, K. (2018). An improved weighted least connection scheduling algorithm for load balancing in web cluster systems. *International Research Journal of Engineering and Technology (IRJET),* 5(3), 6.
7. Khryshchenyuk, R.A. (2020). Modified method of load distribution in infocommunication networks [Master's thesis] [Khryshchenyuk Roman Andriyovych; Igor Sikorsky Kyiv Polytechnic Institute].

УДК 004.657

Ігор Бояршин, Анна Дорошенко, Павло Регіда

СПОСІБ БАЛАНСУВАННЯ ЗАПИТІВ ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ЇХ ОБРОБКИ НА ОСНОВІ ДУБЛЮВАННЯ ІНФОРМАЦІЇ В РОЗПОДІЛЕНІЙ СИСТЕМІ ЗБЕРІГАННЯ ДАНИХ

Зростання популярності віддалених обчислень та колективного віддаленого доступу до інформації призводить до значного збільшення навантаження на вузли зберігання даних, що забезпечують зберігання цієї інформації. Як наслідок, одним зі способів покращення продуктивності системи є використання додаткових вузлів зберігання даних, які дублюють інформацію, до якої відбувається колективний доступ. Це дозволяє розділити запити користувачів між декількома вузлами, ефективно збільшуючи загальну інтенсивність їх обробки. Запропонований алгоритм роботи балансувальника навантаження дозволяє вирішити задачу розподілення даних та запитів користувачів між вузлами в розподіленій системі зберігання даних.

Загальний об'єм даних користувачів на вузлах віддаленого зберігання даних та кількість запитів до них є значним, що призводить до зростання навантаження на систему. Тому необхідно знайти нові способи підвищення інтенсивності обробки запитів системою, беручи до уваги обмеженість дискового простору.

Наразі існують роботи присвячені аналізу методів розподілення навантаження у системі. У літературі описані такі методи, які широко використовуються на практиці, як Round Robin та Weighted Round Robin, які дозволяють більш ефективно розподілити навантаження у системі. Також існують роботи, які розглядають удосконалені варіанти алгоритму Round Robin, які враховують технічні характеристики та продуктивність серверів, серед яких такий алгоритм, як Weighted Least Connections.

Хоча існують методи розподілення навантаження в системах зі сталою інтенсивністю опрацювання запитів, методи для систем, в яких інтенсивність опрацювання може бути змінена для підлаштування до вхідної інтенсивності запитів у реальному часі є мало дослідженими.

Метою дослідження є створення оптимального алгоритму роботи балансувальника, що буде забезпечувати необхідну інтенсивність опрацювання запитів системою, і при цьому досягати мінімально можливого використання дискового простору на вузлах зберігання даних.

Описано метод балансування запитів між декількома вузлами зберігання даних, який забезпечує підлаштування під інтенсивність надходження запитів у реальному часі. Проведено симуляцію роботи системи та порівняння з існуючими алгоритмами балансування RR та WRR з різними параметрами, а також аналіз роботи системи з двома модифікаціями базового алгоритму.

Проведено аналіз та симуляція роботи запропонованого алгоритму роботи балансувальника навантаження та виконано порівняння ефективності його роботи з іншими поширеними алгоритмами балансування. Запропонований алгоритм показав близьку до інших алгоритмів ефективність роботи і використав менше дискового простору для низької інтенсивності потоку вхідних запитів, та значно кращу ефективність роботи для високої інтенсивності вхідного потоку з лише дещо більшим використанням дискового простору.

Ключові слова: *балансування навантаження; дублювання даних; планування.*

Рис.: 5. Табл.: 1. Бібл.: 7.

Boyarshin Igor – Student, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute” (37 Pobedy Av., 03056 Kyiv, Ukraine).
Бояршин Ігор Іванович – студент, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського» (просп. Перемоги, 37, м. Київ, 03056, Україна).
E-mail: igor.boyarshin@gmail.com
ORCID: http://orcid.org/0000-0002-1727-5305

Doroshenko Anna – Student, Department of Computer Engineering, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute” (37 Pobedy Av., 03056 Kyiv, Ukraine).
Дорошенко Анна Юріївна – студентка, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського» (просп. Перемоги, 37, м. Київ, 03056, Україна).
E-mail: annadoroshenko03@gmail.com
ORCID: http://orcid.org/0000-0001-7023-1961

Rehida Pavlo – Assistant, Department of Computer Engineering, National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute” (37 Pobedy Av., 03056 Kyiv, Ukraine).
Регіда Павло Геннадійович – асистент, кафедра обчислювальної техніки, Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського» (просп. Перемоги, 37, м. Київ, 03056, Україна).
E-mail: pavel.regida@gmail.com
ORCID: http://orcid.org/0000-0002-6591-7069
Scopus Author ID: 57202317133