Scientia Iranica (2024) 31(22), 2088-2095

Sharif University of Technology
Scientia Iranica
Transactions E: Industrial Engineering
https://scientiairanica.sharif.edu

Methods to synchronize data in a microservice architectur

Nikolay Nikolaevich Sheyanov [a, *], Mikhail Petrovich Sinev [a], Darya Olegovna Neshko [b], Dmitry Vladimirovich Pashchenko [a], Dmitry Anatolyevich Trokoz [a], and Alexey Ivanovich Martyshkin [a]

a. Penza State Technological University, 440039, Penza, Russia.
b. Penza State University, 440026, Penza, Russia.

* Corresponding author: fedorovna.sl@mail.ru (N.N. Sheyanov)

Received 13 April 2019; received in revised form 22 February 2022; accepted 27 June 2022

Keywords

Microservice architecture;
Microservice;
Message broker;
Apache Kafka;
State machine;
Finite-state machine.

Abstract

This article discusses the problem of data synchronization methods using microservice architecture. Microservices is a popular and widespread software architecture today. The article investigates three main ways of interaction of microservices. They are event-based communication, interaction through direct HTTP requests and messaging, and also highlights and analyzes their advantages and disadvantages. The main purpose of the article is to analyze and make offer of the optimal option for solving the problem of synchronizing interacting microservices in real time. The optimal solution involves using the Apache Kafka message broker. It publishes data streams and subscriptions to them, as well as stores and processes them. Mathematical modeling of the proposed data synchronization method was described by constructing its state macine, as well as a system of canonical equations.

1. Introduction

At the moment, microservice is the most popular way to develop software. This approach involves splitting the application into separate microservices, each of which fulfills its own separate business purpose. For each microservice, there are the following characteristic requirements [1-6]. For example, it should be small and independent, should fulfill a certain business requirement, interact with other microservices using a pattern of smart endpoints and dumb pipes and also adhere to decentralized management [7]. One of the main parts of the microservices is the interaction of microservices. This means data exchange between them. An important element of the microservice paradigm is the decentralization of data. It is often implemented by allocating its own database for each microservice. This allows you to isolate data from other services, thereby maintaining data stability and security [8]. Let's consider a specific problem: we need to synchronize user data between different microservices. For example, data about the full name, address, contacts and date of birth. At the same time, there are the following requirements. Namely, it should be real time synchronization, the correctness of the data and their delivery to the consumer microservice of the data should be guaranteed [9]. There are several different ways of such exchange, but each of them has its advantages and disadvantages.

The first way is to interact through direct HTTP-requests. At the same time, requests can be executed both synchronously and asynchronously, but it should be understood that using synchronous HTTP-requests, we violate the concept of weak connectivity of services [10]. This method is the simplest in execution, but it is not suitable for scaling, since if several services need to receive the same data from this service, then it will be necessary to implement all communication channels. This approach will require more and more resources to maintain the correct operation of all information channels in case of big number of services in the application (Figure 1).

The second way of communication of microservices is event communication [11]. In this case, a message broker is used for interaction. At the same time, records of events that were produced by separate services are sent between services. Services must explicitly know how to respond to each of the incoming events. Depending on the event, the

To cite this article:
N.N. Sheyanov, M.P. Sinev, D.O. Neshko, D.V. Pashchenko, D.A. Trokoz, and A.I. Martyshkin "Methods to synchronize data in a microservice architectur", *Scientia Iranica* (2024), **31**(22), pp. 2088-2095. https://doi.org/10.24200/sci.2022.53164.3088

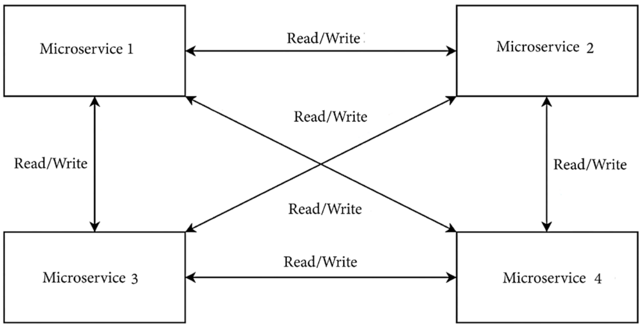

Figure 1. HTTP communication.

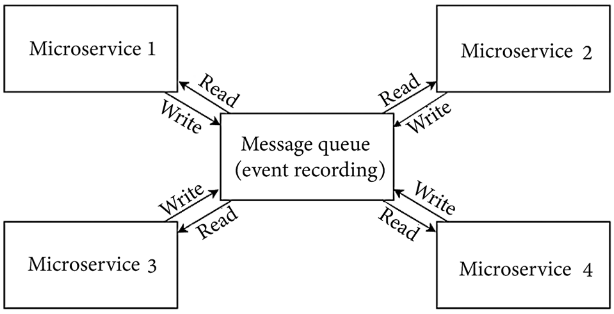

Figure 2. Event communication.

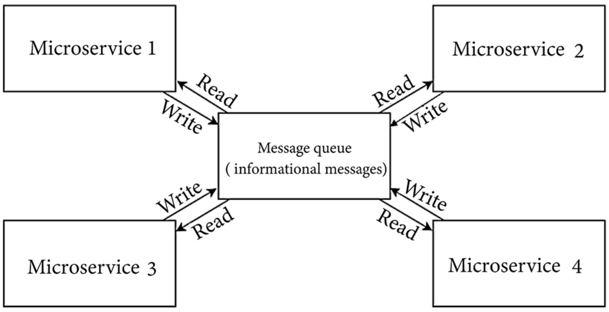

Figure 3. Message communication.

service performs the necessary logic. It allows you to maintain a weak connection between services and at the same time track only significant events. With this method, useful data is not transmitted between services, then using event communication is an impossible solution [12] (Figure 2).

The third method is messaging. In this case, the services do not interact with each other directly, but use message broker is a special mechanism. All services have access to the broker. Broker is the central link between microservices [13]. Services produce data in the corresponding topics, while others receive this data. The messaging approach uses publish-subscribe pattern in which multiple services can receive data from a single publisher [14]. The difficulties of this method are the lack of a guarantee of message delivery [15], as well as the coordination of the structure of the messages sent (Figure 3).

2. Methods

The problem of data synchronization between services can be solved by using optimal messaging method. You may use

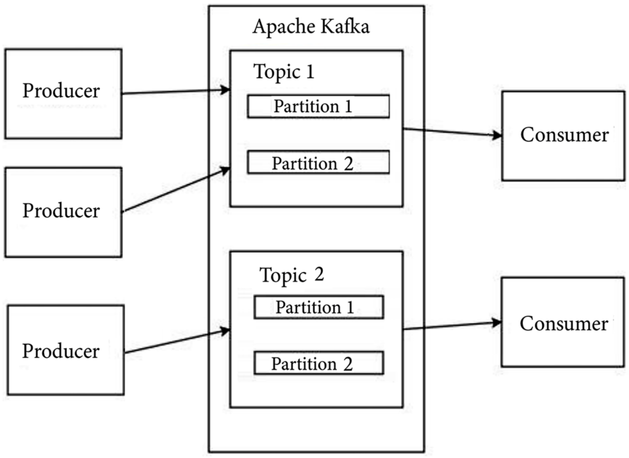

Figure 4. Apache Kafka message broker interaction architecture.

Apache Kafka to minimize the disadvantages of this method (Figure 4) [10]. Apache Kafka is a streaming platform that publishes data streams and subscriptions to them, as well as stores and processes them. Kafka provides huge scaling capabilities and provides a centralized platform for microservices to interact. Another advantage of Kafka is the possibility of customizing the storage time of messages, which ensures replication, integrity and data storage for any period of time. And finally, streaming processing increases the level of abstraction, which in turn allows Kafka to calculate derived streams and datasets dynamically based on data streams [16].

Thus, having considered the main options for microservices interaction, we can conclude that the messaging method is most suitable for solving the problem of data synchronization between different services. At the same time, the best implementation of this method is using message broker [17-20]. For example, Apache Kafka, it provides real time message sending and processing, and can also guarantee message delivery [21].

In order to efficiently and quickly implement an application using a programming language, first you need to build a mathematical model describing the system [22]. In our case, we will use a finite deterministic machine as follows (Figure 5):

S_0 – Initial state of the system;
S_1 – Receiving messages by the consumer;
S_2 – Message package conversion;
S_3 – Converting a specific message;
S_4 – Deleting a message from a package during conversion;
S_5 – Validation of the message package;
S_6 – Validation of a concrete message;
S_7 – Recording a validation error message and deleting a message from a package during validation;
S_8 – Generating a message map for processing;
S_9 – Distribution of message handlers;
S_{10} – Processing a batch of messages;
S_{11} – Processing a concrete message and saving the message data;
x_1 – The presence of messages in the package;
x_2 – The presence of messages for conversion in the package;
x_3 – Successful conversion of a concrete message;
x_4 – Availability of validation messages in the package;
x_5 – Successful validation of a concrete message;
x_6 – Availability of messages to be issued to the handler.

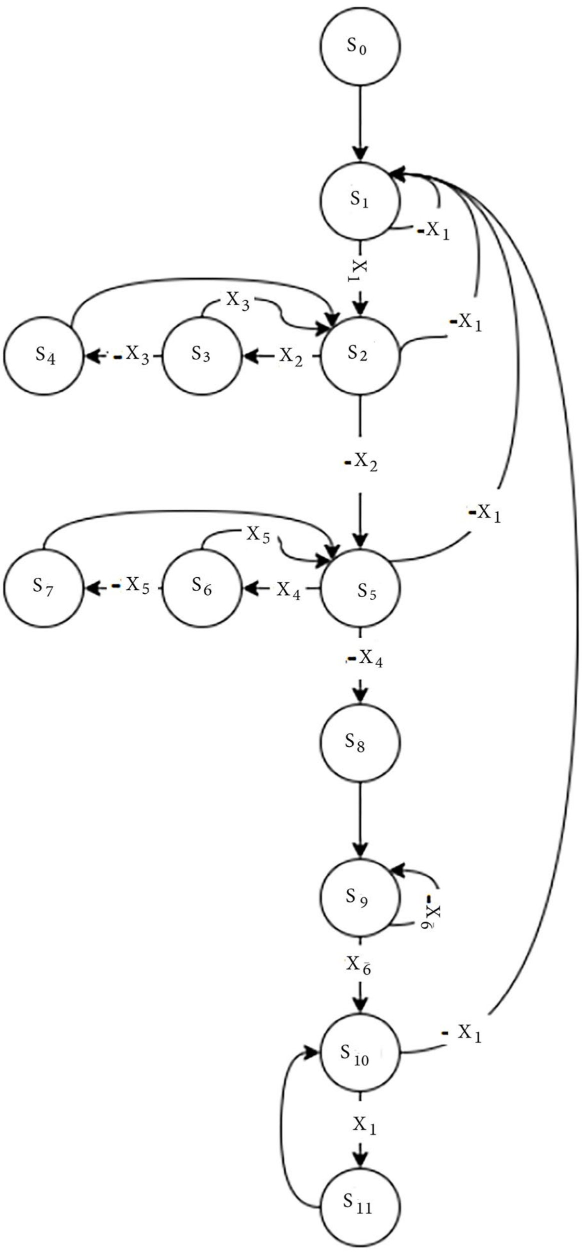

Figure 5. Automatic model of the algorithm for receiving messages by the consumer application.

Let's construct a system of canonical equations for this state machine [23-25]:

Formula 1: System of canonical equations

$$S_1 = S_0 \cup S_1 \overline{x_1} \cup S_2 \overline{x_1} \cup S_5 \overline{x_1} \cup S_{10} \overline{x_1};$$
$$S_2 = S_1 x_1 \cup S_3 x_3 \cup S_4;$$
$$S_3 = S_2 x_2; \qquad S_4 = S_3 x_3;$$
$$S_5 = S_2 x_2 \cup S_6 x_5 \cup S_7;$$
$$S_6 = S_5 x_4; \qquad S_7 = S_6 x_5;$$
$$S_8 = S_5 x_4; \quad S_9 = S_8 \cup S_9 x_6;$$
$$S_{10} = S_9 x_6 \cup S_{11}; \quad S_{11} = S_{10} x_1. \qquad (1)$$

Using the constructed automaton model, we can build an algorithm for processing incoming messages using the Java programming language and the spring framework. Before starting the development of the application, it is necessary to decide and approve exactly what data we should receive from the message broker, in what format this data will be transmitted. After that, it is necessary to develop an algorithm for processing this data in accordance with business requirements and design a database table for their final storage in our system [26-30].

3. Results

The sequence of data synchronization from the producer service to the consumer service consists of the algorithm shown in Figure 6.

Initially, the service producer, in accordance with certain requirements, performs the formation and processing of data on its side. After that, it converts them into JSON format and sends this message to the Kafka broker. The message is saved to the commit log. Each record contains additional information, such as the partition name, topic name, message offset. It allows further manipulation of data collection from the broker. At this time, the service consumer listens to the topics to which he is subscribed for the presence of new messages. From this moment, message processing begins in the consumer service.

As soon as new messages are published, the listener subtracts a message packet, the size of which is configurable, and passes it for processing to the Message Service message processing service. Messages are divided into Linked Hash Map < Integer, Person>, where the key is the global identifier of the person in system, and the value of map is the person with all her data by addresses and contacts. Each message is converted from JSON to person, address, and contact objects. After that, using the Validator class, the person is checked for the correctness of the data, if the data is incorrect, then a corresponding message is output to the log, and this message is not processed. Later, the person's data is placed in the card [31]. This algorithm makes it possible not to process intermediate data of a person within a given message package. This allows you to reduce the number of calculations, because if there are 10 records in the message package concerning the same person, it will be processed only 1 time.

As soon as the final map of all processed messages is formed, it is passed to Thread Pool Task Executor, a class implementing parallel data processing management. Each message from the map is passed to the Message Processor, where it is processed, in parallel with the processing of other messages. The size of the Thread Pool Task Executor is set via the settings file [32-34].

In Message Processor, all further processing of the message takes place, matching the global attributes of the person and the local ones, calculating some parameters, etc. At the end, the final data is stored in the database of our system. As soon as processing of all messages of the packet is finished, the listener is ready to receive a new packet of messages.

In order to get maximum efficiency, we need to choose the most suitable work parameters, both from the point of view of the effectiveness of the solution and from an economic point of view (see Figure 7). Let's consider the main parameters that will appear in the testing process:

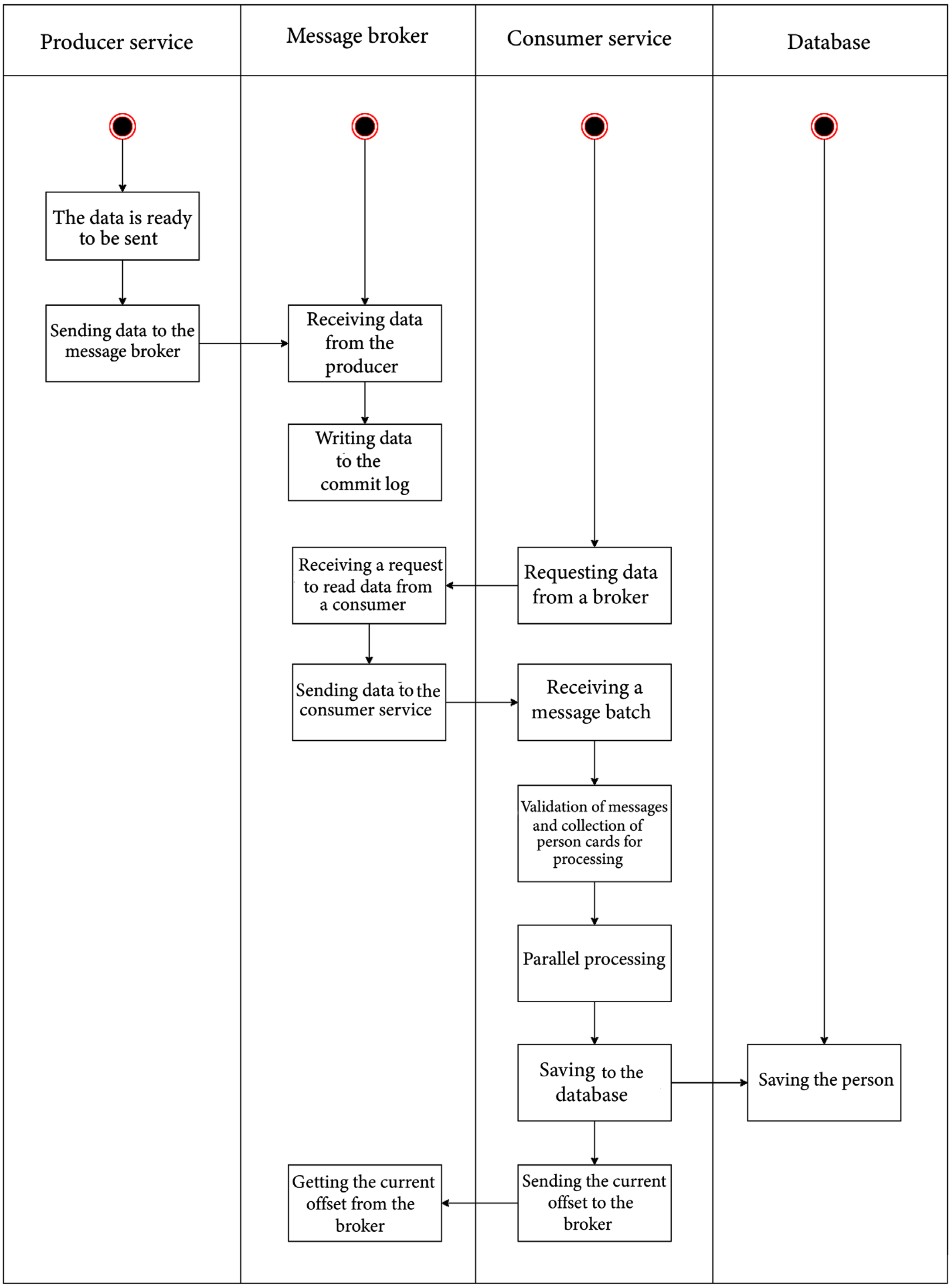

Figure 6. Application activity diagram.

- Number of threads – the number of threads processing a batch of incoming messages;
- Time (ms)–the time taken to process the entire message pool;
- Queue size–the maximum number of messages in the queue (async.max.pool.records parameter);
- Batch size–the number of messages deducted in one poll operation (max.poll.records parameter);
- Relative gain ratio–the performance gain ratio relative to the previous result;
- Absolute gain coefficient–the performance gain coefficient relative to the base value.

In order to calculate the relative and absolute growth rate, we use the following formulas:

Formula 2: Calculation of the relative growth rate

$$K_i^{OTH} = \frac{T_{i-1}}{T_i}, \qquad i \geq 1, \qquad aK_i^{OTH} = 1, \tag{2}$$

where K_i^{OTH} is the relative growth rate; T_i the time spent processing the message pool.

Table 1: Processor messages specification based on proposed method**.**

Number of threads	Time (ms)	Messages	Queue size	Batch size	Relative growth rate	Absolute growth rate
8	1262392	10000	128	512	1,000	1,000
16	636132	10000	128	512	1,984	1,984
32	320190	10000	128	512	1,987	3,943
56	199883	10000	128	512	1,602	6,316
64	162848	10000	128	512	1,227	7,752
128	83221	10000	128	512	1,957	15,169

Formula 3: Calculation of the absolute growth rate

$$K_i^{a\sigma c} = \frac{T_0}{T_i}, \qquad i \geq 1, \qquad aK_i^{a\sigma c} = 1, \tag{3}$$

where $K_i^{абс}$ is the absolute growth rate; T_i the time spent processing the message pool.

We will perform testing with the number of processed messages equal to 10000 (Table 1).

As a result, we get that the best performance under real resource and technology constraints is observed when

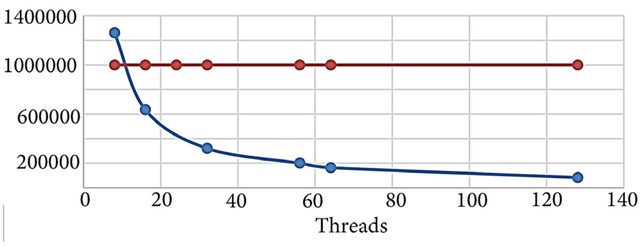

Figure 7. Graph of message processing time depending on the number of handler threads.

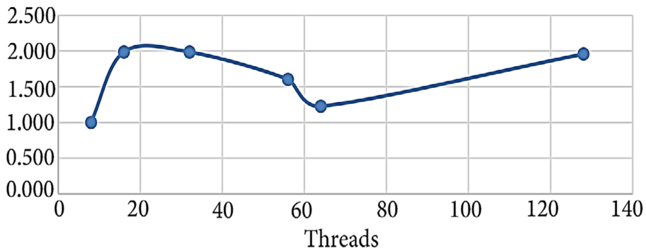

Figure 8. Graph of the relative growth rate.

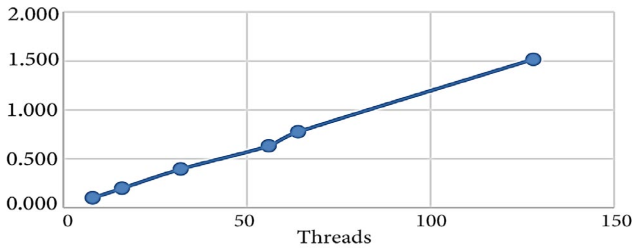

Figure 9. Graph of the absolute growth rate.

working on 64 threads (Figures 8 and 9). At the same time, acceptable performance is also observed when working on 32 threads. Let's consider possible variants of processors on the market for January 2021.

As shown in Figures 10 and 11, thus, the difference in the average cost of processors with 32 and 64 threads is 303,248 rubles, and the ratio of the cost of a processor for 64 threads and 32 is 2.849.

This means that with a 1.602-fold increase in productivity, we spend 2.849 times more in resources. This amount of financial costs is impractical, and therefore it turns out that the best processor option for the task of synchronizing data between services, taking into account the price-performance ratio, is a 32-stream processor. This processor meets the performance limitations and performs the required task in full for the time we need, while it is an adequate option on the part of financial costs for the company.

4. Conclusion

In the course of the research work, the analysis of existing methods of data synchronization in the microservice architecture in real time was carried out. The optimal method for solving the problem was identified, a mathematical model was constructed describing the mechanism of data synchronization using the messaging approach using the Apache Kafka message broker, and a range of parameters acceptable for the effective operation of the application was selected, both from the point of view of system performance and from an economic point of view.

Acknowledgments

The research was conducted at the expense of a grant from the Russian Science Foundation № 21-71-00110, https://rscf.ru/project/21-71-00110/.

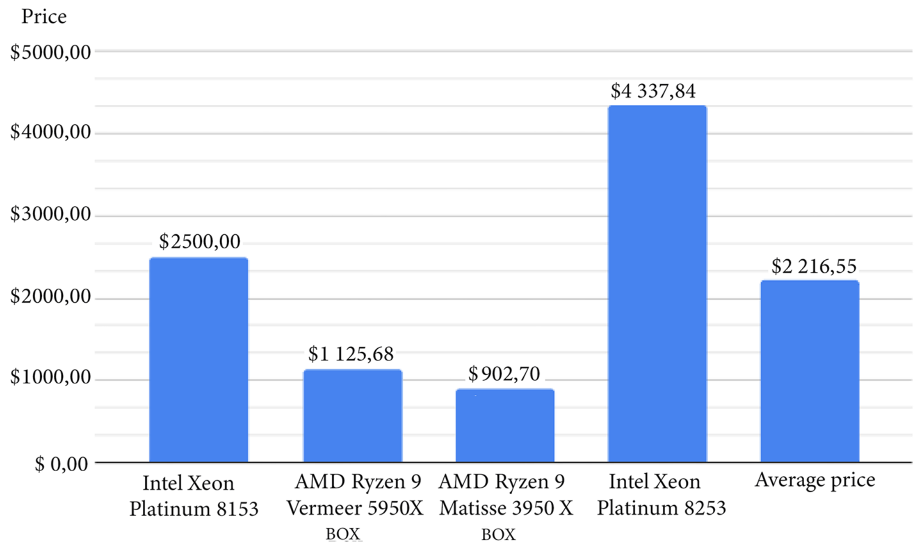

Figure 10. Price chart of processors with 32 threads.

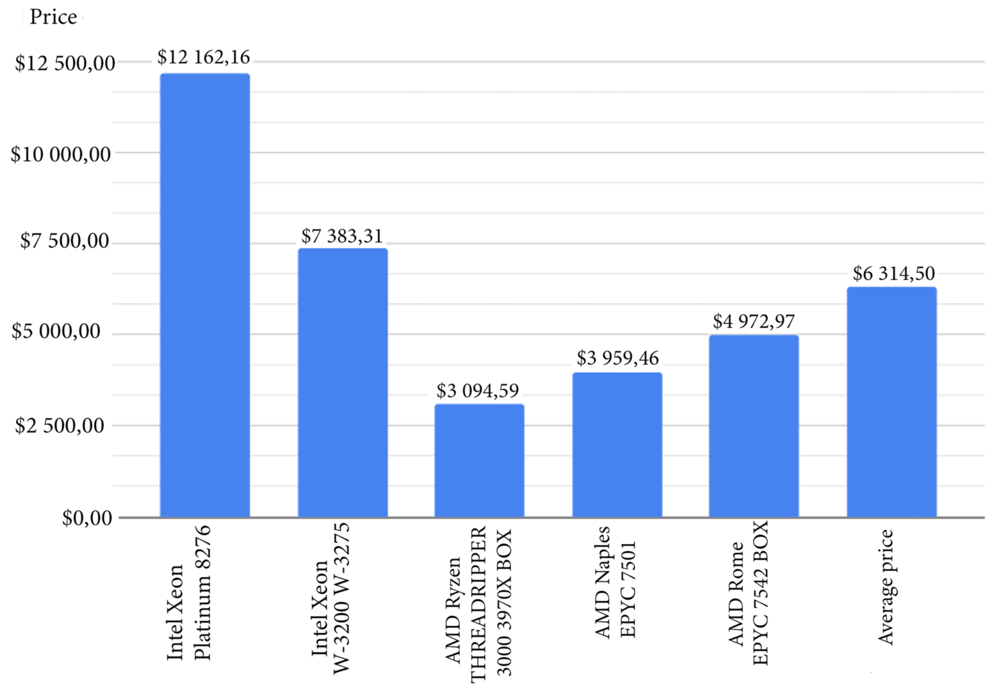

Figure 11. Price chart of processors with 56 or 64 threads.

References

1. Richardson K., *Microservices, Patterns of Development and Refactoring*, St. Petersburg: Peter, p. 544 (2019).
2. Whitesell, S., Richardson, R., and Groves, M.D., *Decentralizing Data. In Pro Microservices*, NET 6 Apress, Berkeley, CA, pp. 137-170 (2022). https://doi.org/10.1007/978-1-4842-7833-8_6
3. Qazani, M.R.C., Asadi, H., Khoo, S., et al. "A linear time-varying model predictive control-based motion cueing algorithm for hexapod simulation-based motion platform", *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, **51**(10), pp. 6096-6110 (2021). DOI: 10.1109/TSMC.2019.2958062
4. Akhir, E.A.P., Bachok, R., Arshad, N.I., et al. "Conceptual framework for SIDS alert system", *4th International Conference on Computer and Information Sciences (ICCOINS)*, pp. 1-5, IEEE (2018). DOI: 10.1109/ICCOINS.2018.8510596
5. Modoni, G.E., Caldarola, E.G., Sacco, M., et al. "Synchronizing physical and digital factory: benefits and technical challenges", *Procedia Cirp*, **79**, pp. 472-477 (2019). https://doi.org/10.1016/j.procir.2019.02.125
6. McNally, B., Lu, Y., Shively-Ertas, E., et al. "A simple and effective methodology for generating bounded solutions for the set K-covering and set variable K-covering problems: A guide for or practitioners", *Review of Computer Engineering Research*, **8**(2), pp. 76–95

(2021). https://doi.org/10.18488/journal.76.2021.82.76.95

7. Girrbach, P. "The metaphorical culturalistic approach to technology assessment", *Tehnički Glasnik*, **15**(4), pp. 554-561 (2021). DOI:10.31803/tg-20210628194103
8. Al-Masri E. and Mahmoud Q.H. "A broker for universal access to web services", *Seventh Annual Communication Networks and Services Research Conference*, pp. 118-125 (2009). DOI: 10.1109/CNSR.2009.27
9. Qazani, M.R.C., Asadi, H., Bellmann, T., et al. "Adaptive washout filter based on fuzzy logic for a motion simulation platform with consideration of joints' limitations", *IEEE Transactions on Vehicular Technology*, **69**(11), pp.12547-12558 (2020). DOI: 10.1109/TVT.2020.3023478
10. de Toledo, S.S., Martini, A., and Sjøberg, D.I. "Identifying architectural technical debt, principal, and interest in microservices: A multiple-case study". *Journal of Systems and Software*, **177**, 110968 (2021). https://doi.org/10.1016/j.jss.2021.110968
11. Pashchenko, D.V., Jaafar, M.S., Zinkin, S.A., et al. "Directly executable formal models of middleware for MANET and cloud networking and computing", *J. Phys. Conf. Ser.,* **710**(1), p. 12024 (2016), DOI: 10.1088/1742-6596/710/1/012024
12. Qazani, M.R.C., Asadi, H., Mohamed, S., et al. "An optimal washout filter for motion platform using neural network and fuzzy logic", *Engineering Applications of Artificial Intelligence*, **108**, 104564 (2022). https://doi.org/10.1016/j.engappai.2021.104564
13. Narhid, N., Shapira, G., and Pavlino, T. "Apache Kafka. Streaming Data Processing and Analysis", *St. Petersburg: Peter*, **320** (2019). DOI: 10.1109/ICICET.2018.8533771
14. Irandoost, A. and Kargar, S. "An integrated optimization of routing and scheduling of liner ships in offshore logistics management", *Journal of Research in Science, Engineering and Technology*, **9**(02), pp. 17-35 (2021). DOI: 10.24200/jrset.vol9iss02pp17-35.
15. Sun, X., Liang, Y., and Huang, H. "Design and implementation of internet of things platform based on microservice and lightweight container", *IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, **9**, pp. 1353-1357 (2020). DOI: 10.1109/itaic49862.2020
16. Božić, D. "Applying Simulation Modelling in Quantifying Optimization Results", *Tehnički Glasnik*, **15**(4), pp. 518-523 (2021). https://doi.org/10.31803/tg-20210326111551
17. Nagothu, D., Xu, R., Nikouei, S.Y., et al. "A microservice-enabled architecture for smart surveillance using blockchain technology", *IEEE International Smart Cities Conference (ISC2)*, pp. 1-4 (2018). 10.1109/ISC2.2018.8656968
18. Seng, J.L. and Chen, T.C. "An analytic approach to select data mining for business decision", *Expert Systems with Applications*, **37**(12), pp. 8042-8057 (2018). https://doi.org/10.1016/j.eswa.2010.05.083
19. Chang, C.I., *Hyperspectral Data Processing: Algorithm Design and Analysis*, John Wiley and Sons, London, UK (2013). DOI: 10.1002/9781118269787
20. Jia, C., Tan, C.Y., and Yong, A. "A grid and density-based clustering algorithm for processing data stream", *Second International Conference on Genetic and Evolutionary Computing,* IEEE, pp. 517-521 (2008). DOI: 10.1109/WGEC.2008.32
21. Srinivasareddy, S., Narayana, Y., and Krishna, D. "Sector beam synthesis in linear antenna arrays using social group optimization algorithm", *National Journal of Antennas and Propagation*, **3**(2), pp. 6-9 (2020). DOI: 10.31838/NJAP/03.02.02
22. Prokofiev, O.V. and Savochkin, A.E. "Additive noise effect on the error of time interval forming", *Proceedings Global Smart Industry Conference, GloSIC*, pp. 255–258, 9267820 (2020). DOI: 10.1109/GloSIC50886.2020.9267820
23. Deinum, M., *Java Enterprise Services, Spring Boot 2 Recipes*, Apress, Berkeley, pp. 239-256 (2018). https://www.springerprofessional.de/en/spring-boot-2-recipes/16301616
24. Esmaeeli, J., Amiri, M., and Taghizadeh, H. "A new approach in the DEA technique for measurement of productivity of decision-making units through efficiency and effectiveness", *Scientia Iranica*, **30**(2), pp. 822-832 (2023), DOI: 10.24200/sci.2020.54858.3961.
25. Salehi, S.M., Farrahi, G.H., and Sohrabpour, S. "A new technique of the first and second limits", *Scientia Iranica*, **24**(3), pp. 1171-1180 (2020). 10.24200/sci.2017.4098
26. Sarvestani, E. and Khayati, G.R. "An integrated model for predicting the size of silver nanoparticles in montmorillonite/chitosan bionanocomposites: A hybrid of data envelopment analysis and genetic programming approach", *Scientia Iranica*, **28**(3), pp. 1871-1883, (2021). DOI: 10.24200/sci.2020.52510.2748
27. Dhivya, S., Iswariyalakshmi, B., Banumathi, V., et al. "Image integration with local linear model using demosicing algorithm". *International Journal of Communication and Computer Technologies*, **5**(1), pp. 36-42 (2023). https://ijccts.org/index.php/pub/article/view/60
28. Barzamini, H. and Ghassemian, M. "Comparison analysis of electricity theft detection methods for advanced metering infrastructure in smart grid", *International Journal of Electronic Security and Digital Forensics*, **11**(3), pp. 265-280 (2019). DOI:10.1504/IJESDF.2019.10020548
29. Alkawaz, M.H., Veeran, M.T., and Bachok, R. ``Digital image forgery detection based on expectation maximization algorithm'', In 2020 16th *IEEE International Colloquium on Signal Processing and Its Applications (CSPA)*, pp. 102-105, IEEE (2020). DOI: 10.1109/CSPA48992.2020.9068731
30. Dahmardeh, H., Zareh, M., Mirzadeh, A., et al. "Brain emotional learning basic intelligent control for congestion control of TCP networks", *J Basic Appl Sci Res*, **3**(1), pp. 345-349 (2013).
31. He, S., Zhao, L., and Pan, M. "The design of inland river ship microservice information system based on spring cloud", *5th International Conference on Information Science and Control Engineering (ICISCE),* pp. 548-551 (2018). DOI: 10.1109/ICISCE.2018.00120
32. Elfaki, A.O., Abouabdalla, O.A., Fong, S.L., et al. "Review and future directions of the automated

validation in software product line engineering", *Journal of Theoretical and Applied Information Technology*, **42**(1), pp. 75-93 (2012). https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=8e587c553d8c2ed6133a929a6712718b38ed7c73

33. Arzo, S.T., Scotece, D., Bassoli, R., et al. "MSN: A playground framework for design and evaluation of micro services-based sdN controller", *Journal of Network and Systems Management*, **30**(1), pp. 1-31 (2022). https://doi.org/10.1007/s10922-021-09631-7
34. Ahmadi, Z., Haghighi, M., and Validi, Z. "A novel approach for energy optimization in distributed databases in wireless network applications", *Journal of Management and Accounting Studies*, **8**(3) (2020).

Biographies

Nikolay Nikolaevich Sheyanov is the student of the Department of programming, He works in Penza State Technological University, 440039, Russia, Penza, BaydukovProyezd / Gagarin Street, 1a/11. The main field of activity are: development of parallel algorithms for expert systems; hardware and software implementation of high-performance computing systems.

Mikhail Petrovich Sinev is Associate Professor of Programming Department, Penza State Technological University, 440039, Russia, Penza, BaydukovProyezd/ Gagarin Street, 1a/11. The main field of his activity are development of parallel algorithms for expert systems; hardware and software implementation of high-performance computing systems.

Darya Olegovna Neshko is student of the Department of Computer Engineering, Penza State University, 440026, Russia, Penza, Krasnaya Street, 40. The main field of activity are development of parallel algorithms for expert systems; hardware and software implementation of high-performance computing systems.

Dmitry Vladimirovich Pashchenko is the Rector of Penza State Technological University, Penza State Technological University,440039,Russia,Penza,BaydukovProyezd/Gagarin Street, 1a/11. The main areas of activity are modeling and formalization of models of multi-threaded computing systems using the mathematical apparatus of Petri nets; development of parallel algorithms for expert systems; hardware and software implementation of high-performance computing systems.

Dmitry Anatolyevich Trokoz is Vice-Rector for scientific work of Penza State Technical University. He is in the Penza State Technological University, 440039, Russia, Penza, BaydukovProyezd / Gagarin Street, 1a/11. The main areas of activity are modeling and formalization of models of multi-threaded computing systems using the mathematical apparatus of Petri nets; development of parallel algorithms for expert systems; hardware and software implementation of high-performance computing systems.

Alexey Ivanovich Martyshkin is Head of the Programming Department of Penza State Technical University. The main areas of activity are modeling of models of multi-threaded and multiprocessors computings systems using the mathematical apparatus of systems and Queuing networks; development of parallel algorithms for expert systems; hardware and software implementation of high-performance computing systems.