Petri Net-Based Model of Peer-to-Peer Dataset Replication in Big Data

Ilija Hristoski[1], Tome Dimovski[2]

[1]Faculty of Economics – Prilep, St Kliment Ohridski University – Bitola, Prilepski Braniteli St 133, Prilep, N. Macedonia
[2]Faculty of Information and Communication Technologies, St Kliment Ohridski University – Bitola, Partizanska St, Bitola, N. Macedonia

[1]ilija.hristoski@uklo.edu.mk, [2]tome.dimovski@uklo.edu.mk

Abstract—**In the contemporary business landscape, the Big Data paradigm helps companies harness their abundant data in their ever-lasting pursuit for new opportunities. Keeping Big Data infrastructure highly performant, scalable, dependable, and available is the key challenge such companies face with. All of these aspects are usually addressed by deploying fault-tolerant systems. Dataset replication is just one of the crucial operations that has to be carried out regularly to improve system resilience. The paper aims at proposing performance models of the execution of reading and writing operations found in one of the main architectures involved in dataset replication: the Peer-to-Peer architecture, based on the utilization of the class of Generalized Stochastic Petri Nets. Such models can be utilized for simulation purposes to obtain various performance metrics vis-à-vis different working scenarios including different input parameters.**

Keywords – Big Data infrastructure, P2P dataset replication, read/write operations, performance modeling, Generalized Stochastic Petri Nets

I. INTRODUCTION

The rising popularity of large-scale applications, such as social networking, the Internet of Things, and scientific research, has resulted in a speedy production of massive amounts of data belonging to various categories. Turning ‘Big Data’ into highly valuable insights, actions and outcomes is a key premise to better decision making, more effective customer engagement, sharper competitive edge, hyper-efficient operations, as well as compelling product and service development. Because such data is diverse and distributed on a wide scale, managing it efficiently represents a considerable problem. Attaining adequate data availability raises serious challenges to companies that have adopted the Big Data paradigm. Nonetheless, providing high-performance levels, i.e. timely responses to read/write queries against ever-increasing data volumes, became of utmost importance and priority in the Big Data world. In this context, the process of storing the same data in multiple locations, known as ‘data replication’, allows for increased data availability and accessibility, reduced data access costs, and improved fault tolerance. It also improves Big Data systems’ resilience and reliability. All of these have become, *de facto*, a must-have for successful digital transformation in a global economy.

Realizing the importance of the concept of dataset replication in today’s business environment, in this paper, we develop and present performance models of the execution of the READ and the WRITE requests in Peer-to-Peer (P2P) dataset replicated architecture with two nodes/peers, based on the class of Generalized Stochastic Petri Nets (GSPNs).

The paper is structured as follows. Section II briefly overviews the recent research made on this topic. Section III elaborates the notion of dataset replication. The class of GSPNs is explained in Section IV. Section V introduces GSPN sub-models depicting the processing of both the READ and the WRITE requests in a P2P dataset replicated architecture with two nodes/peers. The related discussion is subject to Section VI. The last section concludes.

II. Related Research

Since their emergence in the late ’90s of the XX century, various classes of stochastic Petri nets have been regularly utilized for performance analysis of replicated systems. Lately, those classes of PNs are being used for modeling and evaluation of Big Data systems’ infrastructural and architectural components. What follows is a brief overview of the related research.

A Stochastic Petri Net model of a replicated file system in a distributed environment, where replicated files reside on different hosts and a voting algorithm is used to maintain consistency, as described in [1].

Reference [2] focuses on the implementation and evaluation of Petri Net-based formal models of execution of various operations performed on data within the Big Data paradigm, including replication, through the newly introduced concept of Active Data programming model, which allows code execution at each stage of the data life cycle.

Some of the recent studies that have been carried out have been focused on HDFS, the main component of Hadoop, which provides a Big Data storage service, where data is reliably kept in a distributed fashion on different servers. In one such study, the development of a mathematical model, aimed at representing the storage service activities of HDFS and formulating its dependability attributes, has been proposed, based on the utilization of Stochastic Petri Nets. Such a model accurately quantifies two important dependability metrics – reliability and availability of HDFS [3].

Reference [4] discusses the adoption of Petri Nets (PNs) in creating a visual model of the MapReduce framework to analyze its reachability property, by presenting a real big data analysis system to demonstrate the feasibility of the proposed PN model. The model is then used to describe the internal procedure of the MapReduce framework in detail, to list common errors, and to propose an error prevention mechanism.

Recognizing the fact that data replication is not only a costly process but also a wastage of energy resources, Rizwan Ali et al. have applied the class of Colored Petri Nets (CPNs) for both modeling and analysis, to address resource utilization issues of CAROM, a hybrid file system [5].

III. Peer-to-Peer Dataset Replication

A plethora of novel storage strategies and technologies have been invented to achieve efficient, cost-effective, and highly scalable storage solutions, due to the necessity to store large Big Data datasets, often in multiple copies. The term ‘Big Data storage’ refers to the infrastructure that is designed specifically to store, manage and retrieve massive amounts of data in such a way that data can easily be accessed, used, and processed by Big Data applications and services. It is a compute-and-storage architecture that can be used to collect and manage huge-scale datasets and perform real-time data analytics. To fulfill its purpose, Big Data storage primarily supports storage and input/output (read/write) operations on storage with a very large number of data files and objects.

One of the underlying mechanisms behind Big Data storage technology is dataset replication. Reference [6] points out that “replication stores multiple copies of a dataset, known as replicas, on multiple nodes.” It provides high scalability and availability because “the same data is replicated on various nodes.” The achieved data redundancy ensures that data cannot be lost when a single node fails, i.e. replication ensures fault tolerance of the Big Data infrastructure and therefore it significantly improves system resilience and reliability.

In this paper, we put the focus on Peer-to-Peer (P2P) dataset replication, an alternative method to Master/Slave dataset replication. With P2P replication, all nodes operate at the same level. Each node, known as a peer, is equally capable of handling both READ and WRITE requests. However, each write is copied to all peers simultaneously, whilst data is read from just one peer, no matter which one, as portrayed in Fig. 1 [6].

In Fig. 1, the load balancer is distributing users’ WRITE requests evenly between the two nodes/peers. WRITE requests refer to the insertion, update, and delete operations on the Big Data dataset, held within identical replicas across all nodes.

There are two common approaches in deploying P2P replication. According to the first one, which does not include a load balancer, each single WRITE request is being simultaneously sent to and carried out on all nodes. According to the second approach, which includes a load balancer, the WRITE request is first being processed solely on one of the nodes (e.g. Node #1), and then, after the replication occurs among the nodes, the dataset residing on Node #2 gets updated, and Replica B becomes identical to Replica A.

In both approaches, users' READ requests are being also distributed evenly across peers through the process of load balancing. In this case, the read operation is being carried out using any of the nodes, under the assumption that each of them maintains updated and identical datasets.

It should be notified that P2P replication is susceptible to write inconsistencies, which can occur as a result of the late concurrent propagation of updates of the same data across numerous peers. This issue, however, can be addressed by implementing either a pessimistic or an optimistic concurrency strategy [6].

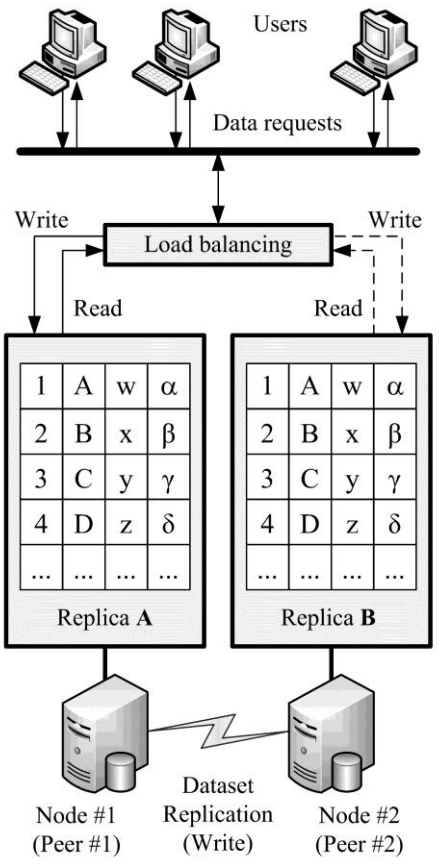

Figure 1. Schematic representation of a P2P dataset replication using two peers and load balancing.

IV. GENERALIZED STOCHASTIC PETRI NETS

The class of Generalized Stochastic Petri Nets (GSPNs), which belongs to the family of Markovian stochastic Petri Nets, has been widely utilized for performance and reliability modeling and evaluation of complex distributed systems (manufacturing systems, multiprocessor systems, control systems, communication protocols, organizational activities, hardware and software systems, etc.), showing non-deterministic behavior. GSPNs are a graphical tool intended for formal description of discrete-event dynamic systems that exhibit characteristics of concurrency, synchronization, parallelism, mutual exclusion, blocking, and conflict, as well as a mathematical tool, aimed at carrying out advanced formal analysis [7][8].

GSPNs represent bipartite graphs, consisting of two classes of nodes (places and transitions), as well as directed arcs connecting particular nodes. They utilize two types of transitions: (a) exponentially distributed timed transitions, drawn as thick empty bars, used for modeling of random delays associated with the execution of activities, and (b) immediate transitions, drawn as thin black-colored bars, used for representing logical actions that do not consume time. The latter ones allow for the modeling of branching probabilities that are independent of timing specifications. In general, transitions represent events. An event occurs when the corresponding transition fires. Besides transitions, a GSPN model also includes places, drawn as non-colored circles, which represent conditions. Tokens, drawn as small black-colored circles, are always put into places; they circulate throughout the GSPN model by moving from a place to a place when transitions fire, thus denoting conditions holding at any given time. Directed arcs, which are drawn from places to transitions, and from transitions to places, are used to indicate which combination of conditions must hold for an event to occur (i.e. a transition to fire), and which combination of conditions holds after the event occurs. Each arc also has a multiplicity/weight value that indicates the number of tokens required from a source place, or the number of tokens provided to a target place. A transition fires only if it is enabled, i.e. if all of its input places contain at least one token. Depending on the multiplicity of input and output arcs attached to a transition, firing it removes at least one token from each input place, and puts at least one token into each output place it is connected to. This is equivalent to an event that is enabled by a combination of conditions. GSPN models also support the usage of inhibitor arcs, which are always drawn from places to transitions, and end with a small circle, meaning that the transition is disabled when its input place is marked with at least one token. Any distribution of tokens over the places in a GSPN model represents a specific configuration, known as a marking. The markings in a GSPN model in which only exponential transitions are enabled are known as tangible markings. All other markings are vanishing markings, which specify logical changes in the modeled system.

The performance evaluation of the analyzed system using GSPNs includes the following four stages: (1) Modeling the observed system using a GSPN; (2) Generating the reachability graph/tree through a marking process; (3) Computing the steady-state probability distribution by solving the resulting reachability graph as a Markov chain; (4) Obtaining the required performance measures from the steady-state probabilities.

V. GSPN-Based Models of Read/Write Operations in P2P Dataset Replication

Having minded the famous quote attributed to the prominent British statistician George E. P. Box, stating that “all models are wrong but some models are useful.” we present two GSPN-based models in the subsequent sections. Both models refer to the P2P dataset replication architecture with two nodes/peers. The first one portrays the execution of a single WRITE request, whilst the second one models the execution of a single READ request. In both models, a ‘Round Robin’ load-balancing scheme of the incoming requests is being implemented.

A. *Processing of WRITE requests: the model*

Fig. 2 shows the GSPN model of the execution of a single WRITE request. The firing of the exponential transition *T_write_request*, which occurs with a rate of $\lambda_{\text{WRITE\_REQUESTS}}$, sends the WRITE request to the load balancer (a token in the place *P_write_request*). After Node #1 (Peer #1) is being selected to process the request (a token in the place *P_n1_selected*), the processing of the WRITE request can start only if Node #1 is idle (a token in the place *P_n1_idle*). The time needed to process the WRITE request is exponentially distributed and its mean equals $1/\mu_{\text{WRITE}}$. After processing the WRITE request by Node #1 (a token in the place *P_n1_write_end*), a token is being put into the place *P_n2_start*, to process the same WRITE request on the peering dataset. At the same time, the firing of the immediate transition *T_replicate_to_n2* also puts a token into the place *P_n1_to_n2*, which indicates that the processing that is going to happen on Node #2 is a result of a replication process, not a result of the initial processing of a WRITE request coming directly from the load balancer. The replication, i.e. the processing of the WRITE request coming from Node #1 on Node #2 occurs only if Node #2 is idle (a token in the place *P_n2_idle*). The dataset residing on Node #2 is being updated by the firing of the exponential transition *T_n2_write*, which fires with a rate of μ_{WRITE}. This puts a token in the place *P_n2_write_end*, which is an input place for two immediate transitions. The first one is transition *T_do_not_replicate_to_n1*, which becomes enabled as soon as there is a token in the place *P_n1_to_n2*. The firing of this transition means that the dataset residing on Node #1 has been replicated to Node #2 (a token in the place *P_n1_replicated*). The second immediate transition, to which the place *P_n2_write_end* represents an input place, is *T_replicate_to_n1*, which becomes enabled if there is no token in the place *P_n1_to_n2*, due to the inhibitor arc originating from this place. The firing of the enabled immediate transition *T_replicate_to_n1* means that the WRITE request has been previously processed for the first time by Node #2 and it is sent to Node #1 to be processed for dataset updating purposes, i.e. replication. It can happen only if Node #1 is idle, i.e. if there is a token in the place *P_n1_idle*.

The inclusion of the GSPN segment resembling the load balancer in the model shown in Fig. 1, albeit unnecessary, is made just for consistency and completeness reasons because the modeled system shows the processing of a single WRITE request.

B. *Processing of READ requests: the model*

The execution of a single READ request is being described by the GSPN model, portrayed in Fig. 3. The firing of the exponential transition *T_read_request*, which occurs with a rate of $\lambda_{\text{READ\_REQUESTS}}$, sends the READ request to the load balancer (a token in the place *P_read_request*). Initially, Node #1 is being selected for its processing.

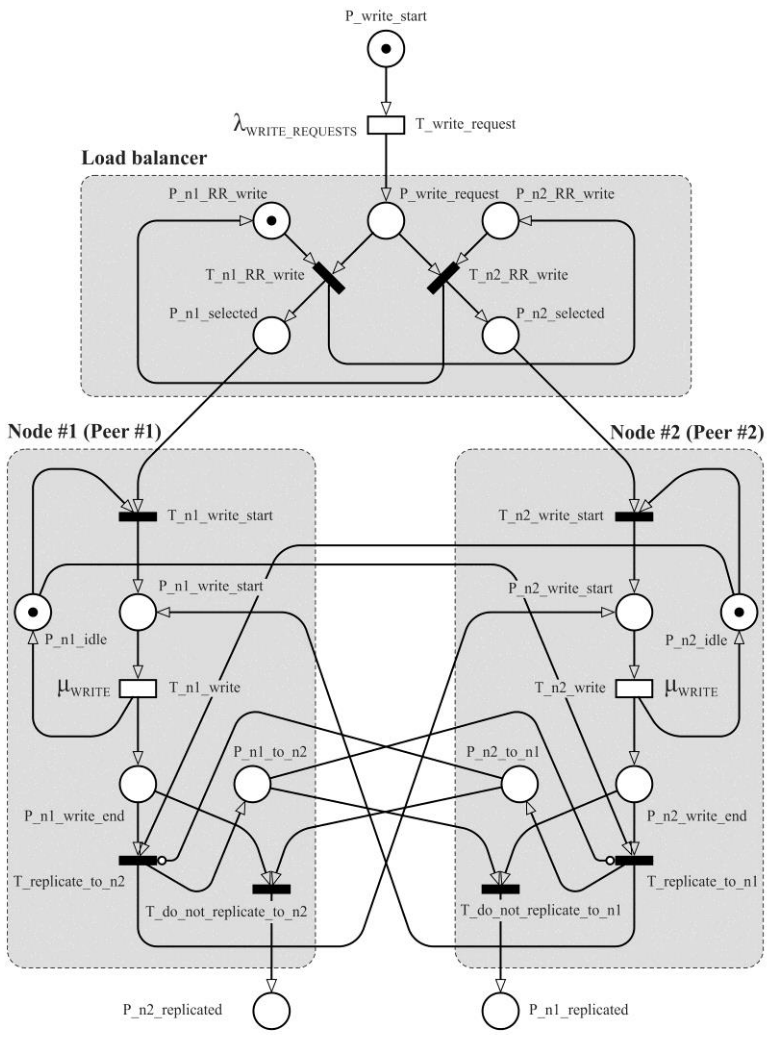

Figure 2. GSPN model of the processing of a single WRITE request in a P2P replicated architecture with two nodes.

Before execution of the reading operation, the searching operation upon data that need to be read is being carried out on the dataset stored within Node #1, i.e. a token in the place *P_n1_search_start* is being put if the Node #1 is idle (a token in the place *P_n1_idle*). The firing of the exponential transition *T_n1_search* occurs with a rate of μ_{SEARCH}, which puts a token back to the place *P_n1_idle*, and also puts a token into the place *P_n1_search_end*. The outcome of the searching operation can be twofold: (a) either the searched data has been found (immediate transition *T_n1_found* becomes enabled, and fires with a probability of p_{FOUND}), or (b) the searched data has not been found (immediate transition *T_n1_not_found* becomes enabled, and fires with a probability of $1 - p_{FOUND}$). If the latter one happens, a token will be put in the place *P_n1_not_found1*. However, if the immediate transition *T_n1_found* fires, the reading operation starts (a token in the place *P_n1_read_start* enables the exponential transition *T_n1_read*). The reading of data lasts, on average, $1/\mu_{READ}$ units of time. After the exponential transition *T_n1_read* fires, a token is being put in the place *P_n1_read_end*. In this particular case, the immediate transition *T_OR1* becomes enabled, since there are no tokens in the place *P_n2_not_found2* (i.e. the READ request has not been processed by Node #2).

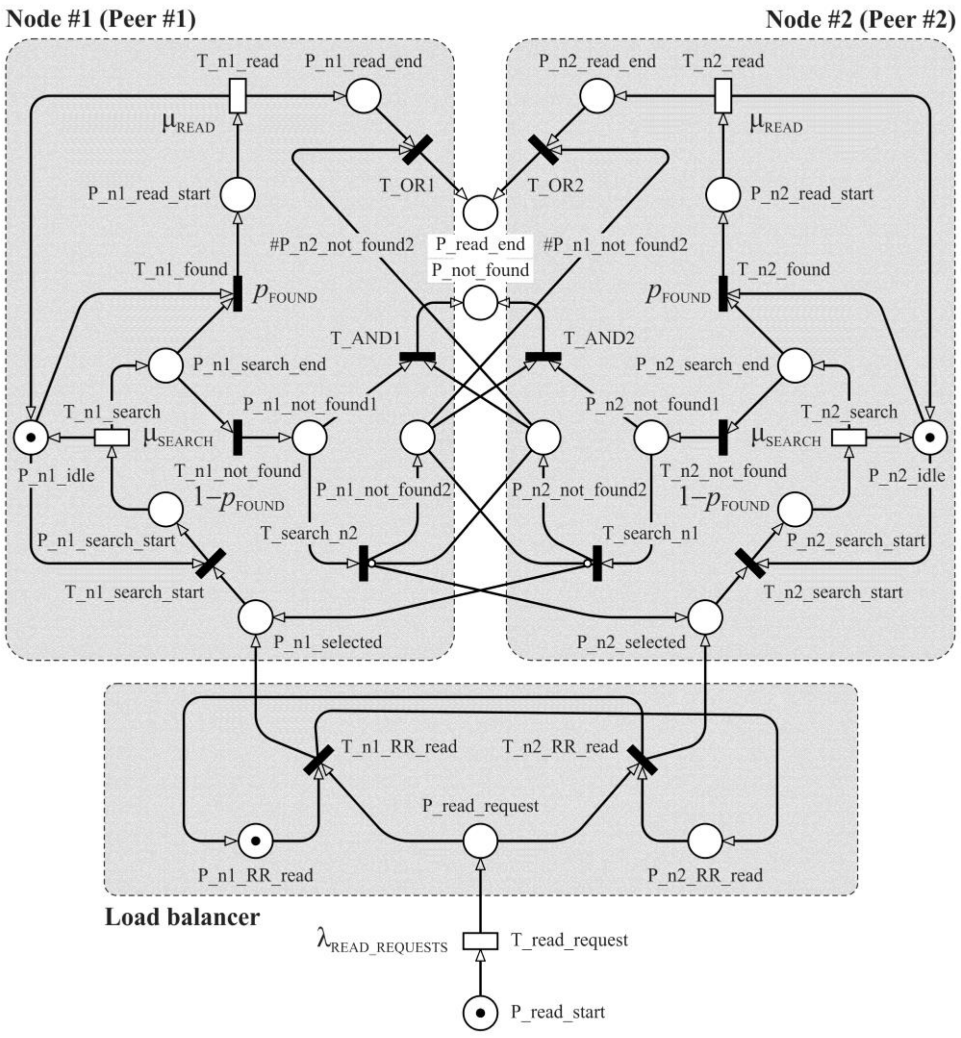

Figure 3. GSPN model of the processing of a single READ request in a P2P replicated architecture with two nodes.

Note that the arc originating from this place to transition *T_OR1* has a multiplicity of *#P_n2_not_found2*, i.e. it is equal to the number of tokens currently present in the place *P_n2_not_found2*. The firing of the immediate transition *T_OR1* puts a token into the terminal place *P_read_end*.

If the searched data is not found in the dataset residing on Node #1, the searching operation is being executed on the dataset residing on Node #2, because the READ request can be processed by any peer. This is accomplished by the firing of the immediate transition *T_search_n2*, which takes the token out from the place *P_n1_not_found1* and puts it into the place *P_n2_selected*. This transition is being enabled due to the existence of an inhibitor arc coming from the place *P_n2_not_found2*, and due to the absence of a token in this place, meaning that the search operation has not been carried out on Node #2 yet.

Given a token in the place *P_n2_selected*, the immediate transition *T_n2_search_start* fires, but only if Node #2 is idle (a token in the place *P_n2_idle*). The firing of this transition puts a token into the place *P_n2_search_start*, which enables the exponential transition *T_n2_search*. It fires with a rate of μ_{SEARCH} and puts a token into the place *P_n2_search_end*. Now two immediate transitions become concurrently enabled: the transition *T_n2_found* (meaning that the searched data has not been found within the dataset held in Node #2) and the transition *T_n2_not_found*. The first one fires with a probability of p_{FOUND}, and the latter one with a probability of $1 - p_{\mathrm{FOUND}}$.

If the searched data is found on Node #2, the reading operation starts by placing a token into the place *P_n2_read_start*, which enables the exponential transition *T_n2_read*, but only if the Node #2 is idle at that moment (a token in the place *P_n2_idle*). After $1/\mu_{\mathrm{READ}}$ units of time, the exponential transition *T_n2_read* fires and puts a token into the place *P_n2_read_end*. Because there is already a token residing in the place *P_n1_not_found2* (the reading operation failed on Node #1 because the searched data were not found), the immediate transition *T_OR2* becomes enabled and immediately fires, by putting a token into the terminal place *P_read_end*.

On the other hand, if the searched data is not found on Node #2 (a token in the place *P_n2_not_found1*), after it was not found also on Node #1 (a token in the place *P_n1_not_found2*), the immediate transition *T_AND2* becomes enabled and fires, by taking out the tokens from both previously mentioned places and by putting a token in the terminal place *P_not_found*. Note that, given a token in the place *P_n2_not_found1*, the immediate transition *T_search_n1* remains disabled, because of the existence of an inhibitor arc originating from the place *P_n1_not_found2*, which already contains a token (the searched data was not found in the dataset residing on Node #1).

VI. Discussion

The GSPN sub-model related to the processing of the WRITE request contains a single starting place (*P_write_start*) and two terminal places (*P_n1_replicated* and *P_n2_replicated*). A token in the place *P_n1_replicated* means that the writing operation has been initially executed on Node #1 and afterward the WRITE request has been also processed on Node #2 so that both dataset replicas become identical. In the same manner, a token in the place *P_n2_replicated* means that the writing operation has been initially executed on Node #2, and subsequently the WRITE request has been also processed by Node #1 so that both dataset replicas become identical. It should be also notified that this particular sub-model does not include/model the operation of accessing the dataset before the execution of the writing operation, for simplicity purposes.

The GSPN sub-model related to the processing of a READ request contains a single starting place (*P_read_start*) and two terminal places (*P_read_end* and *P_not_found*). A token in the place *P_read_end* means that the READ request has been successfully addressed by either of the two peers, regardless of which one of them initially processed the request. On the other hand, a token in the place *P_not_found* means that none of the peers contain the searched data to be read in their replicated datasets, so none of them can successfully address the READ request.

The list of input parameters for both GSPN sub-models, as well as their meaning, is given in Table I.

The verification process of the two proposed GSPN sub-models, which includes structural analysis (estimation of the state-space, evaluation of traps and siphons, finding out P- and t-invariants), has been carried out using TimeNET 4.5 [9], but other dedicated software packages, such as PIPE 2 [10] and GreatSPN 2.0 [11] can be utilized for this purpose, as well. It proved that both sub-models are accurate and capture the intrinsic logic and behavior of the real P2P replicated system with two nodes. However, the validation process, which aims at proving the credibility of the sub-models, has yet to be performed, by comparing the results of the stationary and transient analysis against a real P2P replicated system with two nodes/peers.

TABLE I. Input parameters.

Parameter	Meaning
$\lambda_{\mathrm{WRITE\_REQUESTS}}$	WRITE requests' arrival rate
μ_{WRITE}	WRITE requests' processing rate
$\lambda_{\mathrm{READ\_REQUESTS}}$	READ requests' arrival rate
μ_{SEARCH}	SEARCH operation processing rate
μ_{READ}	READ requests' processing rate
p_{FOUND}	Probability of finding the searched data during the execution of READ requests

The next process to be conducted is a performance analysis (both stationary and transient). The stationary analysis is particularly important, since it results in obtaining the steady-state probabilities of the modeled systems, either analytically, or by computer simulation.

Finally, both GSPN sub-models can be slightly modified and merged into a single model to capture the behavior of the P2P replicated system under simultaneous READ and WRITE requests. This could lead to a complex GSPN model in which the only common building elements should be the places *P_n1_idle* and *P_n2_idle*.

VII. Conclusion

The paper contributes towards the performance evaluation of Peer-to-Peer replicated architectures, often utilized within the Big Data paradigm, by presenting suitable GSPN-based models of the execution of the READ and WRITE requests. The proposed GSPN-based models of the two types of request can provide significant insights vis-à-vis the performance of the modeled system, by obtaining the following performance metrics for various operating scenarios: the average response time, the average queue lengths, the average number of the READ and WRITE requests waiting in queues to be processed by peers, peers' throughput as a function of requests' arrival rate, peers' utilization, etc.

In this particular case, there are three obvious limitations associated with GSPN modeling. The first one states that the hereby presented GSPN models do not allow for distinguishing among different types of WRITE requests (insertion, update, and delete operations), so if there is a need for their representation, each of them should be modeled by a distinct GSPN substructure, which would considerably increase the complexity of the overall GSPN model. Next, GSPNs do not allow for modeling other load balancing approaches, except the 'Round Robin' and the 'Ad Hoc' schemes. The third limitation refers to the fact that GSPN-based models of the READ and WRITE requests would become increasingly complex as the number of portrayed peers rise. The increased complexity of the GSPN model can incur computational intractability and an inability to successfully evaluate any performance metrics.

Future work will be directed towards a performance analysis of the proposed GSPN sub-models vis-à-vis various input parameters, i.e. various operating scenarios, and conveying a comparative analysis of the obtained results with those yielded by the performance analysis of corresponding GSPN-based sub-models representing the execution of the READ and WRITE requests within a Master/Slave replicated architecture.

References

[1] Dugan, J. B., & Ciardo, G. (1997). *Stochastic Petri Net analysis of a replicated file system.* Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.16.748&rep=rep1&type=pdf

[2] Simonet, A., Fedak, G., Ripeanu, M. (2012). *Active Data: A programming model for managing Big Data life cycle*. Research Report No. 8062. Inria, Le Chesnay Cedex, France.

[3] Chattaraj, D., Sarma, M., & Samanta, D. (2019). Stochastic Petri net based modeling for analyzing dependability of big data storage system. In *Emerging Technologies in Data Mining and Information Security* (pp. 473-484). Springer, Singapore.

[4] Chiang, D.-L., Wang, S.-K., Wang, Y.-Y., Lin, Y.-N., Hsieh, T.-Y., Yang, C.-Y., Shen, V. R. L., & Ho, H.-W. (2021): Modeling and analysis of Hadoop MapReduce systems for Big Data using Petri Nets. *Applied Artificial Intelligence*, *35*(1), 1−25.

[5] Rizwan Ali, M., Ahmad, F., Hasanain Chaudary, M., Ashfaq Khan, Z., Alqahtani, M. A., Saad Alqurni, J., Ullah, Z., & Khan, W. U. (2021). Petri Net based modeling and analysis for improved resource utilization in cloud computing. *PeerJ Computer Science*, *7*, e351.

[6] Erl, T., Khattak, W., & Buhler, P. (2015). *Big Data fundamentals: Concepts, drivers and techniques*. North Vancouver, Canada: Prentice-Hall/Arcitura Education, Inc..

[7] Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., & Franceschinis, G. (1995). *Modelling with Generalised Stochastic Petri Nets*. Hoboken, NJ, USA: John Wiley & Sons, Inc..

[8] Balbo, G. (2007, May). Introduction to generalized stochastic Petri nets. In *International school on formal methods for the design of computer, communication and software systems* (pp. 83-131). Springer, Berlin, Heidelberg.

[9] *TimeNET 4.5 Official Website* (2021). Available at: https://timenet.tu-ilmenau.de/#/

[10] *PIPE 2 Official Website*. Available at: http://pipe2.sourceforge.net/

[11] *GreatSPN 2.0 Official Website* (2008). Available at: http://www.di.unito.it/~greatspn/ index.html