Petri Nets for Modelling of Message Passing Middleware in Cloud Computing Environments

Oleg Iakushkin[1(✉)], Yulia Shichkina[2], and Olga Sedova[1]

[1] Saint-Petersburg University, 7/9 Universitetskaya nab., St. Petersburg 199034, Russia
o.yakushkin@spbu.ru

[2] Department of Computer Science and Engineering, Saint Petersburg Electrotechnical University "LETI", St. Petersburg, Russia

Abstract. Cloud systems allow to run parallel applications using solutions with distributed heterogeneous architecture. Software development for heterogeneous distributed environment requires a module-based design. The components in such module system are connected by means of telecommunications network enabling message passing. This article describes an interaction model for components in distributed applications. The model was designed based on the paradigm of Variable Speed Hybrid Petri Nets and allows to analyse system performance at various tiers: selection of the optimum approach to load balancing between components; making scaling decisions to enhance performance of certain modules; fine-tuning the interaction between system components. The model is not contingent on particular tools a user might employ to implement a solution; it also provides a monitoring data integration functionality.

The model contains descriptions of standard messaging patterns linking components of distributed applications. These patterns include request-reply and publish-subscribe. Load balancing algorithms for various schemes of these patterns usage have been developed for a cloud environment.

Keywords: Cloud services · Messaging patterns · Systems architecture · Petri nets · Messaging middleware · Distributed applications

1 Introduction

Cloud platforms have become a flexible computing environment that can run a wide range of applications. They allow dynamic allocation and control of heterogeneous computing resources, which means computing can be provided as a service. This enables scaling of active applications within a common network of data centres located all around the world.

O. Iakushkin—This research was partially supported by SPbU (Saint Petersburg State University) grants 9.37.157.2014, 0.37.155.2014 and Russian Foundation for Basic Research grant (project no. 16-07-01113 and no. 16-07-01111).

There are tasks that can be solved on an individual cloud platform. However, creation of a hybrid cloud computing environment is also relevant [4,5,9,10]. Such environment pools together the resources of private data processing centres and commercial suppliers of cloud services.

Cloud platforms can solve a host of tasks which are fundamentally different. At first approximation, we can single out the following three types: 1. handling many small tasks (e.g., mass calculation); 2. processing large data arrays; 3. calculation of a single big task, represented be a coherent system of control commands that changes its state after each new command is introduced. These types of tasks set out different requirements for load balancing and scaling. Remarkably, the first type is the only one that allows easy scaling even within a grid. On the whole, however, cloud environment is impossible to master without tools enabling interaction between components [12,15–19].

This paper is focused on components of distributed applications that run on different computing resources united in a single computing network. These components must function in a continuous and uninterrupted way; moreover, they require constant interaction and information exchange. As a rule, modular solutions allow usage of computing resources with adjustable capacity which can be altered while the task is being solved. Coordination and load balancing are a major challenge for communication systems that connect modules with each other. This paper offers a mathematical model allowing formal description of such systems. The validity of the model is not contingent on particular technological solutions.

2 Related Work

Petri nets are employed to describe and analyze the features of distributed communication systems.

The use of Generalised Stochastic Petri Nets (GSPN) for messaging middleware in broker-based architecture was discussed by Fernandes et al. based on the case of IBM Web Server solution [8]; the major attention was given to publish-subscribe and message queuing patterns. The Coloured Petri Net (CPN) was later used by Fahland and Gierds [7] to develop models allowing analysis of Enterprise Integration Patterns (EIP) described in [14]. Some of these models can classify as middleware for message passing: Pipes and Filter, Message Router, Message Translator, Message Endpoint, Recipient List, Aggregator, Request-Reply, Channel Adapter. However, load balancing and coordination of multiple sending/receiving nodes were not addressed; neither were the temporal aspects of message passing. We should note, that most of modern message-passing systems by default employ Round-Robin to perform load balancing. The technique utilizes the algorithm that partly correlates with the one presented in [21], where it is used to describe Ethernet packet switching. In this paper, we took into account the load balancing description model contained in [21]. The approach to formalization of middleware systems description was mentioned in [22]; the work offered a transition from the concept 'API plus informal prose' to the concept

'API plus formal description'. In [22], there was also provided an example of formal description for Common Object Request Broker Architecture (CORBA). However, this method described only the behaviour of the system, without regard to its mathematical dimension. Therefore, it was not supported by the expert community.

Wester-Ebbinghaus [2] used a web-service to give an example of a full-fledged programme whose code is based on a Petri nets model. The stability problem in two particular classes of queuing systems was analysed by Konigsberg [20]; the analysis involved timed Petri nets, Lyapunov methods, and max-plus algebra.

So, we demonstrated there is a wide range of research works using Petri nets to design a formalized approach to the architecture of message-passing software. The task is handled in terms of both the internal structure and an outside observer which analyzes the system's components. However, most of the models of message passing and load balancing limit themselves to one of the following options: they either examine broker-based architecture (i.e., the tasks are solved externally), or analyze separate client/server pairs. As a result, the failure of interacting components remains largely neglected. Moreover, there is one more issue that is overlooked: the functioning of systems which include many modules communicating with each other and concurrently running various message-passing patterns, scaling methods, and balancing strategies.

3 The Interaction Model for Components of Distributed Application Architecture in a Cloud Environment

This paper describes a model specifying components interaction in a distributed application architecture in a cloud environment. The model was built using message-passing patterns allowing to directly link computing nodes. The patterns used in the model are available in ZeroMQ[1] [13,23] and NamoMsg[2] systems: request-response, publish-subscribe and pipeline. These solutions are widely used by large companies to solve a broad variety of calculation tasks: from high-energy physics (e.g., by European Organization for Nuclear Research, CERN) to independent business applications [1,6]. Our model is mainly distinguished by the opportunity to involve an outside controller. The controller can allocate additional computing load, invoke or stop service system modules, and monitor interaction in the service network. Introduction of the controller allowed a major optimization of the model's logic.

The model accommodates patterns enabling forwarding of large messages split into sets of fragments. In essence, we are speaking of data streaming supported by load balancing functionality. We also examine load balancing for the cases requiring inversion of control (i.e., when a client needs to maintain connection to a particular server for an unspecified number of request/reply cycles).

The formal description of the systems in question is generally performed using Petri nets, whose apparatus provides a rich arsenal of means. To select the

[1] http://zeromq.org/.
[2] http://nanomsg.org/.

ones suited for our task, we analyzed typical features of distributed systems: the same channels are able to pass various types of messages; the time needed for message passing correlates with the capacity of communication channels between the nodes; possible data loss while delivering messages; temporal unavailability of certain components; connection loss between modules according to Brewer's conjecture [11]. Petri nets allow to produce a valid formal model reflecting this particularities. A detailed mechanism of such description is available in Colored Extended Variable Speed Hybrid Petri Nets (VHPN).

We designed a model based on the principles of formalism and graphical representation outlined in [3].

Noteworthily, there is a crucial issue that should be resolved prior to the development of the model: the mark symbol for VSPN must be bound to any message forwarded within the system, unless explicitly stated otherwise. This binding is of paramount importance for the case under review.

3.1 The Model of the Architecture Component 'The Choice of a Message Receiver'

Let us we consider message sending from a client to one of the equivalent servers. The examined libraries employ the Round-Robin algorithm to balance the load between servers at the side of the client. The algorithm for automatic balancing using Petri nets is given in Fig. 1.

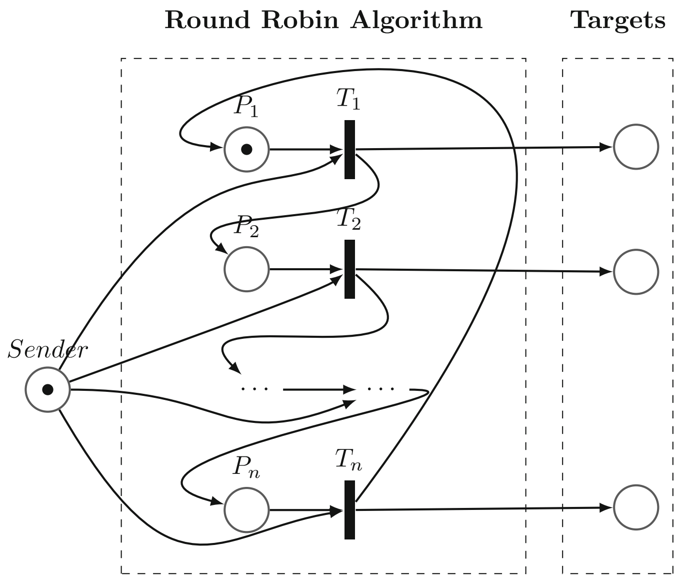

Fig. 1. The algorithm for automatic selection of a receiving node from an array of servers.

3.2 Models of the Architecture Component 'Controlling Message Passing When Nodes are Tuned Off or Overloaded'

Clouds offering IaaS allow to scale computing resources on running systems. However, the budgets of end users are not unlimited. Therefore, it is sound to make an estimate of a distributed system's capabilities that is, to calculate the maximum number of server computing nodes available to the end users; and to analyze the system operation in case of potential shutdown of nodes.

The design of a module incorporated into the client's load balancing system for message queuing is shown in Fig. 2. The design provides for a possible shutdown or failure of a remote server. Here the mark is responsible for the state of the remote node. If the mark is 'turned off', the Round-Robin algorithm will skip this node when sending a message.

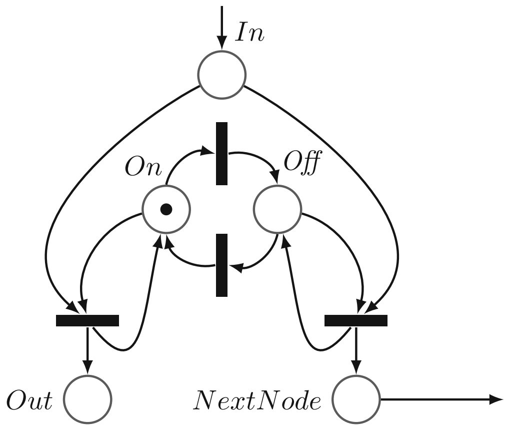

Fig. 2. The model for automatic skipping of dysfunctional nodes while sending messages.

Confirmation of message receipt and processing is an important mechanism which can be easily modeled using Petri nets (Fig. 3). This can be implemented by means of inhibitor arcs and a function of time that describes transitions between sending and receiving nodes. However, such mechanism is not easy to put into practice.

Here it is the server that must response to message loss or processing failure. This is the major difference from the model where a server node is excluded from the sending list by the client itself. As a result, a server's failure or slow response due to overload can be communicated to the client by means of only two strategies of inhibitor arcs behaviour:

- Temporal waiting for confirmation of message receipt by a server node;
- Outside control mechanism that monitors the client's and the server's nodes; it should be capable of initiating the message resend.

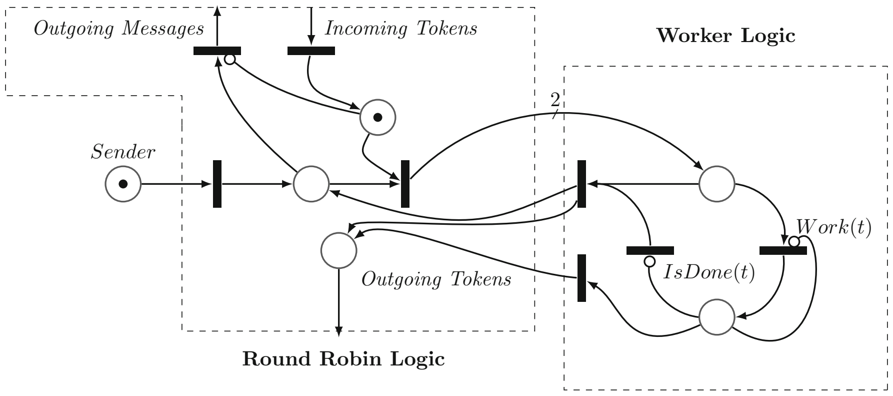

Fig. 3. The model for confirmation of message receipt and successful processing.

The former strategy can result in multiple resending of one and the same message to all server nodes available. This might be the case, if the server node or the network are overloaded and cannot promptly notify the client about receipt and processing. This situation can be critical for the whole system.

The strategy described is implemented in broker-based messaging systems. Let us examine how it works in RabbitMQ[3] broker-based solution [24] which uses the wide-spread Advanced Message Queuing Protocol, or AMPQ[4] [25]. The client and the server are connected through a broker which forwards the client's messages to the server. If we use the receipt confirmation strategy, the broker subsequently resends the client's query to each of subscriber servers. The broker utilizes the Round-Robin balancing algorithm to go over the servers until the confirmation is received within the timespan set by the client. In other words, if no confirmation is received in a due time, the message is sent to be processed by another potential node. The system developer can set the waiting time before the messages are sent, but this approach reduces the system's throughput by more than two orders of magnitude[5].

The latter strategy requires an external controller and cannot be used as an out-of-the-box solution within the message-passing systems under review. The core features of such controller are described below as a general model, flexible enough to suit a variety of concrete solutions.

3.3 The Model of External Controller in Message Passing

The peer-to-peer networks provide little practical opportunities for using an outside controller. The reasons include their unlimited size and unlimited remoteness of computing nodes. Cloud platforms are more well-suited for the purpose: they

[3] http://www.rabbitmq.com/.
[4] http://www.amqp.org/.
[5] http://www.rabbitmq.com/confirms.html.

are hosted by particular data processing centres, which allows monitoring of the nodes' operation.

Hybrid cloud systems use geographically distributed computing resources and a network of data processing centres. In such systems, controlling components can be separated, their parts being bound to certain locations. This branch of Computer Science requires further study, which should involve laborious efforts and a large number of versatile resources. For this reason, we will restrict our analysis to the use of our model for mass message passing. We will not comment upon such issues as scaling and bidirectional communications analysis.

The choice of a receiving server is best performed by an outside controller deployed in the cloud. The model thereof is illustrated in Fig. 4. It enables the choice of an appropriate receiver for each message sent. The controller uses the Round-Robin algorithm to evaluate load balancing in the whole system. In the model of the architecture component 'choice of a message receiver' (see Picture 1), each client independently performs the choice of the receiving server. If the number of servers is limited, such solution may result in an uneven load and a system failure. The problem is solved, if we use a controller that monitors message passing from clients to server nodes.

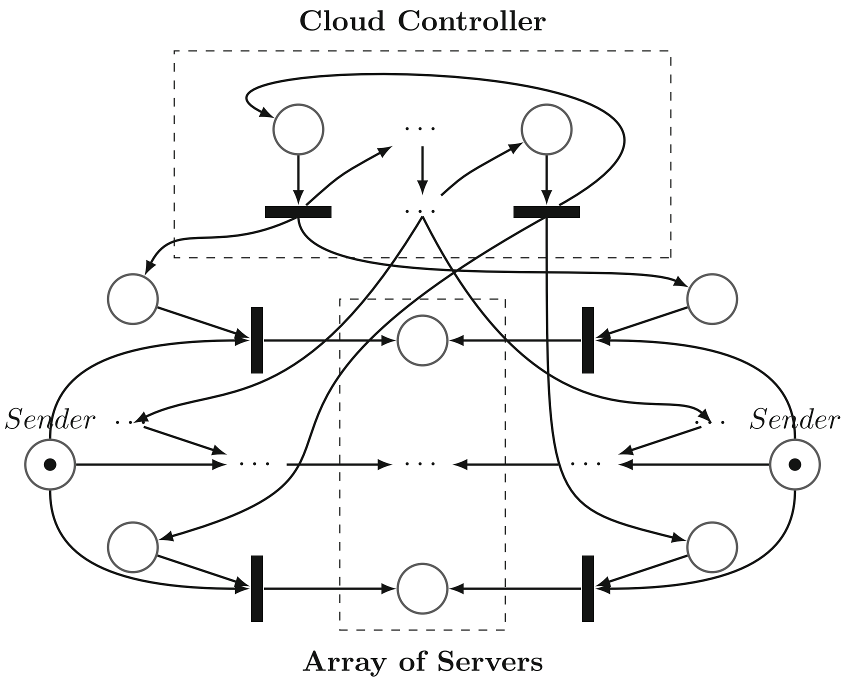

Fig. 4. The controller-based model of message passing from clients to a bunch of server nodes.

This solution has a drawback: the system relying on an outside controller and the Round-Robin load balancing algorithm will send each message at least three times as slow as compared to client-side balancing. The external controller node will need to collect data from each sending node in every case of message passing.

However, the outside controller opens up new load balancing opportunities. There are strategies allowing to optimize the number of messages required by the controller for proper operation, and its impact on the system in general:

– heartbeat tool used to communicate the statistics of the nodes' functioning;
– external (system-based) and internal utility programmes monitoring the nodes' work;
– loading one node to a full capacity before the load is distributed to the next node.

These external tools can be introduced to the load balancing system as an extension to the model in Fig. 4.

3.4 The Load Balancing Model for Data Streaming

The model in Fig. 5 contains a mark that is forwarded from the client to the server. The mark is a positive number, not necessarily an integer, that reflects the state of the data stream. The load balancing in this model relies on a counter of the streams transmitted to each server. The data are streamed to the least loaded server.

The streams balancing system can be also included into the cloud controller model displayed in Fig. 4. Moreover, the final model can accommodate all the extensions mentioned above.

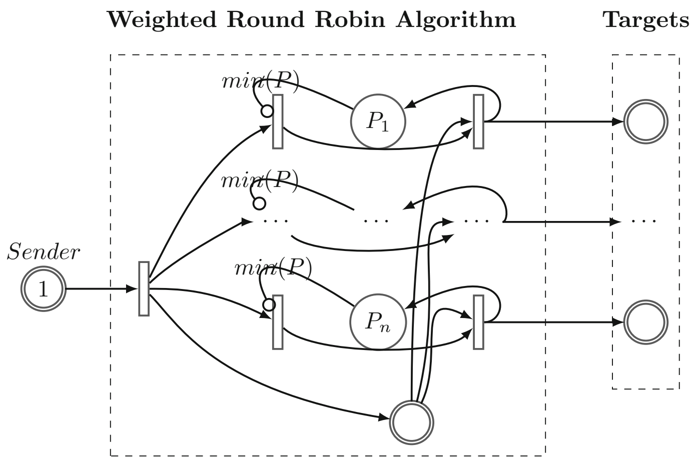

Fig. 5. The load balancing model for data streaming.

3.5 Bidirectional Communication Model

A message exchange dialogue is an important tool of nodes interaction. For example, the control over a client's movements is given to a navigator that is, a particular instance of an external server. Until the client arrives to the specified point, it notifies the server about every command executed. Such long-term bilateral communication does not comply with the Round-Robin algorithm going from one server to another. In this case, the initial task is known to the first server only. Therefore, it is this server that should track the changes in the object it interacts with.

The dialogue model between the client and the server is provided in Fig. 6. Two colors are used:

- *EndT* the color indicating a message ending the dialogue;
- *T* the color indicating any other message.

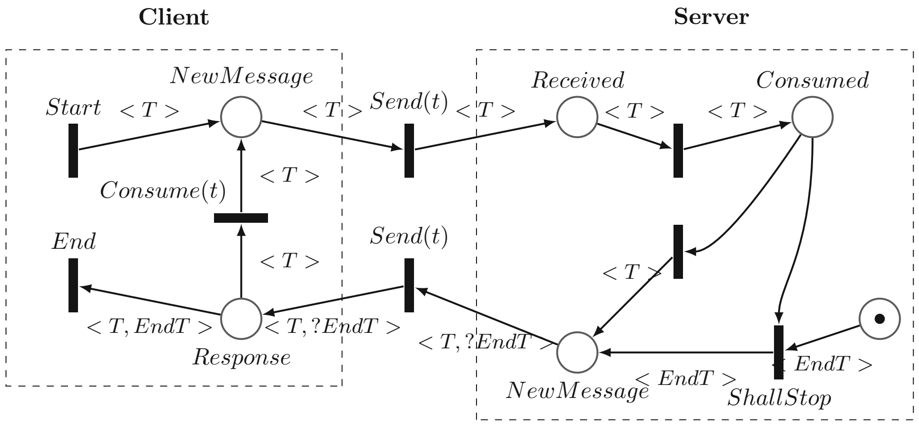

Fig. 6. The model of bidirectional communication.

The interaction continues until the server signals the client to exit. Importantly, the system can include a server activity check. In this case, the interaction will end, if the server shuts down. However, a correct exit from the dialogue is only possible at the level of a full-scale system, based on a particular business logic. For this reason, it cannot be fully presented here.

The model shown in Fig. 7 can be applied to load balancing in long-term bidirectional communications. It can be deployed at the level of the outside controller in order to monitor multiple clients

3.6 Publish-Subscribe

The model of message passing to many clients is shown in Fig. 8. Every node in the system has two options: it is either a subscriber, or does not receive messages.

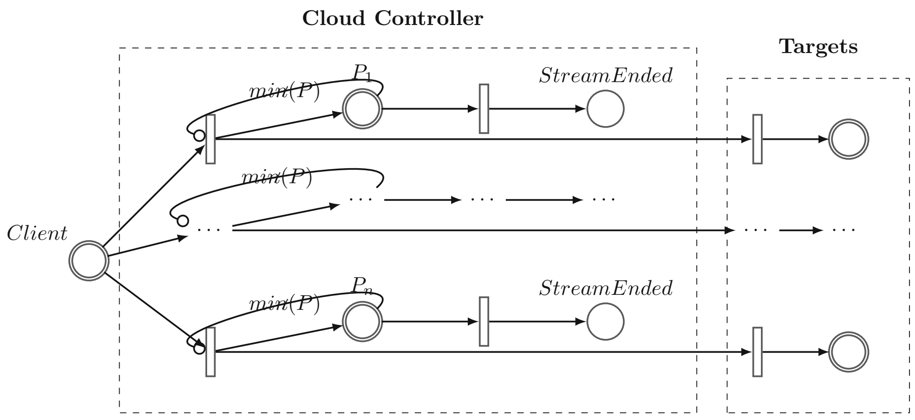

Fig. 7. The load balancing model for long-term dialogues.

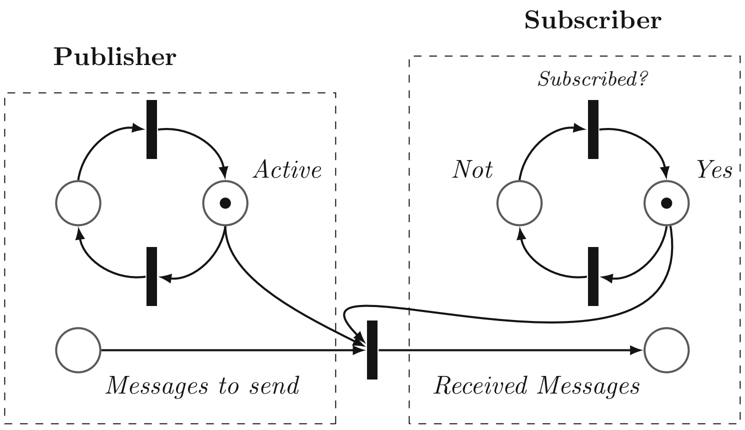

Fig. 8. The model of Publish-Subscribe pattern.

For hybrid cloud systems, it is important to allow message passing to network segments which are very remote from the publisher or require pre-processing of messages.

The model including a broker which forwards messages beyond the cloud system can be easily presented using a publisher with a subscriber which is retransmitting received messages to peers local to its cloud.

3.7 Remarks

The system's work can be modeled using the function of time and the function of message length, provided we know data transmission speed. The system can be described using several layers corresponding to data transmission channels distinguished by name or by message type.

4 Conclusions

We offered a model which allows to describe temporal and quantitative qualities of the systems examined. These qualities are described in conjunction with logical ties between both system components and their particular implementations. We also reviewed the major interaction patterns of modular distributed programmes executed in a cloud computing environment. The paper demonstrates the possibilities of combined and separate use of the schemes analysed. Their relevance and extension prospects have been explained and justified.

The study is based on practical work of service systems development using communication libraries RabbitMQ and ZeroMQ. The development projects were carried out at St. Petersburg State University. The work presented here is a stage in a larger study. Our long-term goal is to design a platform enabling development of complex dynamic systems deployed in a variety of clouds. The platform will employ both standard and user- defined messaging and interaction patterns. Petri nets were used to create a component model of distributed cloud service systems.

Acknowledgments. This research was partially supported by SPbU (Saint Petersburg State University) grants 9.37.157.2014, 0.37.155.2014 and Russian Foundation for Basic Research grants (project no. 16-07-01113 and no. 16-07-01111). Microsoft Azure for Research Award (http://research.microsoft.com/en-us/projects/azure/) as well as the resource center "Computer Center of SPbU"(http://cc.spbu.ru/en) provided computing resources. The authors would like to acknowledge the Reviewers for the valuable recommendations that helped in the improvement of this paper.

References

1. Arce, P., Maureira, C., Bonvallet, R., Fernandez, C.: Forecasting high frequency financial time series using parallel ffn with cuda and zeromq. In: 2012 9th Asia-Pacific Symposium on Information and Telecommunication Technologies (APSITT), pp. 1–5, November 2012
2. Betz, T., Cabac, L., Duvigneau, M., Wagner, T., Wester-Ebbinghaus, M.: Software engineering with petri nets: a web service and agent perspective. In: Haddad, S., Yakovlev, A. (eds.) ToPNoC IX. LNCS, vol. 8910, pp. 41–61. Springer, Heidelberg (2014)
3. David, R., Alla, H.: Discrete, Continuous, and Hybrid Petri Nets. Springer, Heidelberg (2010)
4. Degtyarev, A., Gankevich, I.: Balancing load on a multiprocessor system with event-driven approach. In: Gavrilova, M.L., Tan, C.J.K. (eds.) Trans. on Comput. Sci. XXVII. LNCS, vol. 9570, pp. 35–52. springer, Heidelberg (2016). doi:10.1007/978-3-662-50412-3_3
5. Degtyarev, A.B., Logvinenko, Y.V.: Agent system service for supporting river boats navigation. Procedia Comput. Sci. **1**(1), 2717–2722 (2010). ICCS 2010
6. Dworak, A., Charrue, P., Ehm, F., Sliwinski, W., Sobczak, M.: Middleware trends and market leaders 2011. In: Conference Proceedings C111010 (CERN-ATS-2011-196), FRBHMULT05, 4 p., October 2011

7. Fahland, D., Gierds, C.: Analyzing and completing middleware designs for enterprise integration using coloured petri nets. In: Salinesi, C., Norrie, M.C., Pastor, Ó. (eds.) CAiSE 2013. LNCS, vol. 7908, pp. 400–416. Springer, Heidelberg (2013)
8. Fernandes, S., Silva, W., Silva, M., Rosa, N., Maciel, P., Sadok, D.: On the generalised stochastic petri net modeling of message-oriented middleware systems. In: 2004 IEEE International Conference on Performance, Computing, and Communications, pp. 783–788 (2004)
9. Gankevich, I., Gaiduchok, V., Gushchanskiy, D., Tipikin, Y., Korkhov, V., Degtyarev, A., Bogdanov, A., Zolotarev, V.: Virtual private supercomputer: design and evaluation. In: Computer Science and Information Technologies (CSIT), 2013, pp. 1–6 (2013)
10. Gankevich, I., Korkhov, V., Balyan, S., Gaiduchok, V., Gushchanskiy, D., Tipikin, Y., Degtyarev, A., Bogdanov, A.: Constructing virtual private supercomputer using virtualization and cloud technologies. In: Murgante, B., et al. (eds.) ICCSA 2014, Part VI. LNCS, vol. 8584, pp. 341–354. Springer, Heidelberg (2014)
11. Gilbert, S., Lynch, N.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News **33**(2), 51–59 (2002)
12. Grishkin, V., Iakushkin, O.: Middleware transport architecture monitoring: topology service. In: 2014 20th International Workshop on Beam Dynamics and Optimization (BDO), pp. 1–2, June 2014
13. Hintjens, P.: ZeroMQ: Messaging for Many Applications. O'Reilly, Sebastopol (2013)
14. Hohpe, G., Woolf, B.: Enterprise integration patterns. In: 9th Conference on Pattern Language of Programs, pp. 1–9 (2002)
15. Iakushkin, O.: Intellectual scaling in a distributed cloud application architecture: a message classification algorithm, pp. 634–637 (2015) (cited By 0)
16. Iakushkin, O.: Cloud middleware combining the functionalities of message passing and scaling control, vol. 108 (2016) (cited By 0)
17. Iakushkin, O., Grishkin, V.: Messaging middleware for cloud applications: Extending brokerless approach. In: 2014 2nd International Conference on Emission Electronics (ICEE), pp. 1–4, June 2014
18. Iakushkin, O., Grishkin, V.: Unification of control in P2P communication middleware: towards complex messaging patterns. In: Simos, T.E., Tsitouras, C. (eds.) Proceedings of the International Conference of Numerical Analysis and Applied Mathematics 2014 (ICNAAM-2014), AIP Conference Proceedings, vol. 1648. Amer Inst Physics, Melville (2015)
19. Iakushkin, O., Sedova, O., Valery, G.: Application control and horizontal scaling in modern cloud middleware. In: Gavrilova, M.L., Tan, C.J.K. (eds.) Transactions on Computational Science XXVII. LNCS, vol. 9570, pp. 81–96. Springer, Heidelberg (2016). doi:10.1007/978-3-662-50412-3_6
20. Konigsberg, Z.: Timed petri nets modeling and lyapunov/max-plus algebra stability analysis for a type of queuing systems. Int. J. Pure Appl. Math. **86**(2), 301–323 (2013)
21. Pedroso, C.M., Fonseca, K.: Modeling weight round robin packet scheduler with petri nets. In: International Conference on Communication Systems, vol. 1, pp. 342–345. IEEE (2002)
22. Rosa, N.S., Cunha, P.R.F.: Behavioural specification of middleware systems. J. Braz. Comput. Soc. **12**, 63–74 (2006)

23. Sstrik, M.: The Architecture of Open Source Applications, vol. 2. CreativeCommons, Mountain View (2012)
24. Videla, A., Williams, J.J.: RabbitMQ in Action. Manning, Shelter Island (2012)
25. Vinoski, S.: Advanced message queuing protocol. IEEE Internet Comput. **10**(6), 87–89 (2006)