UDC: 004.75:004.415.5:004.62 DOI 10.36910/775.24153966.2025.83.10

О.І. Сипягін[1], О. В. Швайкін[2], В. С. Лопухович[3]
магістр (інформаційна безпека), інженер повного стека / інженер з інформаційної безпеки, floLive, вул. Бар-Кохва 21, м. Бней-Брак, Телавівський округ, Ізраїль[1]
Salesforce Developer, VRP Consulting, США, 268 Bush Street #3836, San Francisco, CA 94104[2]
магістр, Senior Software Engineer (contractor), Disney Streaming, 3005 Carrington Mill Blvd, Morrisville, NC 27560, USA[3]

ВПЛИВ СТИСНЕННЯ ДАНИХ НА ЕФЕКТИВНІСТЬ ПРЯМОЇ ВЗАЄМОДІЇ СЕРВІСІВ У МІКРОСЕРВІСНІЙ АРХІТЕКТУРІ

У сучасних мікросервісних архітектурах, що дедалі частіше реалізуються в умовах високої навантаженості, розподіленого середовища та динамічної масштабованості, спостерігається зростання інтересу до моделей прямої взаємодії між окремими сервісами без використання традиційних проксі-рішень – так званий proxyless-підхід. Така модель взаємодії дозволяє зменшити час відгуку системи, уникнути затримок, спричинених проміжною маршрутизацією, оптимізувати інфраструктурні витрати, зменшити кількість потенційних точок відмови й забезпечити більшу архітектурну гнучкість, масштабованість, контроль над потоками даних і відповідність принципам cloud-native. У цьому контексті особливого значення набуває впровадження алгоритмів стиснення даних як одного з ключових засобів оптимізації обміну інформацією між сервісами. Стиснення дозволяє істотно зменшити обсяг переданої інформації, знизити навантаження на мережу, мінімізувати затримки в обробці запитів і сприяти зменшенню ресурсоспоживання при інтенсивному міжсервісному трафіку. У статті здійснено теоретичний аналіз впливу застосування стиснення даних на якість, продуктивність і надійність прямого обміну в межах мікросервісної архітектури. Основна увага зосереджена на безвтратних алгоритмах GZIP і Snappy, що є найуживанішими у cloud-native середовищах із підтримкою REST і gRPC. Проаналізовано особливості їх інтеграції, залежність ефективності від формату і структури даних, типу API-запитів (одиночних чи масових), а також від рівня мережевої латентності та обчислювального навантаження. Окреслено ключові переваги GZIP у випадках високого навантаження й потреби в глибокому стисненні, а також Snappy – при обмеженнях на затримки і пріоритеті швидкодії. Визначено потенційні обмеження щодо сумісності між сервісами, додаткового навантаження на CPU та конфігураційної гнучкості при ручному керуванні параметрами компресії. Підкреслюється важливість забезпечення узгодженості налаштувань на обох кінцях взаємодії, включно з кодуванням заголовків і підтримкою відповідних форматів. Зроблено висновок, що адаптивний і контекстно-залежний підхід до вибору алгоритму стиснення, з урахуванням характеру API, структури навантаження, топології взаємодії, пріоритетів обробки та мережевих характеристик, є критично важливим для забезпечення стабільної, надійної й ефективної роботи мікросервісів у межах proxyless-архітектури сучасних хмарних і гібридних платформ.

Ключові слова: *мікросервіси, proxyless-архітектура, стиснення даних, GZIP, Snappy, затримка, продуктивність сервісів.*

O. Sypiahin, O. Shvaikin, V. Lopukhovych

THE IMPACT OF DATA COMPRESSION ON THE EFFICIENCY OF DIRECT INTERACTION BETWEEN SERVICES IN A MICROSERVICE ARCHITECTURE

In modern microservice architectures, which are increasingly implemented in high-load, distributed environments with dynamic scalability, there is a growing interest in models of direct interaction between individual services without the use of traditional proxy solutions – so-called proxyless approaches. This interaction model reduces system response time, avoids delays caused by intermediary routing, optimizes infrastructure costs, decreases the number of potential points of failure, and ensures greater architectural flexibility, scalability, control over data flows, and compliance with cloud-native principles. In this context, the implementation of data compression algorithms becomes particularly important as one of the key means of optimizing information exchange between services. Compression significantly reduces the volume of transmitted data, lowers network load, minimizes request processing latency, and helps reduce resource consumption during intensive inter-service traffic.

This paper provides a theoretical analysis of the impact of data compression on the quality, performance, and reliability of direct communication within a microservice architecture. The focus is on lossless algorithms such as GZIP and Snappy, which are among the most widely used in cloud-native environments supporting REST and gRPC. The analysis explores the specifics of their integration, the dependence of efficiency on data format and structure, the type of API requests (single or batch), as well as network latency levels and computational overhead. The advantages of GZIP are highlighted for high-load scenarios that require deep compression, while Snappy is preferred in cases with strict latency constraints and a priority on speed. Potential limitations related to service compatibility, CPU overhead, and configuration flexibility when managing compression parameters manually are also identified. The importance of configuration consistency on both ends of service interaction is emphasized, including proper header encoding and support for the required formats.

The conclusion is drawn that an adaptive and context-aware approach to selecting a compression algorithm – considering the nature of the API, the structure of the load, the interaction topology, processing priorities, and network characteristics–is critically important to ensure stable, reliable, and efficient microservice operation within proxyless architectures of modern cloud and hybrid platforms.

Keywords: *microservices, proxyless architecture, data compression, GZIP, Snappy, cloud infrastructure, traffic optimization.*

Problem statement. In the context of the active implementation of cloud solutions and service-oriented logic in the architecture of modern software systems, ensuring the effective interaction of microservices within a distributed environment is of particular relevance. The proliferation of proxyless approaches, which involve the rejection of centralized proxy servers, is aimed at minimizing delays, increasing scalability, and reducing infrastructure maintenance costs [1]. At the same time, such direct interaction between services creates an additional load on the network and the server part of the system, especially in conditions of high-frequency data exchange and limited resources typical for containerized environments [2]. In practice, microservice systems serve significant volumes of requests in real time, often using APIs of external platforms, telemedicine systems, mobile applications, or information gateways with variable bandwidth. In such circumstances, one of the most effective approaches to improving the efficiency of inter-service exchange is the implementation of data compression algorithms that reduce traffic, speed up information transfer, and ensure economical use of resources without losing accuracy [3]. The choice of a suitable algorithm - GZIP, Snappy, or another - depends on the type of data, traffic specifics, and performance requirements. At the same time, the implementation of compression in a proxyless architecture requires careful coordination at the level of service logic, configurations, and interaction protocols [4].

Thus, an urgent scientific and practical problem is to determine the conditions under which data compression provides an increase in efficiency without complicating the architecture, and to establish the relationship between the type of algorithm, the configuration of direct interaction, and the performance of the service system. Research in this area allows us to formulate recommendations for developers on the optimal use of compression in a microservice environment, especially in critical high-load applications.

Analysis of recent research and publications. Actual aspects of designing microservice architecture in the context of containerization and cloud environments are covered in the works of such researchers as Y. Katkov, O. Zinchenko, A. Petrenko, A. Fatima, S. Lakkireddy, Z. Lu, X. Sun, X. Merino. These papers address the issues of scalability, resilience, service design, security of microservices interaction, as well as technical challenges associated with limited resources in modern IT infrastructures. However, despite the existing developments, most of them focus on general aspects of architectural solutions or security in cloud computing, while the specific problem of the impact of data compression on the efficiency of direct (proxyless) interaction between microservices, in particular in the context of intensive load and unstable bandwidth, remains insufficiently studied and requires further systematization.

The purpose of the article is to analyze the impact of data compression algorithms on the efficiency of direct (proxyless) interaction of services in a microservice architecture, to determine the advantages and limitations of implementing compression in conditions of intensive service traffic and limited resources, and to substantiate the technical feasibility of such solutions to improve system performance and scalability.

Summary of the main material. In today's environment of rapid development of digital technologies, microservice architecture has become a key element in building scalable and flexible software systems, especially in environments with high load and limited resources. One of the most pressing issues in this context is optimizing proxyless interaction between services by implementing efficient data compression mechanisms. This issue is especially important in critical applications, such as medical or telemedicine services, where the amount of data transmitted can be significant, and the performance and stability of systems are crucial [1].

The main constructive approach to microservice architecture is to divide a monolithic system into separate independent services that interact with each other via APIs. In cases where inter-service communication is carried out directly, without the use of intermediaries (proxies, API gateways), there is a need for efficient network management, message exchange, and data compression. It is in such conditions that technologies such as GZIP, Snappy, or Brotli can significantly reduce the load on the network and speed up request processing, especially when the frequency of calls is high [4]. Data compression is a technique for reducing the size of information in order to save space or reduce the time it takes to transmit it. In the context of microservices, this means that before sending data from one service to another, this data is first compressed, transmitted over the network in a compressed form, and then decompressed by the recipient back to its original format. This interaction scenario will be the subject of further analysis. The use of compression in microservice architectures offers a number of advantages, including reduced traffic volume and shorter transmission times, which helps improve overall system performance. This allows for efficient exchange of large amounts of data, reducing the load on the network, especially in cases of similar or repetitive traffic. Compression is especially useful during peak loads, when systems simultaneously process a large number of requests, and also increases the efficiency of network resources. At the same

time, the use of compression has certain disadvantages. First, processing compressed data requires additional computing resources on both the client and server sides, which can cause a slight decrease in performance. Secondly, if the algorithms are not set up correctly, data loss or format integrity may occur. In addition, there are potential compatibility issues if services use different compression algorithms. Another difficulty is the need for pre-configuration and additional expertise to work with such data, including debugging and auditing of the transfer.

Implementing compression in microservices environments can reduce network communication latency by up to 30% compared to systems without compression. The greatest effect is achieved in low-bandwidth environments, where even minor optimizations can have a significant impact on overall performance. In telemedicine infrastructures, compression plays an important role in ensuring stable operation of services in remote regions or in conditions of unstable Internet connections [2].

Special attention should be paid to security issues when implementing compression. The use of compression algorithms can potentially create new attack vectors (e.g., CRIME or BREACH) if compression is performed on encrypted traffic. In this regard, it is recommended to apply compression only to unencrypted data, followed by encryption of the communication channel using TLS 1.3 [3].

Another key aspect is the choice of the compression algorithm itself. Depending on the context of use, technical requirements, and the type of data being transmitted, a different approach is appropriate. For example, Snappy, which was developed by Google for fast compression, demonstrates outstanding performance when working with short JSON messages, which is typical for internal cross-service interaction. In turn, Brotli provides a higher compression ratio, but at the expense of greater computational complexity, so it is better to use it in less dynamic scenarios [4], [10]. The comparative characteristics presented in Table 1 allow us to clearly see the advantages of each algorithm depending on the task.

Table 1.

Comparison of compression algorithms for direct interaction of microservices

Type of compression algorithm	Description.	Application examples
Lossless compression	Algorithms that allow you to fully restore the original data after compression.	Huffman coding, Lempel-Ziv (LZ77, LZ78), Deflate (GZIP), Snappy (high speed with moderate compression)
Lossy compression (Lossy)	Algorithms that remove secondary information, so accurate recovery of the original data is impossible.	JPEG (image), MP3, AAC (audio, video)

As you can see from Table 1, each of the algorithms has its own advantages and limitations depending on the purpose of use: GZIP provides a high compression ratio but requires more processing time, while Snappy, on the contrary, has a lower compression ratio but significantly outperforms other algorithms in terms of speed, making it optimal for systems with a high frequency of requests. Brotli combines both approaches, but is more suitable for static content. While lossy formats such as JPEG or MP3 are effective in reducing the amount of multimedia data, their use in microservice interactions is limited due to the inability to fully restore the original data.

In the context of infrastructure, it is important to consider architectural solutions that provide fault tolerance and load balancing. The use of containerized environments, such as Docker and Kubernetes, combined with the concepts of fog/edge architecture, allows for flexible request routing and reduces the overall load on the central nodes of the system [5; 7]. Particularly noteworthy is the implementation of a service mesh, in particular solutions such as Istio or Linkerd, which allow for centralized management of policies for interaction between microservices. Within this architecture, compression policies are implemented as part of the network layer configuration, which greatly simplifies their scaling, maintenance, and subsequent audit [5].

To dynamically adapt to the load, it is advisable to implement multi-agent network management systems that independently adjust the exchange parameters depending on the current situation. This allows you to adaptively enable or disable compression based on predefined metrics (number of requests, average packet size, latency). In combination with machine learning elements, such systems provide an additional level of automation, which is especially important for mobile or unstable environments [6], [8].

In addition, the use of service mesh as a basic infrastructure for microservice interaction allows not only centralized routing management but also efficient load balancing and implementation of quality of service (QoS) policies. Traffic compression in such systems is carried out according to the class of service, which optimizes the use of bandwidth and computing resources without losing the quality of interaction [9].

The issue of architecture compliance with reliability and scalability requirements is one of the key issues in the development of modern digital systems, especially in the context of microservice architecture. The formation of a cloud-native approach, which involves modularity, containerization, automated orchestration, and the ability to scale both vertically and horizontally, allows the system to adapt to changing loads and ensure its fault tolerance even in an unstable environment [4].

To efficiently identify and optimize traffic between microservices in proxyless applications, a comprehensive strategy that combines monitoring, traffic analysis, and data compression algorithms must be implemented. The first step is to monitor and analyze requests: using Prometheus and Grafana, you can collect metrics on the frequency of API calls, the amount of data transferred, and response time. Additionally, the ELK stack (Elasticsearch, Logstash, Kibana) allows you to analyze logs in detail and identify the most loaded or resource-intensive services. The next step is to determine the data format: if the traffic is mainly represented in JSON, XML, or similar formats, it is appropriate to implement Gzip, Brotli, or Snappy. It's important to keep in mind that not all APIs require compression - for example, methods like *getOneEntityById* are best left uncompressed, as testing shows that in such cases, compression creates an unnecessary load. In a proxyless environment, compression is implemented at the level of the services themselves, in particular for unique data formats, using utility classes or wrappers to reduce code reuse. After implementation, performance testing is mandatory to ensure that compression is effective and there is no excessive CPU load. Monitoring should continue, adjusting the compression rules only for really relevant scenarios. In the long term, the compression strategy should be reviewed regularly to take into account changes in technology, data formats, and system requirements.

In a practical context, the issue of implementing data compression in microservice systems is most often solved at the HTTP middleware level or through integration into the transport layer of gRPC communication. However, in the context of direct (proxyless) interaction, where there is no centralized traffic controller, the key challenge is to implement compression on the side of each individual service without disrupting the overall interaction. The experience of using GZIP and Snappy algorithms in such architectures shows that compression not only reduces the size of transmitted packets by 40-70%, but also has a positive effect on the average delay at high request frequency [1], [6].

Using GZIP for REST communication between services allows you to achieve a significant reduction in the amount of data when transferring structured information in JSON format. In a typical scenario, when service A transmits medical or financial data to service B, enabled compression reduces the network load by 55% and the overall response time by 18%. Snappy, on the other hand, demonstrates a lower compression ratio, but is superior in speed, making it suitable for scenarios where minimal latency is critical [2].

In a proxyless infrastructure built using Spring Boot and Kotlin, compression is implemented at the level of filters or interceptors. This allows for compression and decompression of requests and responses before they reach the service's business logic. At the same time, it is necessary to take into account the header encoding (for example, Content-Encoding: gzip) and ensure mutual support for compression at both ends of communication. Failure to comply with these conditions can lead to unpredictable decoding errors or connection disruption [3].

A performance study conducted in the context of modeling a microservice environment with variable latency showed that under average load conditions (200-500 requests per second), GZIP compression reduced response time by ~15%, while Snappy reduced response time by only 8%, but was more stable under peak loads without performance degradation [2], [6]. Thus, the choice of an algorithm depends not only on the desired compression ratio but also on the characteristics of the traffic, data type, and nature of the load.

It should be noted that in conditions of high parallelism of request processing, compression should not block the main threads of execution. Therefore, it is advisable to move compression operations to separate worker threads or use asynchronous mechanisms supported by modern frameworks, such as Kotlin Coroutines or Project Reactor. In combination with the thread-pool configuration, this allows you to maintain performance even in the face of high competition [4].

In addition to technical aspects, the problem of trust between microservices is important, especially in an open architecture or when exchanging data with external partners. Building a trust model for a microservice environment involves not only authentication and authorization, but also checking the integrity of the exchange. In the case of compression, this means using checksums, digital signatures, or HMACs (hash-based message authentication codes) to ensure that the transmitted data has not been altered during transmission [5].

At the same time, service interaction in cloud-fog continuum environments presents additional challenges: in such systems, some services are hosted on edge nodes, where data compression can become too much for the CPU. This is where the need for adaptive logic arises: compression algorithms should only be enabled if certain thresholds of traffic volume or delay are exceeded. This is ensured by telemetry

analysis and orchestration systems such as Kubernetes or OpenShift, which allow distributing the load between nodes and applying compression selectively [7], [8].

The role of architectural design is also key in this process. It is important to consider compression at the stage of service contract design - defining a specification that describes what data is transmitted, in what format, and whether compression is supported. Ignoring this aspect at an early stage can lead to incompatibility between services in the future, which will require costly refactoring or API modifications.

Finally, it's worth noting that compression is not a one-size-fits-all solution, but rather a component of a broader strategy to optimize the direct interaction of services. In some cases (for example, when very short requests or numeric identifiers are transmitted), compression may not make sense or even degrade performance due to the overhead of compression-decompression. Therefore, it is advisable to implement hybrid models where compression is enabled only for large or repetitive data structures, and standard uncompressed exchange is used for the rest of the requests.

Conclusions. The integration of data compression algorithms into the direct interaction of microservices plays a key role in optimizing modern digital architectures in resource-limited environments. Ensuring efficient compression consistent between services can significantly reduce network load, improve system performance and resilience during peak requests, especially in conditions of low bandwidth or unstable Internet connection. Taking into account the specifics of the infrastructure, such as proxyless architecture, mobile edge nodes, and hybrid-cloud environments, requires an adaptive approach to implementing compression. Compression should be designed as a system component, taking into account the service contract, data format support, reconciliation algorithms, and digital security. Implementation of GZIP, Snappy, or Brotli should be accompanied by TLS configuration, integrity control, and policy automation. In the future, it is the flexibility, scalability, and security of compression that will determine the effectiveness of microservice platforms in healthcare, eGovernment, and critical IT infrastructure.

References.

1. Катков Ю. І., Зінченко О. В., Березовська Ю. В., Кладько І. М. Особливості сервіс-дизайну мікросервісної архітектури в хмарних обчисленнях в умовах контейнерної віртуалізації. *Наукові записки Державного університету інформаційно-комунікаційних технологій*. 2024. Вип. 1. С. 47–58. https://doi.org/10.31673/2786-8362.2024.010707

2. Петренко А. І., Болобан О. А. Розробка інфраструктури для інтеграції штучного інтелекту в телемедичні системи за допомогою мікросервісної архітектури. *Таврійський науковий вісник. Серія: Технічні науки*. 2025. Вип. 1. С. 148–158. https://doi.org/10.32782/tnv-tech.2025.1.14

3. Fatima A., Kumar C. K., Panjathan S. U., Doss S. The Security Implications of Microservices in Modern Software Development. In: *Data Governance, DevSecOps, and Advancements in Modern Software*. IGI Global Scientific Publishing, 2025. С. 295–320. https://doi.org/10.4018/979-8-3373-0365-9.ch014

4. Lakkireddy S. Demystifying Cloud-Native Architectures – Building Scalable, Resilient, and Agile Systems. *Journal of Computer Science and Technology Studies*. 2025. Vol. 7, No. 4. P. 836–843. https://doi.org/10.32996/jcsts.2025.7.4.97

5. Lu Z., Delaney D. T., Lillis D. A survey on microservices trust models for open systems. *IEEE Access*. 2023. Vol. 11. P. 28840–28855. https://doi.org/10.1109/ACCESS.2023.3260147

6. Sun X., Cui B., Cai Z. Deep Q-Learning Based Circuit Breaking Method for Micro-services in Cloud Native Systems. In: *CCF Conference on Computer Supported Cooperative Work and Social Computing*. Singapore: Springer Nature Singapore, 2023. P. 348–362. https://doi.org/10.1007/978-981-99-9637-7–26

7. Merino X., Otero C., Nieves-Acaron D., Luchterhand B. Towards orchestration in the cloud-fog continuum. In: *SoutheastCon 2021*. IEEE, 2021. P. 1–8. https://doi.org/10.1109/SoutheastCon45413.2021.9401822

8. Arzo S. T., Bassoli R., Granelli F., Fitzek F. H. Multi-agent based autonomic network management architecture. *IEEE Transactions on Network and Service Management*. 2021. Vol. 18, No. 3. P. 3595–3618. https://doi.org/10.1109/TNSM.2021.3059752

9. Duque A. O., Klein C., Feng J., Cai X., Skubic B., Elmroth E. A qualitative evaluation of service mesh-based traffic management for mobile edge cloud. In: *2022 IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2022. P. 210–219. https://doi.org/10.1109/CCGrid54584.2022.00030

10. Souza A. Software Design and Architecture. In: *Tech Leadership Playbook: Building and Sustaining High-Impact Technology Teams*. Berkeley, CA: Apress, 2024. С. 87–125. https://doi.org/10.1007/979-8-8688-0543-1–4