Indonesian Journal of Electrical Engineering and Computer Science
Vol. 24, No. 3, December 2021, pp. 1596~1603
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v24.i3.pp1596-1603

Deadlock detection in distributed system

Kshirod Kumar Rout[1], Debani Prasad Mishra[2], Surender Reddy Salkuti[3]
[1,2]Department of Electrical Engineering, IIIT Bhubaneswar, Odisha, India
[3]Department of Railroad and Electrical Engineering, Woosong University, Daejeon, Republic of Korea

Article Info

Article history:

Received Jun 12, 2021
Revised Sep 25, 2021
Accepted Oct 6, 2021

Keywords:

Deadlock
Preemption
Request available
Wait-for-graph

ABSTRACT

In highly automated devices, deadlock is a case that occurs when no system can permit its event which may give irrelevant economic losses. A process can request or release resources that are either available or are on hold by others. If a process requesting a resource is not available at any time, then that process enters into the waiting state. But if a waiting state is not converted into its present state, it enters more than two processes are having an indefinite waiting state. The proposed algorithm gives an efficient way for deadlock detection. For the implementation of this work, C++ and python as the basic programming language are used. It gives an idea about how resources are allocated, and how few processes result in deadlock.

Corresponding Author:

Surender Reddy Salkuti
Department of Railroad and Electrical Engineering, Woosong University
17-2, Jayang-Dong, Dong-Gu, Daejeon-34606, Republic of Korea
Email: surender@wsu.ac.kr

1. INTRODUCTION

Deadlocks are one of the vital issues in concurrent programming [1]. It happens when some tasks are locked forever because their requests for resources will never be satisfied. To preserve concurrency, we must get deadlocks under control [2], [3]. Taking into consideration two trains approaching each other on the same one and only track, none of both trains can move forward when they are in front of each other. In the same way, the situation occurs in an OS. Deadlock occurs when there are greater than or equal to two processes holding some resources and waiting for resources held by other processes. Through the example, deadlock can be well explained. Resource 1 is held by process 1 and this process is waiting for resource 2 which is held by process 2, and process 2 is, in turn, waiting for resource 1 [4], [5].

There are various types of deadlocks. Resource deadlock is the kind of deadlock that occurs when 2 programs are having a common resource and stop each other from using that resource which ultimately leads to the termination of the two programs. Another type is Communication deadlock which is a kind of deadlock that occurs when process 1 is trying to send a message to process 2 which is trying to send a message to process 3 which in turn sends a message to A. The Necessary circumstances for a deadlock can be mutual exclusion, hold and wait, no preemption, and circular wait [6], [7]. A mutual exclusion means that only a single process can work at one time [8], [9]. 'Hold and wait' means that the process while holding multiple resources, can request more resources from other processes which are holding those resources. No preemption means that unless the process releases the resource, it cannot be taken from a process [10], [11]. A bunch of processes waiting in a circular form is referred to as the circular wait. According to an example, resource 2 is allocated to process 1 which is requesting for resource 1 [10], [11]. In the same way, resource 1 is allocated to process 2 which is requesting for resource 2. This produces a circular wait loop and leads to a

deadlock. The operating system (OS) considers resources or processes to recover the system from deadlocks. In resources, on preempting the resource we will capture one among the resources from the owner of the resource (process) and [10] provides it to the opposite process with the expectation that it will complete the execution and will release this resource sooner. For rollback to a safe state, to enter into the deadlock state, the OS passes through multiple stages. The OS can roll back the system to its earlier safe state. Thus, checkpointing is done by the OS at every stage. While in process, two basic concepts are involved. For killing a process, the big issue is to decide which process shall be terminated [11], [12]. The OS kills a process that has done the smallest quantity of labor till now. While killing all processes will cause inefficiency within the system because all the processes will execute again from starting [12].

According to the literature survey, various algorithms in distributed deadlock detection have been suggested. Based on how wait-for-graph data is conserved and the search for cycles is performed, detection of deadlock algorithm is categorized into 3 categories as centralized control, distributed control, and hierarchical control. Distributed computing is a study in computer science that deals with multiple systems by sharing data and resources among themselves [13], [14]. According to the Ho-Ramamoorthy algorithm (centralized algorithm), He represented algorithms in two ways known as two-phase and one-phase algorithms. These algorithms keep all the collected status reports and store them into particular tables for the control site to have a consonant view of the system. The two-phase algorithm is a type in the centralized approach of deadlock detection. A status table that contains the status of all processes is initiated at each particular site. The status includes resources in waiting for state or resources which are locked [15]. First, the status table from all sites is requested by the central controlling site. From the received information, a state graph is constructed. Then it searches for cycles [16], [17]. If no cycle exists, we can say deadlock is not there in the system. Otherwise, the control site demands status tables from the other sites and forms a state graph based on common undertakings. But the system is declared in a deadlock state on detecting the same cycle again. The one-phase algorithm is a type in the centralized approach of deadlock detection. Two status tables given as resource status and process tables are maintained by every site [16]. The resource table keeps the transaction details that have been locked or are in a waiting state for resources stored by other sites. The record of resources that are locked by or waited for by the transactions at the specific site is maintained by the Process status table [16]-[18]. Both the status tables from all other sites are requested by the central controlling site [17], [18]. From the received information, a state graph is constructed. Then it searches for cycles. If no cycle exists, we can say deadlock is not detected there in the system. Otherwise, the system is deadlocked. But, talking about the advantages and disadvantages, this algorithm is easy to implement and simple to understand. However, it takes longer response time and delays, over clogging of connecting links and bigger communication overheads over the control sites.

According to the Chandy-Misra-Haas edge chasing algorithm (distributed algorithm) [19], the blocked process uses a special message called as ‘probe’. The Algorithm uses a probe message to detect the presence of cycles. The probe is a three variables (i,j,k) data termed as triplet [20]. Pi is the blocked process site, Pj is the process that is sending the message, Pk is the process to which the message was sent [21]. The blocked process site receives the probe message, keeping the requested resources and forwarding the rest to the next process. If the received probe is the same as the original probe that the blocked process site has sent, the deadlock exists, otherwise, Deadlock doesn’t exist [22]. It is beneficial. Here, it can be started by any site. Deadlock detection can begin anytime if a process is in an indefinite wait state. Information about state graph can be circulated by any means like probe, string, or graph [23]. But sometimes in transaction-wait-for (TWF) graph-based algorithms, sometimes unnecessary message transfer happens giving rise to duplication of deadlocks detection jobs. In probe-based algorithms, overhead is low. It is easy to implement [24].

According to Menasce-Muntz’s algorithm (Hierarchical algorithm) [25], sites are organized in a tree structure. The bottom-level controllers manage the wait-for-graph (WFG) known as leaf. This leaf allocates the resources. The rest called as non-leaf nodes that detect deadlock [26]. These nodes act in a process of inheritance where they can detect the deadlocks only in their children's leaf controllers [27]. Here each parent node maintains a global WFG, which is a union of WFG’s of its children. Whenever a change happens in the graph, it is propagated to its parent node [28]. Before that, the non-leaf controllers are maintained with updated WFG from time to time. The root controller termed as parent applies respective alterations in its TWF graph, looks for the cycles if any exists to detect deadlocks. This approach can provide efficient deadlock detection as it coincides with resource access patterns to the cluster of sites. It reduces the communication cost and dependence on central sites. But if deadlocks are over several clusters, it is more complicated to implement which will be inefficient. But probability for such a situation is less as all the computations are step by step increasing its productivity and reliability [29], [30].

2. RESEARCH METHOD

The proposed algorithm for the detection of deadlock is explained in detail below. Here C++ and python are used as the programming language.

- Step 1: We have simulated software that is used to detect the deadlock in an office environment. The operator of the software is a resource administrator and checks for the availability of resources based on the resources available at the current time.
- Step 2: There are multiple instances of resources available in the office and there are 4 employees in the office using the resources.
- Step 3: They are requesting the use of resources at any given time which the administrator knows and enters as input to the software.
- Step 4: The software at that time randomly distributes the resources, i.e., the resources on hold by employees based on the total number of resources available. We have used rand() to randomly distribute the resources among the employees.
- Step 5: After that, we calculate the resources available and take the resources requested by users as input and check for the possibility of assigning the resources by taking counters if the resources requested by a particular person is less, i.e., the number of printers, scanners, tape drive, fax then the counter value is increased till 4. If the counter value turns out to be 4 then resources are assigned to the person and the total resources available at that time are added upon by the person's resources in the available matrix. If the counter does not go up to 4 in any case then deadlock is detected.
- Step 6: Now coming to different matrices we have used:
  a) fraction[]: It is used for randomly storing the number of resources allocated at any time.
  b) request[]: It is used for storing the resources requested by users at any time.
  c) srand[]: It is used for generating a seed at a particular time otherwise if we use rand() in C++ directly then it will always return the same random numbers after executing multiple times. So by using srand() we will be generating random numbers.
  d) available[]: It is used to check for the resources available by subtracting the total number of resources available with the total number of resources on hold.

3. RESULTS AND DISCUSSION

In this section, it is explained the results of the research and at the same time is given a comprehensive discussion. Results are presented in figures, graphs, tables. According to the below examples, a case study is taken as the problem statement to check deadlock if exists. Here, a scenario of a corporate office is taken into consideration. The employees use the systems such as printers, fax machines, scanners, and drivers for data transmission and other works. These devices are occupied or hold by the users. Here, four employees are considered as the users who utilize the above-said devices as resources. Various cases are explained below where deadlock may or may not occur.

3.1. When the system will be in a deadlock

Table 1 shows the idea of a deadlock state. Here a real-life scenario of an office is taken where four employees are working in a firm. There exist a few systems and devices which are used by them. But due to certain issues, the devices are not properly utilized among all the employees. The resources requirement are more than the availability. And thus due to that some devices are used by some employees for indefinite duration causing the condition of deadlock.

Figure 1 depicts the different resources held by the various employees i.e Kumar, Harsh, Vipul, and Ramesh. Here scanner and driver systems are already on hold and therefore no more systems are available which can be used by other employees. The resource requirements are still not satisfied, and thus the employees have to wait for others to complete their tasks. Due to this, a deadlock occurs in the system.

According to this example, Table 2 shows the idea of a deadlock state. There also exist few system and devices which are used by employees. But due to certain issues, the devices are not properly used among all the employees. The resources requirement are more than the availability. And some resources are held by other employees and thus due to that some systems are used by some employees for indefinite duration causing the condition of deadlock.

Figure 2 depicts the different resources held by the various employees. This is also a deadlock state as the scanner system already on hold which is no more available. This can't be used by other employees for usage. But the resource requirement is still not over by the employees. Therefore, others have to wait to use this function which may lead to an indefinite time duration. Hence it causes a cycle and thus deadlock occurs.

Table 1. Deadlock state when resources are on hold and not available

	Employees	Printer	Fax	Scanner	Driver
Input					
Available Resources	Kumar Harsh Vipul Ramesh	3	4	2	1
		5	6	1	2
		4	5	2	2
		2	2	3	3
		6	7	5	4
Output					
Resource on hold by employees	Kumar Harsh Vipul Ramesh	1	2	4	0
		1	1	0	2
		2	0	1	2
		1	1	0	0
Total Resource on hold		2	4	5	4
Total Resource available		1	3	0	0
Resource required by employees	Kumar Harsh Vipul Ramesh	3	4	2	1
		5	6	1	2
		4	5	2	5
		5	5	3	5

The system is in Deadlock

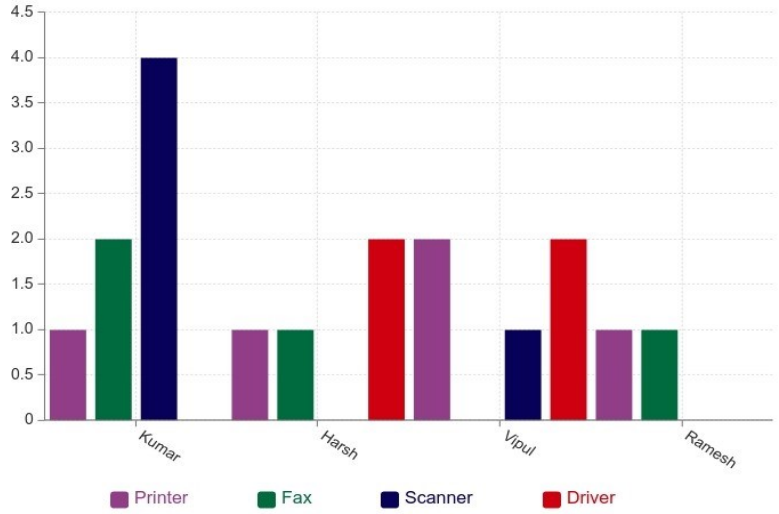

Figure 1. Resource on hold plot

Table 2. Deadlock states when the requirement is not satisfied and resources are not available

	Employees	Printer	Fax	Scanner	Driver
Input					
Available Resources	Kumar Harsh Vipul Ramesh	2	3	1	1
		3	2	1	4
		4	2	2	1
		2	3	2	1
		4	4	4	4
Output					
		0	0	0	0
		2	1	1	1
		1	1	1	0
		0	1	2	1
Resource on hold by employees	Kumar Harsh Vipul Ramesh				
Total Resource on hold		3	3	4	2
Total Resource available		1	1	0	2
Resource required by employees	Kumar Harsh Vipul Ramesh	2	3	1	4
		3	2	1	4
		4	2	2	1
		2	3	2	1

The system is in Deadlock

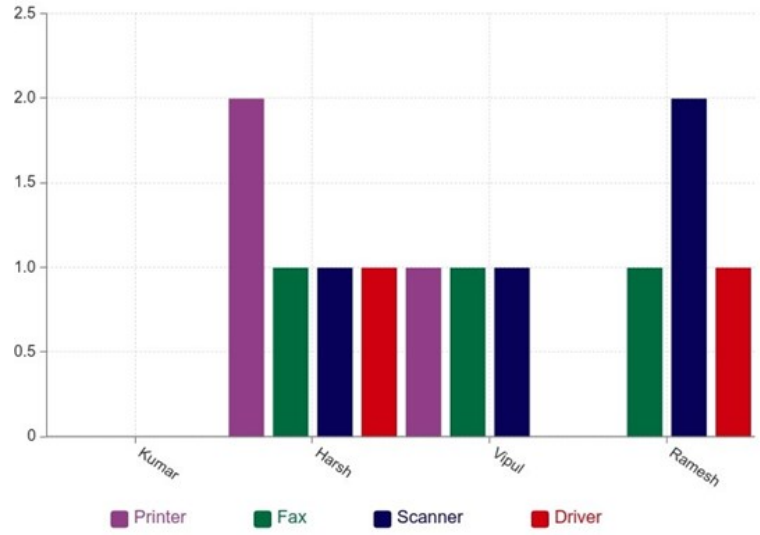

Figure 2. Resource on hold plot

3.1. When the system will not be in a deadlock state

Table 3 shows the different resource allocations to various employees as per the requirements. As all the employees have used the resources and none of them have to wait for others for a longer duration. The available resources are more than the requirement. Thus, this system is not in a deadlock state.

Table 3. Case study for not a deadlock state

	Employees	Printer	Fax	Scanner	Driver
Input					
Available Resources	Kumar Harsh Vipul Ramesh	1	1	2	0
		1	1	1	1
		1	0	1	0
		0	1	0	0
		5	5	4	4
Output					
Resource on hold by employees	Kumar Harsh Vipul Ramesh	1	1	0	2
		0	1	2	1
		2	1	0	0
		1	0	2	0
Total Resource on hold		4	3	4	3
Total Resource available		1	2	0	1
Resource required by employees	Kumar Harsh Vipul Ramesh	1	1	2	0
		1	1	1	1
		1	0	1	0
		0	1	0	0
The system is not in Deadlock					

Figure 3 depicts the resources held by the various employees as per the requirements. The resource requirements by the employees are satisfied at last. The requirement by the employees is less than. Thus, this system is not deadlocked.

Table 4 presents a scenario of no deadlock state as available resources are distributed efficiently as requirements are fulfilled now. And at least one device is not on hold, so that rest can be swapped among themselves. This will not create any cycle and thus no deadlock exists.

Figure 4 shows the different resources held by the various employees as per the requirements. At last, the resource requirements by the employees are satisfied, as all the employees have used the resources and allocations are well distributed. Here, none of them have to wait for others for a longer duration at the end. Thus, this system is also not in a deadlock state.

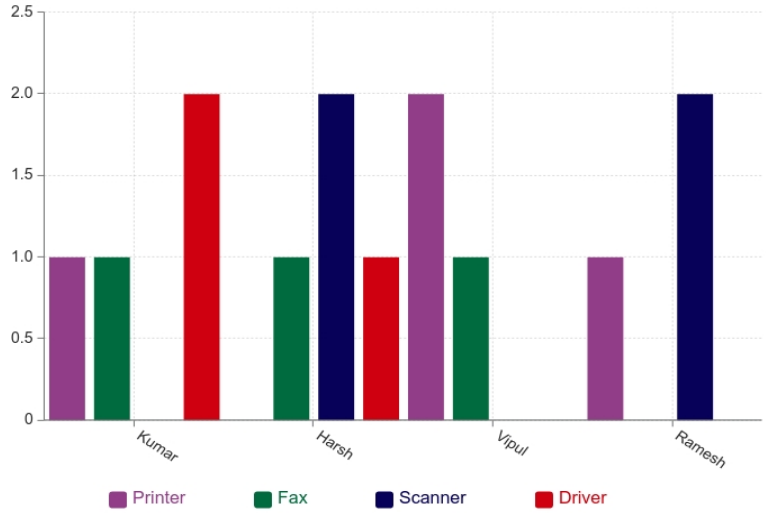

Figure 3. Resource on hold plot

Table 4. Example showing no deadlock condition

	Employees	Printer	Fax	Scanner	Driver
Input					
Available Resources	Kumar Harsh Vipul Ramesh	1	3	2	1
		3	0	1	1
		0	2	1	0
		0	0	2	0
		6	7	5	4
Output					
Resource on hold by employees	Kumar Harsh Vipul Ramesh	2	1	1	2
		0	2	1	0
		0	0	1	0
		2	3	1	0
Total Resource on hold		4	6	4	2
Total Resource available		2	1	1	2
Resource required by employees	Kumar Harsh Vipul Ramesh	1	3	2	1
		3	0	1	1
		0	2	1	0
		0	0	2	0

The system is not in Deadlock

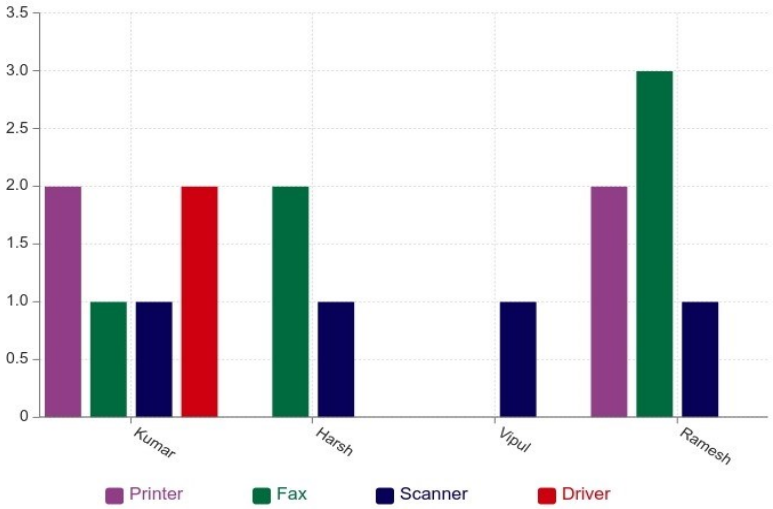

Figure 4. Resource on hold plot

4. CONCLUSION

The proposed deadlock detection mechanism worked perfectly fine for different sections of a distributed system. The performance of the proposed technique was measured in terms of time, space, the number of messages received/sent. The proposed algorithm gave an efficient way for deadlock detection. The work implemented has used C++ and python as the basic programming language. This software was simulated to detect the deadlock in an office environment. At one particular time, the resources were distributed randomly, i.e., the resources were on hold by employees based on the total number of resources available and if any employee was in a waiting state for the resources, the scenario is termed as in deadlock. It gave an idea about how resources were allocated, and how few processes resulted in a deadlock state.

In the future scope, as the number of employees is hard-coded, we can make it dynamic later. The types of resources can be made dynamic in an office scenario (types of resources hard-coded) based on the new resources brought into the company. However, it can be made universal by applying this software in rental stores such as clothing rental stores. A better user interface (UI) can be build by making it more graphical rather than terminal-based. We can improve the space complexity by using recursive functions. A better representation like live statistics can be integrated with UI.

ACKNOWLEDGEMENTS

This research work was funded by "Woosong University's Academic Research Funding-2021".

REFERENCES

[1] M. Singhal, "Deadlock detection in distributed systems," in *Computer*, vol. 22, no. 11, pp. 37-48, Nov. 1989, doi: 10.1109/2.43525.

[2] S. Lee, "Fast, centralized detection and resolution of distributed deadlocks in the generalized model," in *IEEE Transactions on Software Engineering*, vol. 30, no. 9, pp. 561-573, Sept. 2004, doi: 10.1109/TSE.2004.51.

[3] S. Lee, "Efficient generalized deadlock detection and resolution in distributed systems," *Proceedings 21st International Conference on Distributed Computing Systems*, 2001, pp. 47-54, doi: 10.1109/ICDSC.2001.918932.

[4] S. Lee and J. L. Kim, "Performance analysis of distributed deadlock detection algorithms," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 4, pp. 623-636, July-Aug. 2001, doi: 10.1109/69.940736.

[5] S. Selvaraj and R. Ramasamy, "An Efficient Detection and Resolution of Generalized Deadlocks in Distributed Systems," *International Journal of Computer Applications*, vol. 19, no. 1, 2010, doi: 10.5120/412-610.

[6] Y. Cai and W. K. Chan, "MagicFuzzer: Scalable deadlock detection for large-scale applications," *2012 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 606-616, doi: 10.1109/ICSE.2012.6227156.

[7] Z. Li and M. Zhao, "On Controllability of Dependent Siphons for Deadlock Prevention in Generalized Petri Nets," in *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 38, no. 2, pp. 369-384, March 2008, doi: 10.1109/TSMCA.2007.914741.

[8] Z. Li and M. Zhou, "Elementary siphons of Petri nets and their application to deadlock prevention in flexible manufacturing systems," in *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 34, no. 1, pp. 38-51, Jan. 2004, doi: 10.1109/TSMCA.2003.820576.

[9] F. Henskens and M. G. Ashton, "Graph-based Optimistic Transaction Management", *Journal of Object Technology,* vol. 6, no. 6 pp. 131-148, 2007, doi: 10.5381/jot.2007.6.6.a4.

[10] B. M. M. Alom, F. A. Henskens and M. R. Hannaford, "Optimization of Detected Deadlock Views of Distributed Database," *2010 International Conference on Data Storage and Data Engineering*, 2010, pp. 44-48, doi: 10.1109/DSDE.2010.41.

[11] Z. Li and M. Zhou, "Two-Stage Method for Synthesizing Liveness-Enforcing Supervisors for Flexible Manufacturing Systems Using Petri Nets," in *IEEE Transactions on Industrial Informatics*, vol. 2, no. 4, pp. 313-325, Nov. 2006, doi: 10.1109/TII.2006.885185.

[12] K. Chakma, A. Jamatia, and T. Debbarma, "An Analysis and Improvement of Probe-Based Algorithm for Distributed Deadlock Detection," *Lecture Notes on Software Engineering*, vol. 3, no. 4, pp. 285-287, 2015, doi: 10.7763/LNSE.2015.V3.205.

[13] F. Mohanty, S. Rup, B. Dash, B. Majhi, and M. N. S. Swamy, "Mammogram classification using contourlet features with forest optimization-based feature selection approach," *Multimedia Tools and Applications*, vol. 78, pp. 12805–12834, 2018, doi: 10.1007/s11042-018-5804-0.

[14] W. B. Daszczuk, "Communication and Resource Deadlock Analysis using IMDS Formalism and Petri Nets," *The Computer Journal*, vol. 60, no. 5, pp. 729–750, 2017, doi: 10.1093/comjnl/bxw099.

[15] D. Y. Chao and Z. Li, "Structural conditions of systems of simple sequential processes with resources nets without weakly dependent siphons," *IET Control Theory and Applications*, vol. 3, no. 4, pp. 391-403, 2009, doi: 10.1049/iet-cta.2007.0470.

[16] G. Liu, Z. Li, C. Zhong, "New controllability condition for siphons in a class of generalized Petrinets," *IET Control Theory and Applications*, vol. 4, no. 5, pp. 854-864, 2010, doi: 10.1049/iet-cta.2009.0264.

[17] S. D. Kalita, M. Kalita, and S. Sarmah, "A Survey on Distributed Deadlock Detection Algorithm and its performance Evolution," *International Journal of Innovative Science, Engineering Technology*, vol. 2, no. 4, pp. 615-620, Apr. 2015.
[18] R. Cordone, L. Ferrarini, and L. Piroddi, "Enumeration algorithms for minimal siphons in Petri nets based on place constraints," in *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 35, no. 6, pp. 844-854, Nov. 2005, doi: 10.1109/TSMCA.2005.853504.
[19] D. H. Ahn, D. C. Arnold, B. R. de Supinski, G. L. Lee, B. P. Miller, and M. Schulz, "Overcoming Scalability Challenges for Tool Daemon Launching," *2008 37th International Conference on Parallel Processing*, 2008, pp. 578-585, doi: 10.1109/ICPP.2008.63.
[20] K. M. Chandy, J. Misra, and L. M. Haas, "Distributed Deadlock Detection," *ACM Transactions on Computer Systems*, vol. 1, no. 2, pp. 144-156, 1983, doi: 10.1145/357360.357365.
[21] Y-S. Huang, Y-L. Pan, and P-J. Su, "Transition-Based Deadlock Detection and Recovery Policy for FMSs Using Graph Technique," *ACM Transactions on Embedded Computing Systems,* vol. 12, no. 1, pp. 1-13, 2013, doi: 10.1145/2406336.2406347.
[22] D. Warneke and O. Kao, "Exploiting Dynamic Resource Allocation for Efficient Parallel Data Processing in the Cloud," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 6, pp. 985-997, June 2011, doi: 10.1109/TPDS.2011.65.
[23] F. Vernadat, B. Berthomieu, F. Vernadat, and B. Berthomieu, "Time Petri Nets Analysis with TINA," *Third International Conference on the Quantitative Evaluation of Systems - (QEST'06)*, 2006, pp. 123-124, doi: 10.1109/QEST.2006.56.
[24] S. Srinivasan and R. Rajaram, "A decentralized deadlock detection and resolution algorithm for generalized model in distributed systems," *Distributed and Parallel Databases*, vol. 29, pp. 261–276, 2011, doi: 10.1007/s10619-011-7078-7.
[25] P. Sapra, S. Kumar, and R. K. Rathy, "Deadlock Detection and Recovery in Distributed Databases," *International Journal of Computer Applications*, vol. 73, no. 1, pp. 32-36, 2013, doi: 10.5120/12708-9509.
[26] H. Wu, W-N. Chin, and J. Jaffar, "An efficient distributed deadlock avoidance algorithm for the AND model," in *IEEE Transactions on Software Engineering*, vol. 28, no. 1, pp. 18-29, Jan. 2002, doi: 10.1109/32.979987.
[27] N. Farajzadeh, M. Hashemzadeh, M. Mousakhani, and A. T. Haghighat, "An Efficient Generalized Deadlock Detection and Resolution Algorithm in Distributed Systems," *The Fifth International Conference on Computer and Information Technology (CIT'05)*, 2005, pp. 303-309, doi: 10.1109/CIT.2005.69.
[28] W. Lu, Y. Yang, L. Wang, W. Xing, and X. Che, "A Novel Concurrent Generalized Dead- lock Detection Algorithm in Distributed Systems," *International Conference on Algorithms and Architectures for Parallel Processing*, pp. 479-493, Feb. 2015, doi: 10.1007/978-3-319-27122-4_33.
[29] Z. Li and A. Wang, "A Petri Net Based Deadlock Prevention Approach for Flexible Manufacturing Systems," *Acta Automatica Sinica,* vol. 29, no. 5, pp.733-740, 2003.
[30] B. M. M. Alom, F. A. Henskens, and M. R. Hannaford, "Deadlock Detection Views of Distributed Database," *2009 Sixth International Conference on Information Technology: New Generations*, 2009, pp. 730-737, doi: 10.1109/ITNG.2009.220.