КОМПЬЮТЕРНЫЕ НАУКИ, ПРИБОСТРОЕНИЕ И АВТОМАТИЗАЦИЯ
КОМПЬЮТЕРЛІК ҒЫЛЫМДАР, АСПАП ЖАСАУ ЖӘНЕ АВТОМАТТАНДЫРУ
COMPUTER SCIENCE, INSTRUMENTATION AND AUTOMATION

UDC 004.75
IRSTI 20.01.07
https://doi.org/10.53364/24138614_2026_41_2_10

A.B. Kassymova[1*], R.K. Uskenbayeva[1], Young Im Cho[2], V. Elle[1], A.K. Smakhanova[1]
[1] Satbayev University, Almaty, Kazakhstan
[2]Gachon University, Gyeonggi-do, South Korea

[*]*E-mail: a.kassymova@satbayev.university*

METHODS FOR CONSTRUCTING AND EVALUATING A SERVERLESS ARCHITECTURE FOR A DISTRIBUTED INFORMATION SYSTEM UNDER PEAK LOADS

Abstract. *This paper examines an approach to building an event-driven serverless architecture for a distributed information system operating under uneven and peak load conditions. Modern digital services are characterized by sharp fluctuations in the intensity of incoming requests, which requires maintaining operational stability, acceptable response times, and the ability to quickly scale computing resources. Traditional monolithic and container-based solutions in such environments often require upfront capacity reservations or respond to load surges with delays.*

The goal of this study is to develop and experimentally evaluate an architectural solution in which request processing is organized as a stream of independent events using a serverless computing model. To this end, a formalized event model and input load generation scheme are proposed, enabling the reproduction of both normal operating modes and short-term peak impacts.

The effectiveness was assessed based on a series of controlled computational experiments under various load scenarios. The key metrics used were average processing latency, the p95 metric, system throughput, and the error rate during periods of increased activity. The results obtained during the study demonstrate that as the load increases, the increase in latency is manageable, and the system maintains operability during short-term overloads. This allows us to consider the proposed approach as a promising solution for scalable distributed services.

Keywords: *event-driven architecture, serverless computing, distributed information systems, scalability, peak load, tail latency, performance.*

Introduction.

Modern distributed information systems function under conditions of constantly changing load. In real operation, periods of moderate activity are replaced by short-term bursts of traffic caused by mass user actions, external events, regular operations or peculiarities of business processes. For such systems, not only high performance is important, but also the ability to maintain stable operation during a sharp increase in the number of requests.

Traditional architectural approaches based on monolithic applications or container infrastructure do not always provide the necessary flexibility in such scenarios. For the guaranteed passage of peak loads, it is often necessary to reserve computing resources in advance, which leads to their incomplete use during normal work periods. An alternative option is associated with dynamic scaling, but it is also accompanied by time costs for launching additional instances of services and redistributing the load.

In recent years, the model of serverless computing has attracted considerable interest, in which the applied logic is executed in the form of separate functions that are triggered by an event, and the management of the computing infrastructure is transferred to the cloud platform. Such an approach allows reducing operational costs, simplifying system maintenance, and providing a more flexible response to changes in the input flow of requests [1, 2]. By their nature, serverless solutions are closely related to event-oriented architecture, where the interaction of components is built around event processing and asynchronous messaging [3].

Research on serverless technologies covers various aspects of their application. The features of real workloads, the mechanism of automatic scaling, the influence of the reuse of the computing environment on the delay of processing, as well as the limitations that arise at a high intensity of processing are considered [4, 5]. Special attention is paid to the effect of "cold start", when the function starts after a period of idleness and requires additional time for initialization [6]. For systems with strict response time requirements, this factor can have a significant impact on the quality of service.

In addition, recent studies have examined the energy efficiency of serverless platforms, optimization of resource allocation and selection of function boundaries when designing application solutions are discussed [7–9]. There are review works that systematize the accumulated experience of implementing the serverless approach, typical usage scenarios and directions for further development of the technology [10–12]. However, a significant part of the publications is focused either on conceptual analysis or on observations in production environments, while reproducible experimental studies of the behavior of such systems under controlled peak loads are presented to a limited extent.

A research area focused on developing methods to mitigate the negative impact of cold starts, including the use of adaptive and learning approaches. However, empirical studies show that cold starts and tail delays remain a significant factor, especially during short-term peak loads. Recent research confirms that tail metric analysis is crucial for the engineering evaluation of serverless architectures [13-15].

Several studies have examined the performance characteristics and architectural models of serverless computing. However, most existing work focuses either on the theoretical analysis of serverless platforms or on general workload characterization. A comparative analysis of these studies is presented in Table 1.

Table 1 – Comparison of existing studies on serverless architectures

Study	Architecture type	Research focus	Load conditions	Evaluation metrics	Limitations	Key Contribution	Peak-load evaluation
Castro et al. (2019)	Serverless cloud platforms	Overview of serverless paradigm	Cloud workloads	Performance characteristics	Lacks experimental validation under peak loads	Provides foundational understanding of serverless computing	No
Shahrad et al. (2020)	Large-scale serverless infrastructure	Characterization of real workloads	Production cloud traces	Execution time distribution	Focused on observational data, not controlled experiments	Reveals real-world workload behavior	No

Wen et al. (2023)	Serverless systems	Systematic review of serverless computing	General workloads	Performance variability	Does not include experimental modeling	Summarizes existing research trends	No
Vahidinia et al. (2023)	Serverless with ML optimization	Cold start mitigation	Dynamic workloads	Response time	Focus limited to cold start problem	Introduces ML-based optimization approach	Partial
This study	Event-driven serverless architecture	Distributed event processing under peak loads	Synthetic peak load scenarios	Latency, throughput, scalability	Synthetic workload environment	Proposes formalized event model and experimental evaluation under peak loads	Yes

The comparison presented in Table 1 shows that most existing research focuses either on general architectural principles or specific performance aspects of serverless systems. In contrast, this paper proposes a comprehensive approach combining formal modeling, architectural design, and controlled experimental evaluation under peak load conditions. This enables a more systematic assessment of scalability and resilience in distributed event-driven systems.

At the same time, some aspects of serverless system behavior remain poorly understood. Distributed systems must withstand peak loads, cold start effects, and execution time variability, but reproducible experimental evaluation of these factors remains challenging. This makes the design and validation of serverless event-driven system architectures for peak load scenarios a pressing scientific and engineering challenge.

This study addresses the problem of designing scalable distributed systems capable of operating under highly dynamic and peak load conditions. While existing research in serverless computing has primarily focused on general architectural concepts or individual performance aspects, a systematic approach to modeling and experimentally evaluating such systems under controlled peak load conditions remains limited.

The main contributions of this study are:

– a formalized event flow model that provides a reproducible representation of both baseline and peak load scenarios in distributed information systems;

– development of an event-driven serverless architecture that enables dynamic scaling and isolation of parallel request processing;

– a comprehensive experimental methodology combining workload modeling, controlled scenario generation, and statistical analysis of performance metrics;

– comprehensive evaluation of system behavior using latency metrics (including tail latency), throughput, and scalability under various load conditions.

Unlike previous studies, which often rely on observational data or focus on individual performance factors such as cold starts, the proposed approach provides a unified framework for analyzing the scalability and fault tolerance of serverless architectures under peak load conditions.

The results obtained in this study contribute to the development of engineering methods for designing distributed event-driven systems and can be applied in areas such as smart city platforms, IoT infrastructures, and large-scale digital services.

Materials and research methods.

The study was conducted using an engineering and experimental approach focused on the development and evaluation of a serverless architecture for a distributed information system operating under variable and peak load conditions. The methodology includes formalizing the input event flow, describing the request processing architecture, selecting quantitative performance indicators, and conducting a series of reproducible computational experiments.

Event and Input Flow Model.

The operation of the system is represented as a sequence of independent events, each of which corresponds to a distinct user request or internal processing operation. An event can be represented as a set of parameters:

$$e_i = \langle t_i^{in}, a_i, d_i \rangle \quad (1)$$

where t_i^{in} – is the event arrival time; a_i – is the event type; d_i – is the set of parameters associated with event processing.

The input flow intensity is determined by the function $\lambda(t)$, which reflects the change in load over time. For the basic operating mode, a constant intensity is used:

$$\lambda(t) = \lambda_0 \quad (2)$$

To simulate short-term overloads, the load growth factor k is used, then:

$$\lambda(t) = k \cdot \lambda_0, k \geq 1 \quad (3)$$

In the experimental part, scenarios with coefficients $k = 3$ and $k = 5$ were considered, which corresponds to a sharp increase in the number of requests compared to the normal operating mode.

The selected load model is intended to reproduce typical operating modes of a distributed system and analyze the impact of an increase in the intensity of the input flow on the characteristics of event processing. Using the coefficient (k) allows you to simulate different load levels, ensuring reproducibility of experiments and comparability of results between individual scenarios. Despite the simplified nature of the model, it is sufficient to assess the scalability and stability of the architecture under study.

To assess the stability of the system, a simplified service model was additionally used, in which the load factor of the computing subsystem is determined by the expression:

$$\rho = \frac{\lambda}{\mu} \quad (4)$$

where λ – arrival rate of events; μ – average event processing rate.

Stable operation is achieved when the following condition is met:

$$\rho < 1 \quad (5)$$

which means that the system's throughput exceeds the average input flow.

The serverless architecture dynamically adjusts the number of simultaneously executed functions, effectively increasing the value of μ during peak loads.

Event Processing Architecture.

The proposed architecture is built on an event-driven principle. Incoming requests are received by the external access interface and converted into messages sent to the event queue. Independent processing functions are then launched, each performing a separate operation without being tied to pre-allocated computing resources.

The architecture (Fig. 1) consists of several functional layers responsible for request reception, event routing, distributed processing, and persistent storage of results.

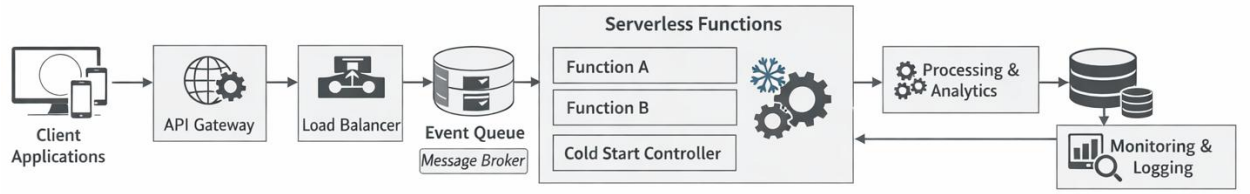

Figure 1 – Architecture of the event-driven serverless processing system

This architectural organization separates the request incoming stage and the computation stage, reduce the mutual influence of parallel operations, and automatically increase the number of functions executed as the load increases.

The processing sequence for a single event can be described by the following *Event processing algorithm:*

1. When an event e arrives, it is placed in the event queue.
2. The scheduler initiates execution of the processing function.
3. The function loads the necessary parameters and execution context.
4. The event is processed, and the result is generated.
5. The result is saved to storage, and the event is marked as processed.

Isolating individual calls reduces the likelihood of cascading delays and improves system resilience when faced with a flood of requests.

Cold start mechanism.

In serverless architectures, the latency to process an event can depend on the state of the computing environment when the request arrives. If the function has already been active and its execution environment is saved by the platform, the request is processed in warm start mode. In this case, the function starts faster because the main components of the environment are already initialized.

Cold start occurs in a situation where a new function execution environment must be created to process an event. This process includes allocating computing resources, loading the runtime, initializing software dependencies, and running custom code. As a result, individual request processing times can increase, which is especially noticeable in tail latency metrics such as p95.

In this study, cold start is considered as an additional factor influencing the latency of event processing under variable and peak loads. To take this into account, in the experimental part, a different probability of a cold start of the function was introduced. This made it possible to evaluate how an increase in the proportion of cold start events affects the tail latency of the system.

Performance Evaluation Metrics.

The following metrics were used to quantitatively evaluate the effectiveness of the architecture.

Average Event Processing Latency:

$$T_{avg} = \frac{1}{N}\sum_{i=1}^{N}(t_i^{out} - t_i^{in}) \quad (6)$$

where t_i^{out} – is the moment the event processing is completed; N – is the total number of events processed.
Standard deviation:

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})2} \quad (7)$$

where σ is the standard deviation, x_i is the separate measurement, $\bar{x}$ is the average value, n is the number of observations.

Standard deviation was used to assess the stability of results between repeated experiments and to determine the degree of variability in the measured parameters.

The p95 quantile metric is the latency value below which 95% of requests are completed.

System throughput:

$$Q = \frac{N}{\Delta t} \quad (8)$$

where Δt – is the observation time interval.
Scalability factor:

$$S = \frac{Q_{max}}{Q_{base}} \quad (9)$$

where Q_{base} – is the throughput in base mode; Q_{max} – is the throughput during peak load growth.

Additionally, the proportion of request processing errors during peak periods was considered.

Experimental Research Methodology.

The experimental evaluation was conducted in a controlled computing environment using synthetically generated event streams. For each load scenario, at least five experiment iterations were performed, after which the average metric values and latency tail characteristics were calculated.

The experimental platform included AWS Lambda services to perform event processing functions, Amazon API Gateway to accept requests, and Amazon SQS as a message queue. Metrics were collected using Amazon CloudWatch. The configuration of the medium remained unchanged in all series of experiments.

The experimental procedure used in this study is summarized in Figure 2. The workflow includes the generation of input workloads, definition of experimental scenarios, submission of event streams to the serverless platform, and systematic collection and analysis of performance metrics. This structured approach ensures consistency and reproducibility of the experimental results.

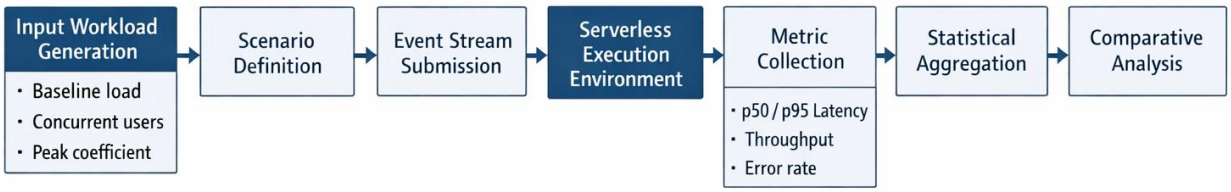

Figure 2 – Experimental workflow for evaluating the event-driven serverless architecture

As shown in Figure 2, the experimental process begins with the generation of input workloads, including baseline traffic, varying numbers of concurrent users, and peak load scenarios. These workloads are translated into event streams and submitted to the serverless execution environment, where event-triggered functions are executed dynamically. During execution, key performance indicators such as latency (p50 and p95), throughput, and error rate

are collected. The results are then aggregated and statistically processed across multiple experimental runs, enabling a consistent comparison of system behavior under different load conditions.

The following scenarios were analyzed during the experiments:

– baseline load with a fixed input flow rate.

– increasing the number of concurrent users.

– short-term load spikes with coefficients $k = 3$ and $k = 5$.

The presented architecture reflects a real-life version of the organization of an event-driven serverless system, while the numerical results presented in the "Research Results" section were obtained during controlled computational experiments using synthetically generated event streams. The study is not aimed at measuring the performance of a specific industrial system, but at analyzing the behavior of the proposed architecture under various load scenarios in a reproducible experimental environment.

This methodology ensures reproducible results and comparability of performance metrics under different system operating modes.

To assess the stability of the results, each series of experiments was performed at least five times. Based on the results of repeated runs, average values of indicators and standard deviations were calculated. This made it possible to reduce the influence of random factors in the computing environment and ensure the comparability of the results obtained between different load scenarios.

Results and discussion.

The experimental section aimed to evaluate the performance, scalability, and resilience of the proposed serverless architecture under various load conditions. All results were obtained in accordance with the methodology outlined in the previous section, using reproducible input stream generation scenarios.

Performance and Scalability

Table 2 presents the average and tail latency of event processing with increasing concurrent user counts. Measurements were performed with a fixed input flow rate in baseline mode.

When analyzing the results, averaged values of indicators obtained from several independent series of experiments were used. To control the stability of measurements, the variability of the results was additionally assessed. In all scenarios considered, deviations between repeated experiments remained insignificant, which indicates the reproducibility of the data obtained.

Table 2 – Event processing latency indicators with increasing number of users

Number of users	T_{avg}, s	p50, s	p95, s
100	0,48	0,42	0,63
500	0,66	0,59	0,96
1 000	0,90	0,82	1,45
2 000	1,08	1,01	1,68
5 000	1,25	1,18	1,93
10 000	1,44	1,35	2,15

Statistical analysis of repeated runs showed that the standard deviation of the measured indicators did not exceed 7–10% of the average value. This allows us to consider the results stable and suitable for further analysis of architecture performance.

The results indicate a gradual increase in latency as the number of users grows, without abrupt jumps or signs of performance collapse. The dependence of p50 and p95 latency on the number of users is clearly shown in Figure 3.

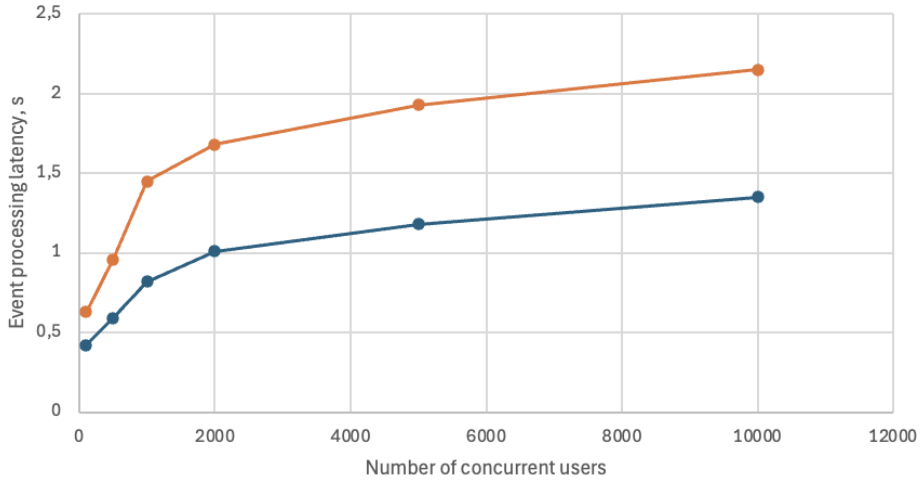

Figure 3 – Dependence of event processing delay (p50 and p95) on the number of concurrent users

As illustrated in Figure 3, latency changes smoothly without abrupt jumps.

System Throughput.

System throughput was estimated as the number of events processed per unit of time. Table 3 shows the throughput values at various load levels.

Table 3 – System Throughput with Increasing Load

Number of users	Throughput, events/sec
100	110
500	210
1 000	320
2 000	410
5 000	560
10 000	730

Throughput increases almost linearly, indicating the absence of a clear bottleneck in the considered workload range.

To better understand the tradeoff between processing speed and system throughput, the relationship between p95 event processing latency and throughput was analyzed for different workload levels (Figure 4).

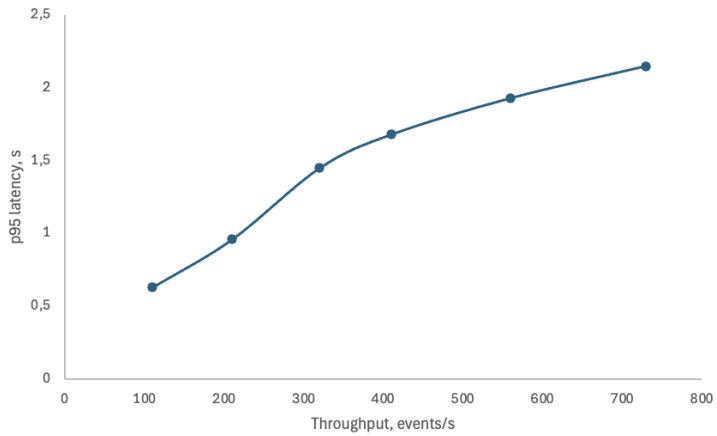

Figure 4 – Relationship between system throughput and p95 event processing latency

Figure 4 shows that increasing throughput is accompanied by a gradual rise in latency at the tail of the distribution. At low and medium load levels, the increase in latency remains moderate, indicating efficient scaling of serverless functions and balanced resource distribution. At higher throughput values, latency continues to increase, but no sharp degradation is observed, confirming the stability of the proposed architecture with increasing processing intensity. This suggests that the architecture maintains a balance between computing power and response time, which is critical for distributed event-driven systems under peak load conditions.

In addition to the overall throughput-latency relationship, it is important to evaluate the system's behavior under short-term peak loads.

Peak Load Resilience.

To evaluate the system's behavior under short-term bursts of requests, experiments were conducted with load growth factors of $k = 3$ and $k = 5$. The results are presented in Table 4, and the change in the p95 metric is shown in Figure 5.

Table 4 – Peak Load Resilience Indicators

Peak coefficient k	p95, s	Error rate, %
1 (base mode)	1,62	0,1
3	1,94	0,2
5	2,31	0,3

No event loss was observed in any of the scenarios considered. The change in tail latency with increasing load intensity is shown in Figure 5.

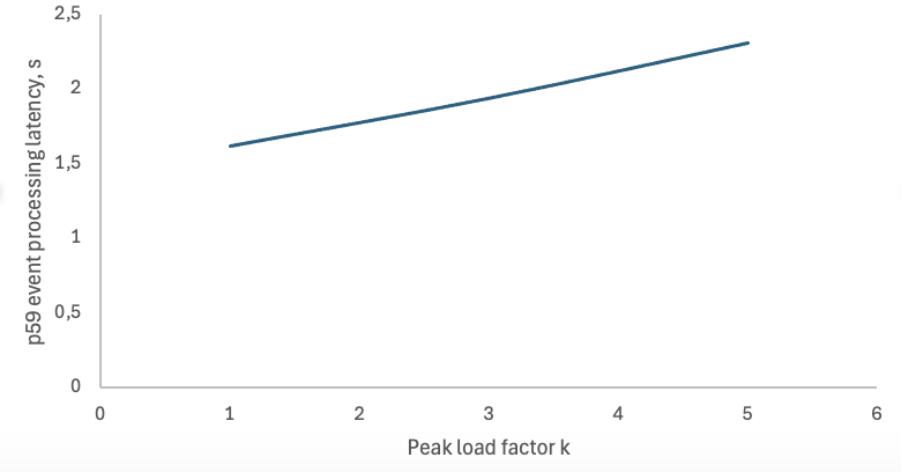

Figure 5 – Impact of peak load growth factor on p95 event processing latency

From a practical perspective, the proposed approach can be applied in systems where the load can change sharply under the influence of external factors, for example, during mass user requests or the launch of service operations.

Impact of cold start latency.

A separate series of experiments was conducted to evaluate the impact of cold start functions. In these scenarios, some requests were processed under warm start conditions, and some were processed under cold start conditions, when the function’s computing environment was created anew.

The probability of cold start varied from 0% to 40%. For each value, the p95 metric was recorded, since it is the tail delay that is most sensitive to single slow calls. The results of the experiment are presented in Table 5.

Table 5 – Influence of cold starts on event processing latency

Cold start probability	p95 latency, s
0 %	1.62
10 %	1.75
20 %	1.93
40 %	2.41

The data in Table 5 shows that an increase in the cold start probability leads to an increase in p95 latency. This is explained by the fact that individual requests require additional time to initialize the computing environment. Moreover, even with a cold start probability of 40%, the increase in latency remains manageable, which indicates that the architecture remains stable under the conditions under consideration.

Comparison with containerized architecture.

To compare performance, a control version of the system was implemented using a containerized architecture with automatic scaling. Experiments were conducted under an equivalent load (5,000 concurrent users) (Table 6).

Table 6 – Comparison of serverless and containerized architectures

Indicator	Serverless	Container
p95 delay, s	1,93	3,10
Scaling time, s	< 0,1	10–40
Error rate at peak, %	0,3	4,5

Table 7 – Architectural characteristics of the compared approaches

Characteristic	Serverless	Container
Resource allocation	Event-driven	Instance-based
Scaling granularity	Function level	Container level
Startup overhead	Low	Medium/High
Infrastructure management	Platform-managed	User-managed
Adaptation to burst load	High	Moderate

Table 7 summarizes the architectural differences that explain the performance characteristics observed during the experiments. These differences should be considered when interpreting latency and scalability results.

The obtained results highlight differences in the scaling dynamics and resilience of architectures under sudden load increases.

Architectural interpretation of the comparison results.

To correctly interpret the results obtained, it is necessary to consider the differences in the operating principles of the compared architectures. In the serverless model, computing resources are allocated automatically in response to incoming events, while the container architecture requires launching additional instances of services as the load increases.

In this experiment, the container system used an automatic scaling mechanism based on the creation of new containers when load thresholds were reached. This process includes allocating resources, launching the container, and initializing the application service, which leads to additional time costs. In contrast, a serverless platform scales at the function level, allowing it to respond much faster to changes in input flow.

The purpose of the comparison was not to prove the absolute superiority of one architecture over another. Container solutions offer several advantages for long-term sustained workloads and the need for complete control over the runtime environment. However, in scenarios with sudden short-term changes in request intensity, the serverless approach demonstrates higher adaptability and lower tail latencies.

The results obtained confirm that the choice of architectural approach should be determined by the nature of the workload and the requirements of a specific subject area.

Experimental results demonstrate that the serverless architecture delivers consistent performance under both intermittent and peak load conditions. The results indicate that increasing the number of concurrent users leads to a predictable increase in both average and tail latency for event processing, without any sudden spikes or degradation of system performance.

As shown in Figure 4, latency increases in a controlled manner as the workload grows, indicating the correct operation of the autoscaling mechanism within the studied range. The absence of nonlinear discontinuities in the p95 latency graph indicates that the system does not reach saturation within the considered load range. This suggests that the chosen event processing and computational resource allocation model adequately matches the nature of the input flow described in the "Materials and Methods" section.

The results of the peak load resilience analysis (Table 4, Figure 5) show that even with a fivefold increase in input flow intensity, the increase in tail latency is moderate. The absence of event loss and the low rate of processing errors indicate that isolating event processing and invoking functions effectively reduce interactions between concurrent requests. This is especially important for distributed systems, where load can fluctuate rapidly due to external factors.

A comparison with a containerized architecture demonstrates differences in scaling dynamics. The serverless architecture is characterized by near-instantaneous provisioning of computing resources, ensuring stable latency performance under rapid load fluctuations. The containerized system exhibits noticeable delays associated with deploying and initializing additional service instances, which is reflected in the tail metrics.

The impact of cold function starts on latency characteristics deserves special attention. Although cold starts lead to increased processing time for individual events, their contribution to overall latency dynamics was limited in the experiments conducted. This may be due to specific load distribution characteristics and the reuse of computing environments during sequential event processing. However, when designing systems with more stringent response time requirements, this factor requires additional consideration and optimization. A discussion of the results shows that the proposed architecture provides a compromise between ease of scalability, resilience to peak loads, and acceptable latency at the end of the critical period.

These results can be used in the design of distributed digital services that require automatic scaling and stable operation under uneven load conditions. Further research is needed to analyze long-term load conditions, power consumption, and system behavior under more complex mixed-use scenarios.

Limitations of the Study.

Despite the positive results obtained in the study, we note several limitations. The experimental evaluation was conducted using synthetically generated event streams, which may not fully reflect the complexity of real-world workloads. Furthermore, the experiments considered a limited range of workload scenarios and system configurations. Cold start mitigation strategies were not thoroughly explored.

Future work may include evaluating the proposed architecture under real-world production workloads, analyzing energy efficiency in serverless environments, and integrating adaptive scheduling mechanisms.

Conclusion.

This study examined and experimentally evaluated an event-driven serverless approach for distributed systems. The focus is on the engineering aspects of architectural design, event model formalization, and experimental evaluation of system performance and fault tolerance.

A model for describing the incoming event flow and a request processing algorithm are proposed, formalizing system behavior under various load conditions. Experimental

measurements confirm that the serverless approach ensures automatic scaling of computing resources and isolation of individual event processing.

Experimental evaluation showed that the proposed architecture exhibits a predictable increase in average and tail latency for event processing with an increasing number of concurrent users. Analysis of peak load scenarios revealed the system's resilience to short-term spikes in incoming flow intensity without event loss and with a limited increase in tail latency. A comparison with a containerized architecture demonstrated the superiority of the serverless approach in terms of scaling time and resilience to sudden load changes.

The findings of this study suggest that event-driven serverless architectures can be considered an effective engineering solution for distributed information systems with highly variable workloads.

Future research areas include expanding the experimental base by analyzing a wider range of workloads, studying the impact of cold start management strategies on tail-end latencies, and evaluating the energy efficiency of serverless architectures over long periods of operation.

The proposed architecture and evaluation methodology can be applied to the design of scalable distributed systems in various domains, including smart city infrastructure, IoT platforms, digital government services, and large-scale event processing systems.

Acknowledgements: *This research has been funded by the Science Committee of the Ministry of Science and Higher Education of the Republic of Kazakhstan (Grant No. BR24993051).*

References

1. Schleier-Smith, J., Sreekanti, V., Khandelwal, A. et al. (2021). What serverless computing is and should become: the next phase of cloud computing. Commun. ACM 64, 5, 76–84. https://doi.org/10.1145/3406011.

2. Hassan, H.B., Barakat, S.A. & Sarhan, Q.I. (2021). Survey on serverless computing. J Cloud Comp 10, 39. https://doi.org/10.1186/s13677-021-00253-7.

3. Castro, P., Ishakian, V., Muthusamy, V., & Slominski, A. (2019). The rise of serverless computing. Commun. ACM 62, 12, 44–54. https://doi.org/10.1145/3368454.

4. Shahrad, M., Fonseca, R., Goiri, Í. et al. (2020). Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC 20); 205–218. URL: https://www.usenix.org/system/files/atc20-shahrad.pdf (date of access: 10.03.2026).

5. Vahidinia, P., Farahani, B. J. & Aliee, F.S. (2023). Mitigating Cold Start Problem in Serverless Computing: A Reinforcement Learning Approach. IEEE Internet of Things Journal; 10; 5, 3917-3927. https://doi.org/10.1109/JIOT.2022.3165127.

6. Wen, J., Chen, Z., Sarro, F. & Wang, S. (2025). Unveiling overlooked performance variance in serverless computing. Empirical Softw. Engg; 30; 2. https://doi.org/10.1007/s10664-025-10615-3.

7. Patros, P., Spillner, J., Papadopoulos, A. V. et al. (2021). Toward Sustainable Serverless Computing. IEEE Internet Computing; 25; 6; 42–50. https://doi.org/10.1109/MIC.2021.3093105.

8. Farahani, R., Kimovski, D., Ristov, Sh., Iosup, A., & Prodan, R. (2023). Towards Sustainable Serverless Processing of Massive Graphs on the Computing Continuum. In Companion of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE '23 Companion). Association for Computing Machinery, New York, NY, USA; 221–226. https://doi.org/10.1145/3578245.3585331.

9. Eismann, S., Scheuner, J., Van, Eyk E. et al. (2021). Serverless Applications: Why, When, and How? IEEE Software; 38; 1; 32–39. https://doi.org/10.1109/MS.2020.3023302.

10. Eismann, S., Scheuner, J., van Hoorn A. et al. (2021). The State of Serverless Applications: Collection, Characterization, and Community Consensus. IEEE Transactions on Software Engineering; 1-1. https://doi.org/10.1109/TSE.2021.3113940.

11. Wen, J., Chen, Z., Sarro, F. & Wang, S. (2023). Rise of the Planet of Serverless Computing: A Systematic Review. ACM Transactions on Software Engineering and Methodology; 32; 5; 131; 1 – 61. https://doi.org/10.1145/3579643.

12. Eismann, S., Scheuner, J., van Hoorn, A., Abad, C. L. & Iosup, A. (2021). A Review of Serverless Use Cases and their Characteristics. arXiv. Technical Report: SPEC-RG-2021-1 Version: 1.2. https://doi.org/10.48550/arXiv.2008.11110.

13. Agarwal, S., Krintz, C. & Wolski, R. (2024). Reinforcement Learning (RL) Augmented Cold Start Mitigation in Serverless Platforms. arXiv. https://doi.org/10.48550/arXiv.2308.07541.

14. Castro, P., Ishakian, V., Muthusamy, V. & Slominski, A. (2021). The server is dead, long live the server: Rise of serverless computing, overview of current state and future trends in research and industry. arXiv. URL: https://arxiv.org/pdf/1906.02888 (date of access: 10.03.2026).

15. Golec, M., Walia, G.K., Kumar, M., Cuadrado, F., Gill, S.S. & Uhlig, S. (2023). Cold Start Latency in Serverless Computing: A Systematic Review, Taxonomy, and Future Directions; 1, 34 pages. arXiv. URL: https://arxiv.org/html/2310.08437v2 (date of access: 10.03.2026).

ПИКТІК ЖҮКТЕМЕЛЕР КЕЗІНДЕ ҮЛЕСТІРІЛГЕН АҚПАРАТТЫҚ ЖҮЙЕНІҢ БЕССЕРВЕРЛІК АРХИТЕКТУРАСЫН ҚАЛЫПТАСТЫРУ ЖӘНЕ БАҒАЛАУ ӘДІСТЕРІ

Аңдатпа. *Бұл мақалада біркелкі емес және шың жүктеме жағдайларында жұмыс істейтін таратылған ақпараттық жүйе үшін оқиғаға негізделген серверсіз архитектураны құру тәсілі қарастырылады. Қазіргі заманғы цифрлық қызметтер кіріс сұранысторының қарқындылығының күрт ауытқуларымен сипатталады, бұл операциялық тұрақтылықты, қолайлы жауап беру уақытын және есептеу ресурстарын тез масштабтау мүмкіндігін сақтауды талап етеді. Мұндай ортадағы дәстүрлі монолитті және контейнерлік шешімдер көбінесе алдын ала сыйымдылықты резервтеуді немесе жүктеменің күрт өсуіне кідірістермен жауап беруді талап етеді.*

Бұл зерттеудің мақсаты - сұранысты өңдеу серверсіз есептеу моделін қолдана отырып, тәуелсіз оқиғалар ағыны ретінде ұйымдастырылатын архитектуралық шешімді әзірлеу және эксперименттік түрде бағалау. Осы мақсатта қалыпты жұмыс режимдерін де, қысқа мерзімді шың әсерлерін де қайталауға мүмкіндік беретін формальды оқиға моделі және кіріс жүктемесін генерациялау схемасы ұсынылады.

Тиімділік әртүрлі жүктеме сценарийлеріндегі бірқатар бақыланатын есептеу эксперименттері негізінде бағаланды. Қолданылған негізгі метрикалар орташа өңдеу кідірісі, p95 метрикасы, жүйенің өткізу қабілеті және белсенділіктің жоғарылауы кезеңіндегі қателік деңгейі болды. Зерттеу барысында алынған нәтижелер жүктеме артқан сайын кідірістің артуы басқарылатынын және жүйе қысқа мерзімді шамадан тыс жүктемелер кезінде жұмыс қабілеттілігін сақтайтынын көрсетеді. Бұл бізге ұсынылған тәсілді масштабталатын таратылған қызметтер үшін перспективалы шешім ретінде қарастыруға мүмкіндік береді.

Түйін сөздер: *оқиғалық-бағдарланған архитектура, бессерверлік есептеулер, таратылған ақпараттық жүйелер, масштабталу, пиковалық жүктеме, шеткі кідіріс, өнімділік.*

МЕТОДЫ ПОСТРОЕНИЯ И ОЦЕНКИ БЕССЕРВЕРНОЙ АРХИТЕКТУРЫ РАСПРЕДЕЛЕННОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ В УСЛОВИЯХ ПИКОВЫХ НАГРУЗОК

Аннотация. *В данной работе рассматривается подход к построению событийно-ориентированной бессерверной архитектуры для распределённой информационной системы, функционирующей в условиях неравномерной и пиковой нагрузки. Для*

современных цифровых сервисов характерны резкие изменения интенсивности входящих запросов, это требует сохранения устойчивости работы, приемлемого времени отклика и возможности быстрого масштабирования вычислительных ресурсов. Традиционные монолитные и контейнерные решения в подобных режимах нередко требуют предварительного резервирования мощностей либо реагируют на всплески нагрузки с задержкой.

Цель данного исследования состоит в разработке и экспериментальной оценке архитектурного решения, в котором обработка запросов организована как поток независимых событий с использованием бессерверной модели вычислений. Для этого предложены формализованная модель событий и схема генерации входной нагрузки, позволяющие воспроизводить как штатные режимы работы, так и кратковременные пиковые воздействия.

Оценка эффективности выполнена на основе серии контролируемых вычислительных экспериментов при различных сценариях нагрузки. В качестве основных показателей использованы средняя задержка обработки, метрика p95, пропускная способность системы и доля ошибок в периоды повышенной активности. Полученные в ходе исследования результаты показывают, что при росте нагрузки увеличение задержек носит управляемый характер, а система сохраняет работоспособность при кратковременных перегрузках. Это позволяет рассматривать предложенный подход как перспективное решение для масштабируемых распределённых сервисов.

Ключевые слова: *событийно-ориентированная архитектура, бессерверные вычисления, распределенные информационные системы, масштабируемость, пиковая нагрузка, хвостовая задержка, производительность.*

Сведение об авторах

Касымова Айжан Бахытжановна	PhD, ассоциированный профессор Кафедры «Программная инженерия», Satbayev University, г. Алматы, Казахстан, E-mail: a.kassymova@satbayev.university Алматы, Республика Казахстан.
Ускенбаева Раиса Кабиевна	Д.т.н., профессор Кафедры «Программная инженерия», Satbayev University, г. Алматы, Казахстан, E-mail: r.k.uskenbayeva@satbayev.university,
Young Im Cho	PhD, Professor, Department of Computer Engineering, Gachon University, South Korea. E-mail: yicho@gachon.ac.kr
Элле Венера Жанаткызы	Докторант PhD, Satbayev University, г. Алматы, Казахстан, E-mail: v.elle@satbayev .university
Смаханова Айжан Корганбековна	Докторант PhD, Satbayev University, г. Алматы, Казахстан, E-mail: 840627402313-D@stud.satbayev.university

Авторлар туралы мәлімет

Касымова Айжан Бахытжановна	PhD, К.И. Сәтбаев атындағы Қазақ Ұлттық Техникалық Зерттеу Университеттің «Программалық инженерия» кафедрасының қауымдастырылған профессоры, Алматы қ, Қазақстан, E-mail: a.kassymova@satbayev.university
Өскенбаева Раиса Қабиевна	Техникалық ғылымдарының докторы, К.И. Сәтбаев атындағы Қазақ Ұлттық Техникалық Зерттеу Университеттің «Программалық инженерия» кафедрасының профессоры, Алматы қ, Қазақстан, E-mail: r.k.uskenbayeva@satbayev.university,
Young Im Cho	PhD, Professor, Department of Computer Engineering, Gachon University, South Korea. E-mail: yicho@gachon.ac.kr
Элле Венера Жанатқызы	PhD Докторанты, К.И. Сәтбаев атындағы Қазақ Ұлттық Техникалық Зерттеу Университетті, Алматы қ, Қазақстан, E-mail: v.elle@satbayev .university
Смаханова Айжан Қорғанбековна	PhD Докторанты, К.И. Сәтбаев атындағы Қазақ Ұлттық Техникалық Зерттеу Университетті, Алматы қ, Қазақстан, E-mail: 840627402313-D@stud.satbayev.university

Information about the authors

Kassymova Aizhan	PhD, Associate professor of Software Engeneering Department, Satbayev University, Almaty, Kazakhstan, E-mail: a.kassymova@satbayev.university
Uskenbayeva Raissa	d.t.sc., Professor of Software Engeneering Department, Satbayev University, Almaty, Kazakhstan, E-mail: r.k.uskenbayeva@satbayev.university,
Young Im Cho	PhD, Professor, Department of Computer Engineering, Gachon University, South Korea. E-mail: yicho@gachon.ac.kr
Elle Venera	PhD student, Satbayev University, Almaty, Kazakhstan E-mail: v.elle@satbayev .university
Smakhanova Aizhan	PhD student, Satbayev University, Almaty, Kazakhstan E-mail: 840627402313-D@stud.satbayev.university