Journal of Theoretical and Applied Information Technology
15th October 2019. Vol.97. No 19
© 2005 – ongoing JATIT & LLS

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

STUDYING SYSTEMS OF OPEN SOURCE MESSAGING

ALEKSANDER BONDARENKO, KONSTANTIN ZAYTSEV

National Research Nuclear University MEPhI (Moscow Engineering Physics Institute),
Kashirskoe Avenue 31, Moscow, 115409, Russia

ABSTRACT

Modern large industrial and financial structures apply numerous various information systems (IS) which exchange data while communicating with each other. In order to implement such communication nowadays, specialized messaging systems are used or transport components comprised of one or several software products. This article compares four open source software products used in messaging systems: Apache Kafka, gRPC, ZeroMQ, and RabbitMQ, which satisfy criteria of Secure Sockets Layer/Transport Layer Security (SSL/TLS) encryption and possibility to operate directly with Java platform applications, that is, to provide Java API. In order to perform these studies, comparison environment was generated with four coordinates: supported communication type, productivity, reliability, and community support.

Keywords: *Open Source Systems, Apache Kafka, Grpc, Zeromq, Rabbitmq, Messaging, Publish&Subscribe, RPC, Streaming.*

INTRODUCTION

With the increase in applied IS, the scope of message exchange also increases, hence, aiming at unification and standardization of exchange, it is required to develop specialized interconnecting systems. In modern world, where Internet of Things (IoT) exerts significant influence on global economy by amount of investments (about $6 trillion) and is already implemented into global banking, it would be reasonable to consider capabilities of various interconnecting software products [1]. In general, interconnecting systems use certain patterns of data exchange, they should be selected prior to selection of protocols, communication methods, and auxiliary infrastructure of developed system. This recommendation is obvious: if it has not been done in advance, then in the course of IS development, while modifying interconnecting systems, it would be required to modify code, architecture, safety model, and its communications with external world [2].

Communication pattern should be selected using several approaches based on message-oriented middleware. At present messaging on the basis of enterprise service bus (ESB) is widely used, it is based on the principles of service-oriented architecture (SOA) [3]. Such approach proved its efficiency during recent years, however, with increased number of transferred data, number and variety of transactions the drawbacks of such approach become obvious, and the developers start to search for more promising and modern technologies [4, 5, 6].

Some crediting and financial institutions start development of intermodular communication for platform architecture. Intermodular communication is based on any messaging system, whether it is message broker or transport library. In large software products, it is also required to implement several communication types focusing on messaging rate or high reliability of delivery. This article compares messaging systems based on publish/subscribe communications [7] and point-to-point communications by means of remote procedure call [8].

Messaging as a basic process of integration of large information systems is always a relevant subject for research. A lot of research has been carried out, characteristic and relevant for its time, studying the methods of exchange and transmission, ready-made software solutions at different stages of the evolution of messaging systems. For example, in the articles [9, 10], the authors evaluate the performance of the ESB architecture, including open source, and in the article [11] the performance of the middleware as a whole. A later study [12, 13] evaluates more modern software solutions, taking into account the requirements of distributed systems and working with big data, such as Apache Kafka and Rabbit MQ. In [14, 15], a comparative analysis of RabbitMQ vs ActiveMQ vs OpenMQ was carried out. Thus, due to the long existence and importance of the messaging process, this topic has been deeply studied. Since every year the IT industry poses new challenges in terms of data volume, reliability and speed of their transmission, new systems appear, existing ones are updated, which leads to the need for repeated research.

With the growth of data volume and complexity of structures, the question arises of the reliability and speed of data delivery for further analysis and processing. Modern solutions with an innovative approach and improved algorithms come to the market of messaging systems to implement reliable, safe and fast message delivery, displacing traditional solutions, for example, based on message queues and the principles of the corporate service bus. In this work, in a comparative analysis, among the software solutions there are such modern software products as Apache Kafka and gRPC that meet the modern challenges of the IT industry.

The research hypothesis is as follows: as a result of comparative analysis, the Apache Kafka software product is a leader in the group of systems that implement Publish / Subscribe interactions; gRPC is a leader in a group of systems that implement point-to-point interactions.

After reading this article, readers will find out what promising software solutions exist for messaging with different types of interaction and the results of a comparative analysis of these solutions by significant criteria for building integration interactions in large IT structures.

2.MATERIALS AND METHODS

2.1. Selection of software for comparison

Software products for analysis were selected with accounting for three main criteria:

- product should support SSL/TLS encryption;
- product should communicate directly with Java platform applications (provision of Java API);
- it should be open source product.

Initial selection of software products was based on analysis of publications about messaging systems. Then, using the mentioned criteria, only appropriate products were selected, i.e., the following messaging systems: Apache Kafka [16], Rabbit MQ [17], ZeroMQ [18], and gRPC [19]. The degree of compliance of the mentioned open source products with the highlighted criteria is summarized in Table 1.

Table 1. Compliance of software products with selected criteria

System / Criterion	Apache Kafka	Rabbit MQ	ZeroMQ	gRPC
SSL/TLS encryption	+	+	+/- (CurveZMQ required)	+
Java API	+	+	+/- (binding jzmq)	+
Open Source	+	+	+	+

2.2. Selection of space coordinates for comparison

The most important characteristic of any software product is the scope of functions which can be performed by this product, and productivity, that is, the scope of performed work per unit time. Banking integrated software also requires for reliability in order to store data in the best way, and community support demonstrating relevance of solution regarding modern trends facilitating competition with existing financial and engineering companies.

Taking this into account, the following comparison coordinates of software products were selected:

1) supported communication types;
2) productivity;
3) reliability;
4) community support.

Apache Kafka and RabbitMQ are separate brokers with midway center, mainly with Publish/Subscribe messaging strategy, and ZeroMQ and gRPC are embedded third-party libraries with point-to-point strategy. Therefore, it would be incorrect to compare all aforementioned products in terms of productivity and reliability due to peculiar features of their architecture. Moreover, since the second group is comprised of libraries, it would be incorrect to discuss their reliability and guarantee of message delivery since this is the task of brokers from the first group. While using libraries, developers of modules should pay attention to reliability of message delivery, the libraries only provide good rate, minimum latency, and user-friendly API. In this regard it was decided to carry out primary comparison of supported communication types between all software products, comparison in terms of productivity and community support in Publish/Subscribe and Point-to-Point groups; and reliability had to be analyzed only in the scope of the first group of products. As a consequence, it would be possible to determine optimum pair of software products covering both communication types.

Sum of estimates for each product was calculated according to Eq. (1).

$$S = 100 \cdot \sum_{i=1}^{n} \omega_i \cdot Z_i , (1)$$

where S was the estimate of tool; n was the number of comparison criteria; Z_i was the value of criterion compliance; ω_i was the criterion weight (from 0 to 1).

The following scale has been proposed for criterion estimation: 0 - if criterion is not complied with or requires for third party software; 1 - if criterion is complied with completely or with minor restrictions; 0.5 - if criterion is complied with significant restrictions.

2.3. Comparison in terms of supported type of communication

In order to compare in terms of this coordinate, it was decided to study in detail specifications of each software and various publications devoted to nonstandard application of software product (Publish/Subscribe for Point-to-Point and vice versa).

Implementation of each communication type has been assessed using the following scale:

- 0, if implementation is impossible;
- 0.5, if implementation requires for significant adjustment of system modules or requires for existence of supplemental/auxiliary library/software;
- 1, if implementation is absolutely possible.

Cumulative estimate has been calculated by Eq. (1), where S is the cumulative estimate of messaging system; n is the number of communication types; Z_i is the estimate of possibility to implement communication; ω_i is the criterion weight (from 0 to 1).

2.4. Comparison in terms of productivity

Productivity is one of the key properties of messaging systems, it is important for integration of corporate applications, especially in low latency solutions. Message brokers Apache Kafka and RabbitMQ were compared according to the following scenario: one broker of each type was established, productivity was tested using specialized utilities of load testing. For Kafka, reading and recording were carried out from/to one topic with one partition, for RabbitMQ – from/to one queue. Messages were read using 10 Java streams. At the first stage, the message size was fixed and then, varying number of messages, the reading/writing rates were measured as well as latency (average, maximum). Then, the message size was increased, and new measurement of varying number of messages was performed.

Transport libraries for remote procedure call in ZeroMQ and gRPC were compared as follows:

ZeroMQ was tested using native tool for load testing by library developers in the command line form. Messages of various size in the range from 1 byte to 512 Mbytes were generated, throughput capacity and latency were measured. gRPC was tested using specialized tool for load testing: ghz, also in command line form. Messages of various size in the range from 1 Kbyte to 64 Kbytes were generated, throughput capacity and latency were measured for two communication subtypes: simple remote procedure call and streaming.

2.5. Comparison in terms of reliability

Apache Kafka and RabbitMQ provide reliability and guaranties of message delivery in different ways. Table 2 shows which tools are used by the software products to guarantee message delivery.

Table 2. Message delivery guarantees

Apache Kafka	**RabbitMQ**
Message durability — messages stored in segment are not lost;	Message reliability — they will not be lost while stored on RabbitMQ;
Message notifications — signal exchange between Kafka (possibly, Apache Zookeeper store), on the one hand, and producer/consumer, on the other hand.	Message notifications — RabbitMQ exchanges signals with producers and consumers.

One of the main indices of reliability is support of various guaranties of message delivery, they can be of three types:

- at-most-once delivery. It means that the message cannot be delivered more than once. In addition, the message could be lost.
- at-least-once delivery. It means that the message will never be lost. In addition, the message could be delivered more than once.
- exactly-once delivery. All messages are delivered strictly once.

2.6. Comparison in terms of community support

The comparison was based on such criteria as the number of stars of project in GitHub; the number of forks directly indicating interest of third party developers to the considered software; the number of commits to main repository; the number of releases; the amount of contributors: active community committers; the interest of large companies to projects and time from the last release of stable version. With the aim of reference, the date of project start was also mentioned, however, this information was not used.

2.7. Generalization of results

Ranks were assigned to the software products according to the results of each comparison. Thus, Rank 1 was assigned to the best software, |then, Rank 2, etc.; if one and the same rank was assigned to several products, then the rank was determined by averaging equation (2)

$$r = \frac{\sum_{i=0}^{n-1}(r' + i)}{n}, (2)$$

where r was the final rank of tools; n was the number of tools with one and the same rank; r' was the rank assigned to all tools.

Comparison results were generalized by summing ranks assigned to the tools in all comparisons and then by ranking of the obtained cumulative ranks.

The criteria for interpreting the results are the ranks assigned to each system in the context of each comparison and generalized comparison results: the lower the rank, the better the system.

3. RESULTS

3.1. Comparison in terms of supported communication types

Software comparisons in terms of supported communications are summarized in Table 3.

Table 3. Comparison in terms of supported communication types

Interaction type		Weight	Apache Kafka	Rabbit MQ	ZeroMQ	gRPC
RPC / Point-to-point	synchronous	1/3	0.5	0.5	1	1
	asynchronous	1/3	0.5	0.5	1	1
Publish / Subscribe		1/3	1	1	0	0.5
Sums of estimates			66.67	66.67	66.67	83.33

3.2. Comparison in terms of productivity

Software comparisons in terms of productivity are summarized in Tables 4–7.

Table 4. Productivity of Apache Kafka.

Test scenario		Recording		Reading		Latency	
Size, Kb	Amount	Messages/s	Mb/s	Messages/s	Mb/s	average	max
1	10,000	13,106.1599	12.5	28,735.63	27.4044	205.56	387
	100,000	36,697.24771	35	120,918.9843	115.3173	617.75	838
	300,000	28,403.71142	27.09	185,299.5676	176.7154	1,065.61	2,838
100	1,000	245.158127	23.38	1,694.9153	161.6397	1,202.94	2,877
	10,000	294.602875	28.1	3,996.8026	381.1648	1,125.8	5,620
	100,000	269.065294	25.66	5,442.7693	519.0629	1,247.17	5,765

Recording rate increases nonlinearly with the amount of sent megabytes, and the reading rate increases nearly proportionally to the amount of sent megabytes. In addition, the higher is the load, the higher is the latency.

Table 5. Productivity of RabbitMQ

Test scenario		Recording		Reading		Latency	
Size, Kb	Amount	Messages/s	Mb/s	Messages/s	Mb/s	average	max
1	10,000	8,688	8.484375	8,688	8.484375	77.5	154
	100,000	14,467	14.12792969	14,467	14.12792969	106	211
	300,000	16,143	15.76464844	16,143	15.76464844	95.5	190
100	1,000	619	60.44921875	2,060	201.171875	48.5	94
	10,000	274	26.7578125	3,958	386.5234375	78.5	152
	100,000	248	24.21875	2,012	196.484375	60.5	118

At reading rate of 1 Kb, message RabbitMQ is inferior to Kafka by nearly two times, however, in the case of 100 Kb, the rate is nearly the same, and in the case of 1000 messages, Rabbit is by far superior than Kafka. However, in most measurements the reading rate of Apache Kafka is higher despite that the latency is also higher.

Therefore, despite high productivity of both brokers, Apache Kafka is more productive than Rabbit MQ; let us assign Ranks 1 and 2 to them, respectively.

Table 6. Productivity of ZeroMQ

Message size	**Latency, ns**	**Throughput capacity, message/s**	**Throughput capacity, Mb/s**
1 B	33.566	7,305,826	58.447
2 B	33.903	6,401,719	102.428
4 B	33.877	6,745,770	215.865
8 B	34.573	6,884,214	440.590
16 B	34.064	6,295,711	805.851
32 B	35.218	4,677,759	1,197.506
64 B	35.736	4,767,554	2,440.988
128 B	35.775	3,885,802	3,979.061
256 B	35.994	2,689,235	5,507.553
512 B	35.932	1,598,083	6,545.748
1 kB	36.836	867,274	7,104.709
2 kB	47.187	407,486	6,676.251
4 kB	44.569	221,717	7,265.223
8 kB	54.324	110,846	7,264.403
16 kB	79.018	54,030	7,081.820
32 kB	93.768	33,698	8,833.729
64 kB	153.736	16,934	8,878.293
128 kB	194.159	8,611	9,029.288
256 kB	312.330	4,377	9,179.234
512 kB	518.899	2,184	9,160.360
1 MB	1,130.965	1,100	9,227.469
2 MB	2,083.748	544	9,126.806
4 MB	3,747.207	269	9,026.142
8 MB	7,212.642	135	9,059.697
16 MB	13,607.344	67	8,992.588
32 MB	27,424.148	33	8,858.370
64 MB	55,334.758	17	9,126.806
128 MB	113,366.031	8	8,589.935
256 MB	222,521.812	4	8,589.935
512 MB	444,433.406	2	8,589.935

Table 7. Productivity of gRPC

Interaction type	Message size	Latency, ns	Throughput capacity, Mb/s
Simple	1kB	40.50	50.50
	2 kB	94	111
	4 kB	148.50	150
	8 kB	202	214.50
	16 kB	250	256
	32 kB	311	307.50
	64kB	365	359
Stream	1kB	22.70	90.30
	2 kB	65	151.40
	4 kB	110	206.50
	8 kB	160.30	267
	16 kB	206	320
	32 kB	250	378.80
	64kB	298	439

Therefore, it is obvious that ZeroMQ is more efficient in terms of both throughput capacity and latency in comparison with gRPC, which can be attributed to the fact that the library uses pure TCP protocol and works over sockets whereas gRPC operates according to more complicated http/2 protocol which results in overhead. ZeroMQ is more productive than gRPC, let us assign Ranks 1 and 2 to them, respectively. Software ranking in terms of productivity is illustrated in Fig. 1.

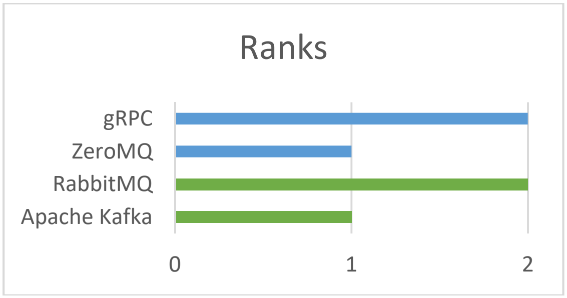

Fig. 1. Ranking of products in terms of productivity.

3.3. Comparison in terms of reliability

RabbitMQ is distinguished from Kafka by usage of batches in messaging. RabbitMQ provides something similar to batching due to:

- suspending each X messages until all notifications are received. RabbitMQ usually groups notifications using "multiple" flag;
- assigning "prefetch" parameter by receivers and grouping notifications using "multiple".

Nevertheless, messages are not sent in batches. This looks like continuous stream of messages and sending notification groups in one message tagged with "multiple" as in TCP protocol. Kafka provides more obvious batching of messages.

In order to prevent failures, Kafka is provided with master–slave architecture at the level of log section, in this architecture masters are referred to as leaders, and the slaves can be referred to as replicas. Leader of each segment can have several slaves. If server with leader fails, then it is assumed that replica becomes leader, all messages are retained, only handling is terminated for a short time.

Kafka prefers the concept of In Sync Replicas (ISR). Each replica can be or not be in synchronized state. In the first case, it receives the same messages as leader for short time interval (usually for the last 10 s). It is excluded from synchronization if it does not receive these messages. This can be caused by network delay, problems with virtual machine node, and the like. Messages can be lost only in the case of leader failure and no participants in replica synchronization. RabbitMQ also provides message replication by queue mirroring but does not provide replica synchronization.

Comparison results in terms of all criteria are summarized in Table 8:

Table 8. Comparison of Apache Kafka and RabbitMQ in terms of reliability

	Weight	**Kafka**	**Rabbit**
At least once	1/10	1	1
At most once	1/10	1	1
Exactly once	1/10	1	1
Message batching	1/10	1	0
Replication	1/10	1	1
Replication synchronization	1/10	1	0
Notification of message reception for producer	1/10	1	1
Transactionality	1/10	1	1
Idempotent messaging	1/10	1	0
Guarantees with regard to message precedence	1/10	1	1
Sum of estimates		**100 %**	**70 %**

Both platforms can implement at-most-once and at-least-one strategies.

Both platforms provide message replication.

Similar factors act on both platforms, thus, it is necessary to search for compromise between throughput capacity and risk of message duplication. Kafka provides idempotent messaging but only for limited traffic.

Both platforms define restrictions for messages in transit, notification of which was not received by sender.

Both platforms provide guarantees concerning order of messaging.

Kafka supports transactions mainly in reading-processing-writing scenario. Herewith, it is necessary to prevent decrease in throughput capacity of the system.

In Kafka, even if a receiver does not process some messages due to hardware failures and incorrect tracing of displacement of last received message, it is still possible to restore displacement of this message (if such case is detected). In RabbitMQ respective messages will be lost.

Kafka can improve its batching efficiency due to packet switching, RabbitMQ does not provide batching due to passive receiving model not preventing receiver conflicts.

In this comparison, Rank 1 is assigned to Kafka 1, Rank 2 – to RabbitMQ. Software ranking in terms of reliability is illustrated in Fig. 2.

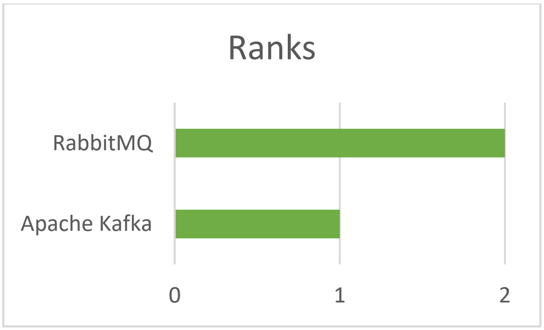

Fig. 2. Ranking of products in terms of reliability.

3.4. Comparison in terms of community support

After detailed analysis of software repositories and number of companies using open source software, the following results were acquired, see Table 9.

Table 9. Comparison of Apache Kafka and RabbitMQ in terms of community support

	Weight	Kafka		Rabbit	
		Index	Estimate	Index	Estimate
GitHub stars	1/7	12,000	1	5,660	0
GitHub forks	1/7	6,500	1	1,600	0
Commits	1/7	6,000	0	18,000	1
Releases	1/7	105	0	244	1
Contributors	1/7	537	1	79	0
Number of using companies (according to stackshare.io)	1/7	623	0	1,154	1
Date of project initiation	0	February, 11	-	February, 07	-
Last stable release	1/7	43 days ago	1	59 days ago	0,5
Sum of estimates		**57%**		**50%**	

Rank 1 is assigned to Apache Kafka, Rank 2 – to RabbitMQ.

Table 10. Comparison of ZeroMQ and gRPC in terms of community support

	Weight	ZeroMQ		gRPC	
		Index	Estimate	Index	Estimate
GitHub stars	1/7	5,160	0	21,000	1
GitHub forks	1/7	1,529	0	4,800	1
Commits	1/7	7,300	0	38,000	1
Releases	1/7	9	0	148	1
Contributors	1/7	376	0,5	456	1
Number of using companies (according	1/7	43	0,5	61	1

to stackshare.io)					
Date of project initiation	0	May, 07	-	March, 15	-
Last stable release	1/7	3 months ago	0	10 days ago	1
Sum of estimates		**14%**		**100%**	

In terms of popularity and community support, Rank 1 is assigned to gRPC, Rank 2 – to ZeroMQ. Software ranking in terms of community support is illustrated in Fig. 3.

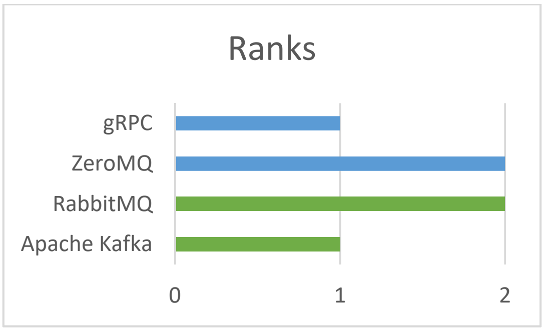

Fig. 3. Ranking of products in terms of community support.

It follows from the presented analysis that Kafka, RabbitMQ, and gRPC are popular and active projects, which is not the case of ZeroMQ. This is confirmed by statistics of Goggle searches for the last year, which serves as a peculiar metrics of tool popularity among customers, this is illustrated in Fig. 4. The data are taken from Google Trends [13].

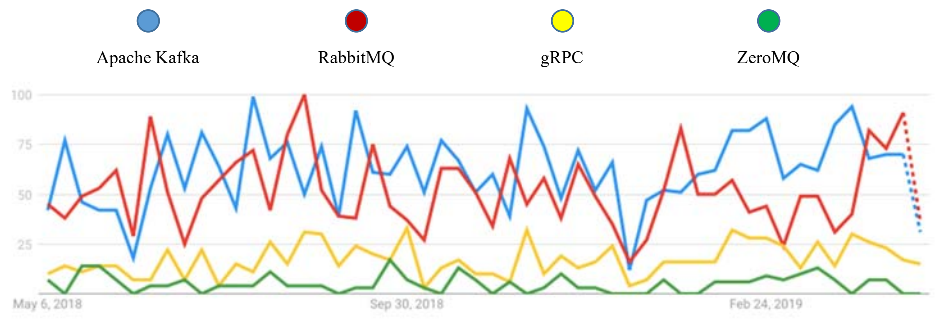

Fig. 4. Popularity of search query regarding the considered products.

3.5. Generalization of comparison results

Generalized comparison results of messaging systems in terms of all coordinates are summarized in Table 11, the final ranking is illustrated in Fig. 5.

Table 11. Generalized comparisons of software products

Comparison coordinate	I		II	
	Apache Kafka	Rabbit MQ	gRPC	ZeroMQ
Productivity	1	2	2	1
Reliability	1	2		
Community support	1	2	1	2
Cumulative rank	3	6	3	3
Final rank	**1**	**2**	**1**	**1**

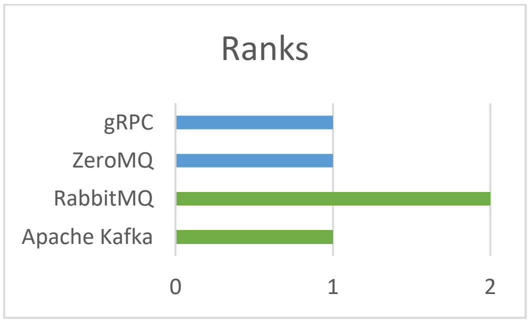

Fig. 5. Final ranking of the considered products.

4. DISCUSSION

This work considered only relatively recent open source messaging systems having interface to communicate with Java.

The main difference from similar studies already existing is that in most of them they compare software solutions only by the criterion of reliability and performance, missing such criteria as the popularity and support of the product by the community, which affects the number of errors and the speed of their correction after detection, size functionality and the use of relevant, fresh and promising technologies and algorithms. In many articles studied, an innovative product from Google - gRPC is almost not found in comparative analyzes with other systems that implement traditional RPC interactions.

There is another even more convenient and time-tested approach with middleware and classical message queues. In such case, middleware for message queuing is used in order to solve problems of synchronous exchange. After recording into such queue, a calling component does not wait for response and can perform other useful operation. Processing component reads the queued messages and generates responses which are received by the calling components in separate stream. Numerous solutions of such type are based, for instance, on ESB developed by IBM [21]. Research is available devoted to analysis and comparison exclusively of middleware; for instance, OpenMQ, ActiveMQ, and MantarayMQ are compared in [22].

Other software solutions are available which can be ranged and compared with the considered products. For instance, Apache ActiveMQ based on JMS [23] is compared with RabbitMQ [14] and OpenMQ [15].

However, there are few studies devoted to comparison of means of remote procedure call: ZeroMQ and gRPC; moreover, since gRPC is quite recent product, it is analyzed and described in significantly less publications than other mentioned products. Hence, this work attempts to consider in general and to compare not only solutions of different architecture but also the recently developed and scantily studied systems.

According to comparison of messaging systems of the second group, Rank 1 was assigned to both products, since ZeroMQ was characterized by significantly higher productivity, and gRPC – by higher community support. However, the following facts should be taken into account: ZeroMQ is a light wrapper of C++ sockets which makes it possible to make point-to-point calls using TCP or IPC; in its turn, gRPC uses modern http/2 protocol [24] with numerous possibilities where main attention is paid to productivity, decrease in latency, use of network and server resources, compression of headers. In addition, ZMQ has not been updated for quite a long time, there are issues of safety regarding encryption, community support decreases, whereas gRPC becomes more and more popular, its releases are published frequently, bugs are corrected quickly, it becomes a standard at RPC market of communications. Contrary to ZeroMQ, where only simpleRPC is available by default, in gRPC communications are expanded to client-side, server-side, and bi-directional streaming [25].

5. CONCLUSION

This work analyzed the most efficient messaging systems. Four software products were considered: Apache Kafka, RabbitMQ, ZeroMQ, and gRPC, which complied with three predefined criteria: support of SSL/TLS encryption, possibility to operate directly with Java platform (provision of Java API), and being developed as open source software.

The comparisons were performed in environment with four coordinates: supported communication types, software productivity, reliability, community support.

Comparison in terms of supported communication types was based on the criteria selected by studying documentation on each software and various publications devoted to nonstandard application of software. The best software product in this comparison was gRPC.

Comparison in terms of productivity was carried out in two groups: Kafka+Rabbit and ZeroMQ+gRPC were compared separately by specialized utilities for load testing. According to these comparisons, the best software in the first group was Apache Kafka, and in the second group – ZeroMQ.

Comparison in terms of reliability in the first group (Kafka and Rabbit) was performed by studying documentation on each software together with highlighting and comparing mechanisms of protection against data loss, data replication, etc. According to these comparisons, the best software was Apache Kafka.

Comparison of the software products in terms of community support was based on project statistics in GitHub repositories, information in public sources concerning use of this or that software by large companies. This comparison was carried out in two groups: Kafka+Rabbit and ZeroMQ+gRPC were compared separately. According to these comparisons, the best software in the first group was Apache Kafka, and in the second group –gRPC.

The hypothesis set at the beginning of the article is partially proved, since the Apache Kafka software product proved to be a leader in the group of systems that implement Publish / Subscribe interactions, while gRPC turned out to be on par with the ZeroMQ product, having an advantage only in certain criteria.

Therefore, as a result of the study, generalized data of a comparative analysis of open source messaging systems were obtained. The best software in the group of systems implementing Publish/Subscribe communication was Apache Kafka, and in the second group implementing point-to-point communication both software products were ranked equally. ZeroMQ was more productive but less promising, functional and supported than gRPC, which lagged behind only in terms of operation rate but supported operation according to modern http/2 protocol, supported additional streaming calls and many other things. In this situation developers should make their choice on the basis of their own preferences upon implementation of their systems of intermodular communication.

ACKNOWLEDGMENTS
This work was supported by the Competitiveness Program of National Research Nuclear University MEPhI (Moscow Engineering Physics Institute), contract with the Ministry of Education and Science of the Russian Federation No. 02.A03.21.0005, 27.08.2013.

REFERENCES:

[1] Gref, H. (2016). 12 technologies will exert drastic influence on economy. https://www.computerworld.ru/news/German-Gref-12-tehnologiy-okazhut-dramaticheskoe-vliyanie-na-ekonomiku (Retrieved: 23.12.2018).
[2] Communication patterns for IoT. Geektimes. (2016). https://geektimes.ru/company/intel/blog/279934/ (Retrieved: 23.12.2018).
[3] Chappell, D. A. (2004). *Enterprise Service Bus*. Sebastopol: O'Reilly Media, Inc.
[4] Decentralized messaging systems. (2014). https://habrahabr.ru/post/240053/ (Retrieved: 23.12.2018).
[5] Microservices were developed before their mainstreaming: Sberbank - Technologies of development. (2016). https://habrahabr.ru/company/jugru/blog/312582/ (Retrieved: 23.12.2018).
[6] Queue server. (2014). https://habrahabr.ru/company/mailru/blog/216363/ (Retrieved: 23.12.2018).
[7] Eugster, P. T. et al. (2003). The many faces of publish/subscribe. *ACM computing surveys (CSUR)*, vol. 35, no. 2, pp. 114-131.
[8] Birrell, A. D. and Nelson, B. J. (1984). Implementing remote procedure calls. *ACM Transactions on Computer Systems (TOCS)*, vol. 2, no. 1, pp. 39-59.
[9] Ahuja, S. P., & Patel, A. (2011). Enterprise service bus: A performance evaluation. Communications and Network, 3(03), 133.
[10] Ahuja, S. P., & Patel, A. (2011). Enterprise service bus: A performance evaluation. Communications and Network, 3(03), 133.
[11] Desmet, S., Volckaert, B., Van Assche, S., Van Der Weken, D., Dhoedt, B., & De Turck, F. (2007). Throughput evaluation of different enterprise service bus approaches. In Proceedings of SERP2007, the 2007 International Conference on Software Engineering Research & Practice (part of the 2007 World Congress in Computer Science, Computer Engineering, and Applied Computing), pp. 378-384.
[12] Sachs, K., Kounev, S., Bacon, J., & Buchmann, A. (2009). Performance evaluation of message-oriented middleware using the SPECjms2007 benchmark. Performance Evaluation, 66(8), pp.410-434.
[13] Le Noac'H, P., Costan, A., & Bougé, L. (2017, December). A performance evaluation of Apache Kafka in support of big data streaming applications. In 2017 IEEE International Conference on Big Data (Big Data). IEEE, pp. 4803-4806.
[14] Rostanski, M., Grochla, K., & Seman, A. (2014, September). Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ. In 2014 federated conference on computer science and information systems . IEEE, pp. 879-884.
[15] Ionescu V. M. (2015) The analysis of the performance of RabbitMQ and ActiveMQ. 14th RoEduNet International Conference-Networking in Education and Research (RoEduNet NER). IEEE, pp. 132-137.
[16] Klein A. F. et al. (2015). An experimental comparison of ActiveMQ and OpenMQ brokers in asynchronous cloud environment. Fifth International Conference on Digital Information Processing and Communications (ICDIPC). IEEE, pp. 24-30.

[17] Introduction Apache Kafka: site. (2018). https://kafka.apache.org/intro (Retrieved: 14.02.2019).
[18] RabbitMQ (2018). RabbitMQ is the most widely deployed open source message broker. Messaging that just works. URL: https://www.rabbitmq.com/ (Retrieved: 14.02.2019).
[19] RabbitMQ (2018) Distributed Messaging – zeromq. ZeroMQ. URL: http://zeromq.org (Retrieved: 14.02.2019).
[20] gRPC (2019). A high performance, open-source universal RPC framework. URL: https://grpc.io (Retrieved: 14.02.2019).
[21] Google Trends (2019) Explore what the world is searching. URL: https://trends.google.ru/trends/ (Retrieved: 20.03.2019).
[22] Chappell D. (2004). Enterprise service bus. O'Reilly Media, Inc.
[23] Ahuja S. P., Mupparaju N. (2014) Performance Evaluation and Comparison of Distributed Messaging Using Message Oriented Middleware. Computer and information science. Vol. 7(4): 9.
[24] Dai J., Zhu X. M. (2010). Design and implementation of an asynchronous message bus based on ActiveMQ. Computer Systems & Applications. Vol. 8: 062.
[25] Corbel R., Stephan E., Omnes N. (2016) 1.1 pipelining vs HTTP2 in-the-clear: Performance comparison. 2016 13th International Conference on New Technologies for Distributed Systems (NOTERE). IEEE, pp. 1-6.
[26] gRPC: (2019). Documentation. gRPC Basics – Java. URL: https://grpc.io/docs/tutorials/basic/java/ (Retrieved: 20.03.2019).