Optimization of SQS Configurations for Efficient Batch Data Processing

OLEKSANDR O. KYRYCHENKO[1], SERGEY E. OSTAPOV[2], OKSANA L. KYRYCHENKO[1]
[1]Department of Mathematical Problems of Control and Cybernetics,
Yuriy Fedkovych Chernivtsi National University,
2 Kotsiubynskoho Street, Chernivtsi,
UKRAINE

[2]Department of Computer Systems Software,
Yuriy Fedkovych Chernivtsi National University,
2 Kotsiubynskoho Street, Chernivtsi,
UKRAINE

Abstract: - The evolution of distributed computing in cloud environments raises the question of finding new approaches to processing large amounts of data. The speed of data arrival and the need to make decisions in real-time adds to the higher complexity. AWS Simple Queue Service (SQS) is one of the popular tools for organizing asynchronous message processing in distributed systems in many scenarios. Relying on SQS with its default settings does not always work well, especially when we are dealing with heavy data processing. That is why figuring out the best setup for each specific task is a key challenge. *Objective*. This study aims to determine the queue setup so that a distributed system can efficiently handle processing 500,000 records while generating many PDF documents. We want to find the sweet spot in queue configuration that keeps the system running smoothly and effectively under heavy workloads. *Method*. The research evaluates the performance of the developed system for the mass generation of PDF documents under various load conditions using classical queuing theory models and their extensions. To assess the impact of different combinations of SQS parameters, key performance indicators such as response time, average queue length, and utilization rate are calculated using mathematical concepts. A PDF document generation software that directly interacts with SQS is developed using Python and AWS SDK Boto3. *Results*. The key factors affecting system performance are the batch size and time of message visibility in the queue. The results showed that proper configurations significantly increase throughput without loss of reliability. Empirical results confirm theoretical expectations and contribute to the selection of optimal parameters. *Conclusions*. The obtained research results enable us to provide practical recommendations for the selection of important parameters for SQS, such as throughput, delay, cost, and reliability, for performing high-load operations in serverless computing.

Key-Words: - serverless, cloud computing, queueing theory, batch processing, high-load systems, performance optimization.

Received: March 22, 2024. Revised: August 25, 2024. Accepted: December 9, 2024. Published: March 12, 2025.

1 Introduction

Amazon Simple Queue Service (Amazon SQS) is a serverless web service used to manage queues of messages that require further processing. This service provides data exchange between different distributed components of the system without message loss and the need for constant availability of each component. In high-load systems, queues allow efficient distribution of data processing between multiple servers or services, [1].

Asynchronous data processing enables theprocessing of the next task without waiting for the completion of the previous one and, in the case of SQS, prevents the inefficient use of system resources during periods of intensive data flow.

Data from different sources will be stored in a queue waiting to be processed. The queue guarantees no message loss on their way, [2].

The research subject is the performance prediction of the distributed serverless application in a cloud environment. Modeling queue behavior allows us to estimate important system metrics, such as how often requests arrive (λ), how quickly they

are processed (μ), and how long it takes to respond, [3].

This research aims to identify the optimal queue configuration to improve system performance.

2 Problem Formulation

2.1 Problem Statement

Organizing distributed computing in serverless applications requires proper interaction between cloud platform services. Most cloud services are used by developers, i.e., with default configuration parameters. This strategy allows us to deploy a serverless application quickly, but it leads to problems in case of increased load.

In many cases, a message queue is the core of a distributed system; for AWS, it is a Simple Queue Service (SQS). The task of the queue is to store received messages and send them for further processing. This method assumes an uneven load over a certain period and requires special attention to the settings. Sudden spikes in service time (μ) and message arrival intensity (λ) can lead to service failures, premature retries of message processing, and errors. Such processes require high-quality modeling to predict system behavior.

Current classical models, despite their strong math, often fail to accurately evaluate application behavior in various operating modes, [4].

It is worth noting the advantages of serverless platforms, such as the automatic scaling of user applications according to load changes.

Choosing the optimal parameters for SQS can effectively balance message processing and waiting time with minimal resource usage. It becomes especially critical in batch processing mode.

The study aims to provide recommendations for selecting SQS configuration parameters to minimize processing time, particularly for large data and high message volume.

2.2 Review of the Literature

Queuing theory is a statistical approach that helps us understand and predict how distributed applications behave. This theory is essential in serverless architecture, ensuring smooth data processing under heavy loads. Classical models fail to ensure scalability and performance due to task arrival (λ) and processing rates (μ) unpredictability. Let us assume that events in the system arrive randomly at an average rate of λ, queueing sequentially. The time it takes to process each event grows much faster as the workload increases. This idea allows us to concentrate on the system's current performance without concerns about the past. It is worth mentioning that the time needed to process an event in real-world situations does not always fit this pattern.

In addition, the event generation time may not correspond to an exponential distribution, [4]. Thus, there is a need to modify classical models, such as M/M/1 and M/M/k. For instance, in hybrid cloud systems, adaptive routing methods have been proposed that adapt at runtime to maximize task processing efficiency and minimize queue delays [5], emphasizing the necessary flexibility to handle adaptive workloads.

Research has shown that implementing adaptive routing policies based on queuing theory increases throughput and reduces latency. Such advances indicate the need for adaptive models that can deal with two key aspects of serverless platforms: to account for the unpredictability of workloads and provide elastic scaling.

Scheduling and prioritization are important aspects of systems designed to handle high loads. Researchers [6], [7], [8] were considering priority-based queues that accelerate the processing of high-priority tasks and minimize delays for critical processes. These methods are becoming increasingly relevant in workflows that combine batch and real-time processing.

The main concept of the research [9] is to optimize queues with batch service mechanisms, where tasks are processed in fixed-size groups rather than individually. These systems differ significantly from classical queue models such as M/M/1, as they involve batch processing, which affects both system performance and dynamics. Increasing the batch size can contribute to increasing throughput, but at the same time can cause longer waiting times due to delays associated with batch formation.

Serverless services such as AWS Lambda and Amazon SQS provide high scalability and cost-effectiveness for modern applications. To dynamically configure parameters, improve performance, and optimize resource utilization such frameworks as FAASTloop have been developed, which select appropriate optimization parameters using statistical models, [10]. The use of cloud environments in such studies has demonstrated the ability of serverless architectures to handle workloads at large scales efficiently, [11].

Elastic scaling of serverless services gives rise to issues related to latency control and resource utilization. Optimizing AWS Lambda performance is a long-term and non-trivial task that requires finding the best configuration, especially under high load conditions [12]. Research shows that a possible

solution is to use AWS Lambda together with SQS. Such a combination can be relatively easily adapted to different workloads, [13].

At present, the influence of SQS configuration on the performance of the developed system can be assessed through empirical tests based on real data; relying solely on theoretical models is insufficient, [14].

2.3 Materials and Methods

Queueing models are the basis for the analysis and optimization of distributed serverless software solutions in cases where there is interaction between several components. One approach to implementing such interaction is message exchange, which can occur both directly and using queues. In the case of serverless architectures, SQS is one of the simplest solutions for organizing message exchange between distributed components. Applying queueing theory to SQS allows assessing the impact of its configuration on system performance, [15].

In queuing theory, the arrival of tasks is modeled using a Poisson process, where the randomness of the task flow is indicated by the average arrival rate (λ). An exponential distribution with an average service rate (μ) is used to model the service time. Together, these characteristics offer a simplified but insightful representation of real systems, [16].

The M/M/1 model describes a single-server scenario, in which a single computing resource processes all incoming tasks.

The use of such a server is determined by the following formula (1):

$$\rho = \frac{\lambda}{\mu}, \tag{1}$$

where $0 \le \rho < 1$ provides stability.

The basic performance indicators are:

- average number of messages in the queue $\left(L_q\right)$:

$$L_q = \frac{\rho^2}{1-\rho}; \tag{2}$$

- is the average waiting time for a message in the queue $\left(W_q\right)$:

$$W_q = \frac{\rho}{\mu(1-\rho)}; \tag{3}$$

- is the average total time for a message (W):

$$W = W_q + \frac{1}{\mu}. \tag{4}$$

These equations demonstrate that $\left(L_q\right)$ and $\left(W_q\right)$ grow exponentially, which leads to a decrease in performance as ρ tends to 1 (high-load server).

This model is particularly relevant for processes of low-level de-parallelization of processing tasks in serverless environments, such as processing messages from an SQS queue in small batches by a single AWS Lambda function. The model allows you to calculate server load, queue length, and waiting time, which are critical for understanding system stability.

The M/M/*k* model is a generalization of a one-server approach for systems with several parallel servers, where data processing is distributed between *k* servers, according to the formula:

$$\rho = \frac{\lambda}{k\mu}, \tag{5}$$

where $0 \le \rho < 1$ provides stability.

The basic indicators of the M/M/*k* model are:

- Probability that all servers are busy (P_{busy}):

$$P_{busy} = \frac{\frac{\rho^k}{k!}}{\sum_{n=0}^{k} \frac{\rho^n}{n!}}. \tag{6}$$

Formula (6) is an Erlang-B formula that calculates the probability of queuing tasks.

- Average number of messages in the queue $\left(L_q\right)$:

$$L_q = P_{busy} \frac{\rho^2}{1-\rho}. \tag{7}$$

- Is the average waiting time for a message in the queue $\left(W_q\right)$:

$$W_q = \frac{L_q}{\lambda}. \tag{8}$$

The M/M/*k* model describes the behavior of massive- parallel systems where multiple AWS Lambda functions simultaneously process message batches from an SQS queue. It is particularly useful for modeling serverless workflows where AWS Lambda is automatically scaled to perform parallel tasks.

This model allows a better understanding of how the increase in the level of parallelism can increase system throughput and reduce queue time,

which is critically important during high-load periods, [17].

To meet the unique requirements of serverless systems, more complex concepts of queuing theory are needed in addition to the basic models. For example, priority queues are often used to ensure that critical tasks are completed faster than less important ones. This is especially important for workflows that combine real-time and batch processing. Similarly, batch processing itself (b) directly affects server load and queue length, scaling the rate of task arrival depending on the batch size:

$$\lambda_{effective} = b\lambda_{batch}. \quad (9)$$

For example, the task arrival rate is directly dependent on the batch size. Batch processing increases throughput, but individual tasks within a batch may experience delays, [18].

The retry mechanism, which is controlled by the visibility timeout parameter in SQS, is another important component of serverless systems.

A message will return to the queue if its processing is not completed within the message visibility time. Misconfigured visibility time settings may result in unnecessary delays and increase the number of messages in the queue, causing unstable system operation.

3 Problem Solution

3.1 Experiments

For practical experiments, we developed and deployed a batch data processing system for generating PDF documents in a serverless environment on the Amazon Web Services (AWS) platform. The architecture of the developed system is shown in Figure 1.

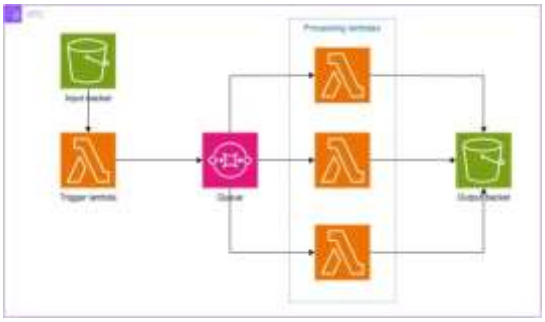

Fig. 1: Architecture of a simple batch data processing system for generating PDF documents

To start working with the system, we need to upload the files to the Amazon S3 Input Bucket. The files contain the data needed to create PDF documents. The data needs to be divided into batches and added to the queue for further processing. For this purpose, we launch a scheduler function that will already interact with the queue. The queue stores the batches and distributes them evenly among the data processors. In sudden load changes, the queue guarantees the successful completion of processing.

Batches from the SQS queue were processed with AWS Lambda. Based on the number of messages in the queue, the cloud platform runs a variable number of function instances, constrained by a predefined limit. The result of each function is a generated PDF file.

We tested the solution using Locust, an effective tool for evaluating serverless application performance under heavy load. During the testing, we evaluated the system's throughput, latency, and automatic scalability under different queue parameters.

The dataset we used to generate single-page PDF files consisted of 500,000 records.

To find the best interrelation between throughput and latency, the batch sizes varied from 10 to 100 messages per batch.

Taking into account that studying SQS behavior requires additional factors, not covered by classical models, we also simulated the message visibility time change from 30 to 900 seconds to assess its influence on system throughput and delay time between messages, which ranged from 0 to 15 minutes in the scenarios where processing incremental data is beneficial.

3.2 Results

Experimental study of the batch processing system allows us to assess the impact of different configurations of the AWS SQS queue on the performance, scalability, and cost of using the developed application. As part of the study, batches of different sizes (10, 25, 50, 75, and 100 records in a batch) were tested, and 500000 records were processed with a fixed period of message storage in the queue (14 days) and with variable values for the visibility timeout (from 10 to 900 seconds) and delivery delay of messages (from 0 to 900 seconds). To determine the optimal configuration that provides the best balance between performance, stability, and cost-effectiveness of the system, the application was tested using different configuration options.

The experimental results are presented in Figure 2, Figure 3, Figure 4, Figure 5 and Figure 6.

Figure 2 shows the distribution of service time in batch processing systems, which provides a more

detailed idea of the system efficiency for different batch sizes.

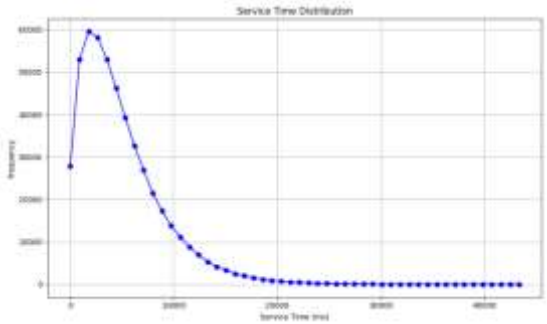

Fig. 2: Distribution of service time for batch data processing systems

The service rate (μ) is the inverse of the average service time (T_s), [19]:

$$T_s = \frac{1}{\mu}. \tag{10}$$

According to observations, the average execution time of one Lambda function for data processing is approximately 2.9 seconds, then the service speed will equal to:

$$\mu = \frac{1}{2.9} \approx 0.34 \frac{tasks}{seconds}. \tag{11}$$

Throughput depends on the batch size and average execution time, [20]:

$$Throughput\left(\frac{records}{seconds}\right) = \\ = \frac{b}{Average\,Execution\,Times(seconds)}. \tag{12}$$

Figure 3 shows the dependence of system throughput on visibility time for different batch sizes (10, 25, 50, 75, and 100 records in a batch).

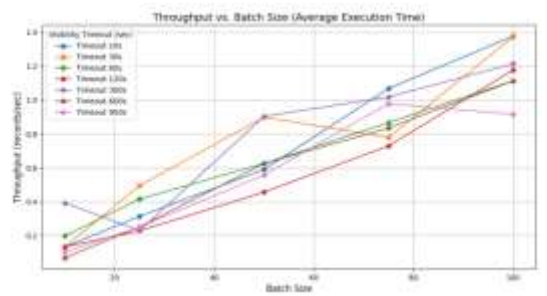

Fig. 3: Impact of visibility time on system throughput for different batch sizes

If the message visibility time is too short, reprocessing occurs. The reprocessing rate is calculated by the formula:

$$Reprocessing\ Probability = \\ = \frac{Execution\,Time\ (seconds)}{Visibility\,Timeout(seconds)}. \tag{13}$$

For our system, the dependence of data reprocessing on message visibility time is shown in Figure 4.

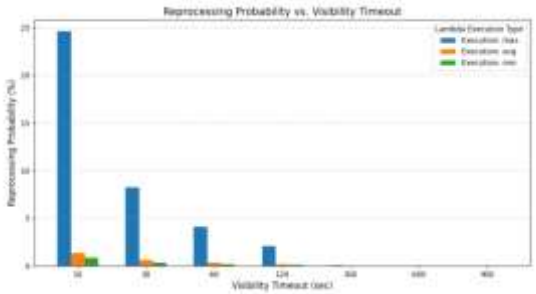

Fig. 4: Impact of message visibility time on reprocessing probability

In cloud batch processing systems, where the frequency of failures (due to serverless function timeouts, retries, or system errors) affects the overall throughput and processing duration, the total time required to process all records should be adjusted taking into account the reprocessing probability:

$$Total\ Processing\ Time = \\ = \frac{Total\ \ Records}{Throughput}(1 + Reprocessing\ Probability). \tag{14}$$

Thus, Figure 5 plots the dependence graphs of the total processing time on the time the message is visible in the queue.

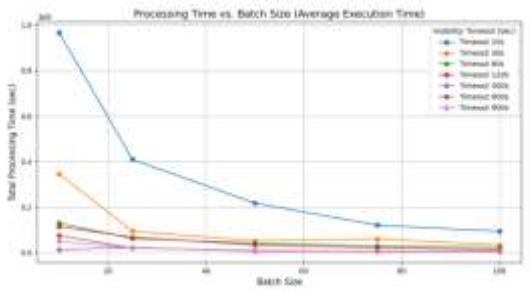

Fig. 5: Impact of message visibility time on total processing time

The heat map in Figure 6 demonstrates the variation of total data processing time depending on different batch sizes and message delivery delays and illustrates the impact of message delivery delay

and batch size on the performance of a data processing system using SQS.

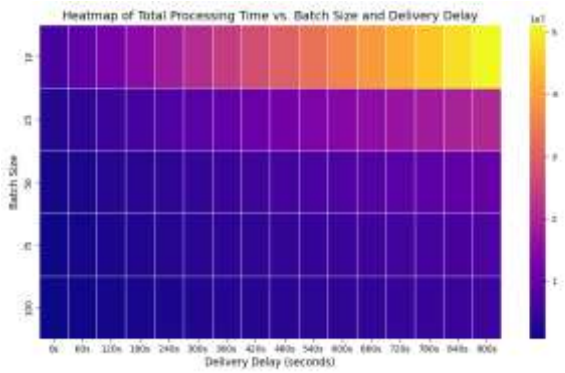

Fig. 6: Impact of message visibility time on total processing time for different batch sizes

During experiments, the queueing period did not demonstrate a direct effect on system behavior.

4 Conclusion

The paper presents a study of the behavior of serverless systems during distributed processing of significant amounts of data. A prototype of a serverless application for the mass generation of PDF documents using Python and the AWS Boto3 library was created. The throughput, latency, and data processing time were analyzed under different configurations. Theoretical hypotheses were tested through empirical experiments.

The scientific novelty of the work lies in the application of queueing theory models to serverless computing. Workflows for processing 500,000 records were simulated, using classical queueing theory models to predict and optimize system behavior. The study extends the theoretical understanding of queue dynamics by introducing a parameterized framework that combines batch size, message visibility timeout, and message delivery delay to analyze their impact on system performance.

The results obtained during the study have practical significance, namely, providing recommendations for better SQS configuring in order to improve software performance. Determining the optimal values of parameters such as batch size (50 records), visibility time (600 seconds), and delivery delay (300 seconds) enables significantly reducing response latency and increasing system throughput. The obtained data emphasize the importance of careful selection of SQS settings according to the load characteristics, which influences the reliability and efficiency of message processing.

Further research involves developing approaches and algorithms to determine the optimal combinations of SQS configuration parameters depending on the predicted loads in batch data processing systems. Furthermore, attention should be paid to finding the optimal rate of message arrival into the system, which, in combination with the appropriate SQS configuration parameters, will minimize the average response time and ensure the optimal performance of serverless solutions.

Declaration of Generative AI and AI-assisted Technologies in the Writing Process

During the preparation of this work the authors used ChatGPT, Grammarly in order to improve the readability, language of the manuscript and enhance academic quality. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

References:

[1] Kaplan, Andreas, Haenlein, Michael, Users of the World, Unite! The Challenges and Opportunities of Social Media. *Business Horizons*, 2010, Vol.53, No.1, pp. 59-68. http://dx.doi.org/10.1016/j.bushor.2009.09.003.

[2] Li, Jing, Cui, Yidong, Ma, Yan, Modeling Message Queueing Services with Reliability Guarantee in Cloud Computing Environment Using Colored Petri Nets, *Mathematical Problems in Engineering*, 2015, Vol. 2015, No. 383846, pp. 1-20. https://doi.org/10.1155/2015/383846.

[3] Kostis Kaffes, Neeraja J. Yadwadkar, and Christos Kozyrakis. Hermod: principled and practical scheduling for serverless functions. In *Proceedings of the 13th Symposium on Cloud Computing (SoCC '22)*. Association for Computing Machinery, New York, NY, USA, 2022, pp. 289–305. http://doi.org/10.1145/3542929.3563468.

[4] Mario Lefebvre. Reducing the Size of a Waiting Line Optimally. *WSEAS Transactions on Systems and Control*, 2023, Vol. 18, pp. 342-345. http://dx.doi.org/10.37394/23203.2023.18.35.

[5] Fatouros, G., Kousiouris, G., Makridis, G. (2023). Enhanced Runtime-Adaptable Routing for Serverless Functions Based on

Performance and Cost Tradeoffs in Hybrid Cloud Settings. *2023 IEEE International Conference on Cloud Computing Technology and Science (CloudCom),* Naples, Italy, 2023, pp. 177-184. http://dx.doi.org/10.1109/CloudCom59040.2023.00038.

[6] Yuqi Fu, Li Liu, Haoliang Wang, Yue Cheng, and Songqing Chen, SFS: smart OS scheduling for serverless functions. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '22).* Dallas, Texas, USA, IEEE Press, 2022, No. 42, pp. 1–16.

[7] Tantalaki, Nicoleta & Souravlas, Stavros & Roumeliotis, Manos. A review on Big Data real-time stream processing and its scheduling techniques, *International Journal of Parallel, Emergent and Distributed Systems,* 2019, Vol.35, No. 5, pp. 571-601. https://doi.org/10.1080/17445760.2019.1585848.

[8] Xu, Le & Venkataraman, Shivaram & Gupta, Indranil & Mai, Luo & Potharaju, Rahul. Move Fast and Meet Deadlines: Fine-grained Real-time Stream Processing with Cameo, *Proceedings of the 18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021,* Santa Clara, CA, USA, 2021, pp. 389-405.

[9] Santhi, K.s & Ramakrishnan, Saravanan. Performance analysis of cloud computing bulk service using queueing models. *International Journal of Applied Engineering Research*, 2017, Vol.12, No. 17, pp. 6487-6492.

[10] Shruti Mohanty, Vivek M. Bhasi, Myungjun Son, Mahmut Taylan Kandemir, and Chita Das. FAAStloop: Optimizing Loop-Based Applications for Serverless Computing. In *Proceedings of the 2024 ACM Symposium on Cloud Computing (SoCC '24).* Association for Computing Machinery, New York, NY, USA, 2024, pp. 943-960. https://doi.org/10.1145/3698038.3698560.

[11] Risco Gallardo, S. *Serverless Strategies and Tools in the Cloud Computing Continuum* (Doctoral dissertation, Universitat Politècnica de València), 2024.

[12] Bechir, M.L., Bouh, C.S., Shuwail, A., Comprehensive Review of Performance Optimization Strategies for Serverless Applications on AWS Lambda. *ArXiv, abs/2407.10397*, 2024. https://doi.org/10.48550/arXiv.2407.10397.

[13] Kaplunovich, A., Joshi, K. P., Yesha, Y., Scalability Analysis of Blockchain on a Serverless Cloud, *2019 IEEE International Conference on Big Data (Big Data)*, Los Angeles, CA, USA, 2019, pp. 4214-4222. https://doi.org/10.1109/BigData47090.2019.9005529.

[14] Alemu, M., *Serverless Automated Assessment of Programming Assignments*. Aalto University Repository, 2023, 70 p.

[15] Amazon Simple Queue Service Documentation, [Online]. https://docs.aws.amazon.com/sqs/ (Accessed Date: December 13, 2024).

[16] Gautam, N., *Operations research and management science handbook*, edited by A. Ravi Ravindran, (1st ed.), CRC Press, 2007.

[17] Makhdoom, I., A new optimum statistical estimation of the traffic intensity parameter for the M/M/1/K queuing model based on fuzzy and non-fuzzy criteria. *Journal of Data Science and Modeling*, 2023, Vol.2, No. 1, pp. 163-184. https://doi.org/10.22054/jdsm.2024.79643.1048

[18] Addya, S. K., Turuk, A. K., Sahoo, B., Sarkar, M., A hybrid queuing model for Virtual Machine placement in cloud data center, *2015 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS)*, Kolkata, India, 2015, pp. 1-3. https://doi.org/10.1109/ANTS.2015.7413642.

[19] Mireslami, S., Rakai, L., Wang, M., Far, B. H., Dynamic Cloud Resource Allocation Considering Demand Uncertainty, *IEEE Transactions on Cloud Computing*, 2021, Vol.9, No. 3, pp. 981-994. https://doi.org/10.1109/TCC.2019.2897304.

[20] Alsurdeh, R., Calheiros, R. N., Matawie, K. M., Javadi, B., Hybrid Workflow Scheduling on Edge Cloud Computing Systems, *IEEE Access*, 2021, Vol. 9, pp. 134783-134799. https://doi.org/10.1109/ACCESS.2021.3116716

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

- Sergey Ostapov was responsible for the research activity planning execution, and formulation of research goals and aims.
- Oleksandr Kyrychenko carried out the research and investigation process; implemented a prototype of a Cloud solution; and prepared a literature review.
- Oksana Kyrychenko created models; analysed sources, validated research outputs, and wrote original drafts.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

No funding was received for conducting this study.

Conflict of Interest

The authors have no conflicts of interest to declare.