

MOSTOVYI Serhii

Khmelnytskyi National University
<https://orcid.org/0000-0002-9505-3206>
e-mail: serhii.mostovyi@khnmu.edu.ua

SYSTEM FOR PREDICTING PROCESS DEADLOCK STATES IN COMPUTER SYSTEMS

The article presents the architecture and experimental evaluation of a system for predicting process deadlock states in computer systems. In contrast to reactive mechanisms that detect cyclic resource dependencies only after a deadlock has been formed, the proposed system is intended to identify a hazardous tendency in advance by analyzing current process signatures. The architecture integrates operating-system data access, process-change monitoring, process-signature construction, current and historical data storage, automated fuzzy-rule-base construction, deadlock-state prediction, and preventive control. The training procedure based on Fuzzy C-Means clustering is separated from real-time prediction, in which priority is given to processes currently waiting for a resource. Four output classes are used: normal, potentially risky, pre-deadlock, and critical. For the pre-deadlock and critical classes, the preventive action is restricted to restarting a non-system process with the shortest residence time in the system, which reduces the amount of discarded computation while breaking a potentially dangerous resource-waiting sequence. The system was evaluated by discrete-event simulation and by experiments on a physical server. For workloads containing 5% and 10% potentially deadlocking processes, the repeated real-system experiments yielded Accuracy values of 99.82% and 99.65%, Precision of 98.39% and 98.49%, and F1-score of 98.19% and 98.25%, respectively. The prediction was generated on average 3.1-3.6 ms before the actual deadlock, while the average processing time was 8.6-8.8% lower than that of reactive resource-allocation-graph analysis.

Keywords: computer system, process, deadlock, deadlock prediction, process signature, resource state, fuzzy rules, Fuzzy C-Means, fuzzy clustering.

МОСТОВИЙ Сергій

Хмельницький національний університет

СИСТЕМА ПРОГНОЗУВАННЯ СТАНУ ВЗАЄМОБЛОКУВАННЯ ПРОЦЕСІВ У КОМП'ЮТЕРНИХ СИСТЕМАХ

У статті представлено архітектуру та результати експериментальної оцінки системи прогнозування стану взаємоблокування процесів у комп'ютерних системах. На відміну від реактивних засобів, що виявляють циклічні ресурсні залежності після їх фактичного формування, запропонована система орієнтована на завчасне визначення небезпечної тенденції за поточними сигнатурами процесів. Архітектура поєднує модулі доступу до системної інформації, спостереження за змінами процесів, побудови сигнатур, зберігання поточних та історичних даних, автоматизованого формування бази нечітких правил, прогнозування і превентивного керування. Процедур наванчання із застосуванням кластеризації Fuzzy C-Means відокремлено від оперативного прогнозування, під час якого аналізуються насамперед процеси, що очікують ресурс. Для передблукуючого та критичного станів керуючий вплив реалізовано як примусовий перезапуск несистемного процесу з найменшим часом перебування в системі. Працездатність системи перевірено імітаційним моделюванням та експериментами на реальній серверній платформі. У повторних експериментах для наборів із 5 % і 10 % потенційних взаємоблокувань отримано Accuracy 99,82 % і 99,65 %, Precision 98,39 % і 98,49 %, F1-score 98,19 % і 98,25 % відповідно. Прогноз формувався в середньому за 3,1–3,6 мс до фактичного взаємоблокування, а середній час обробки був на 8,6–8,8 % меншим порівняно з реактивним аналізом графа розподілу ресурсів.

Ключові слова: комп'ютерна система, процес, взаємоблокування, прогнозування взаємоблокування, сигнатура процесу, ресурсний стан, нечіткі правила, Fuzzy C-Means, нечітка кластеризація.

Стаття надійшла до редакції / Received 09.03.2026

Прийнята до друку / Accepted 15.04.2026

Опубліковано / Published 31.05.2026



This is an Open Access article distributed under the terms of the [Creative Commons CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/)

© MOSTOVYI Serhii

GENERAL STATEMENT OF THE PROBLEM AND ITS CONNECTION WITH IMPORTANT SCIENTIFIC OR PRACTICAL TASKS

The operation of modern computer systems is accompanied by competition among processes and threads for non-shareable or limited-access resources. When resources are mutually held and awaited, a cyclic dependency is formed and a deadlock occurs. The previous analysis [3] showed that existing approaches to deadlock prevention, avoidance, detection, and recovery either require a priori constraints on resource allocation or generate a control action only after a hazardous resource structure has emerged. This substantiates the transition from detecting the fact of a deadlock to advance assessment of the tendency toward its formation.

This research direction has been consistently developed in the author's previous studies. In [1], deadlock prediction based on process states and resource dependencies was considered; [2] addressed prediction of processes entering the deadlock state; and [3] systematized known methods and identified the limitations of reactive detection. In [4], an architectural representation of the prediction system was proposed; in [5], a model and method for constructing a process signature were formalized; and in [6], a prediction method based on process signatures, fuzzy

clustering, and fuzzy inference was developed. These results from the methodological foundation but require an integrated software implementation and experimental validation.

Recent publications by other authors confirm the relevance of the problem. For fog environments, a combination of load balancing and deadlock prediction management has been proposed [7], while DBDAA extends the Banker's algorithm for dynamic real-time safety checks [8]. In software analysis, Archerfish detects interrupt-based deadlocks in the Linux kernel [9], TDFix combines deadlock detection, localization, and repair in Java programs [10], DDMon implements formally grounded black-box detection of deadlocked distributed services [11], and refined segmentation and lock graphs improve the accuracy of dynamic detection and replay of deadlock scenarios [12].

Thus, recent studies mainly develop deadlock avoidance, detection, localization, or replay for particular environments. For operating-system processes, there remains a need for a tool that continuously analyzes resource behavior before a cycle is formed, constructs process signatures, accumulates training data, and assesses the risk of a pre-deadlock or critical state. The use of fuzzy models for assessing hazardous states [13] and the capability of Fuzzy C-Means to identify overlapping classes [14] provide a basis for such a predictive approach.

The aim of this work is to develop and experimentally evaluate a system for predicting process deadlock states on the basis of the previous results [1–6], taking into account modern approaches [7–12]. To achieve this aim, it is necessary to define the architecture and interaction of the modules; organize the acquisition of operating-system events and the construction and accumulation of signatures; separate the training procedure from real-time prediction; implement fuzzy risk assessment and a preventive control action; and evaluate the reliability and timing characteristics of the system.

PRESENTATION OF THE MAIN MATERIAL

Architecture of the prediction system.

The prediction system provides two interrelated operating modes: real-time prediction and training (Fig. 1). During real-time prediction, events are received from the operating system, process signatures are updated, the state is assessed, and, when necessary, a preventive action is generated. The training procedure uses accumulated historical signatures and includes data normalization, Fuzzy C-Means clustering, and generation of a fuzzy rule base. Separating these modes is essential: clustering is considerably more resource-intensive than applying an already generated rule base; therefore, performing it separately from real-time prediction reduces decision-making latency.

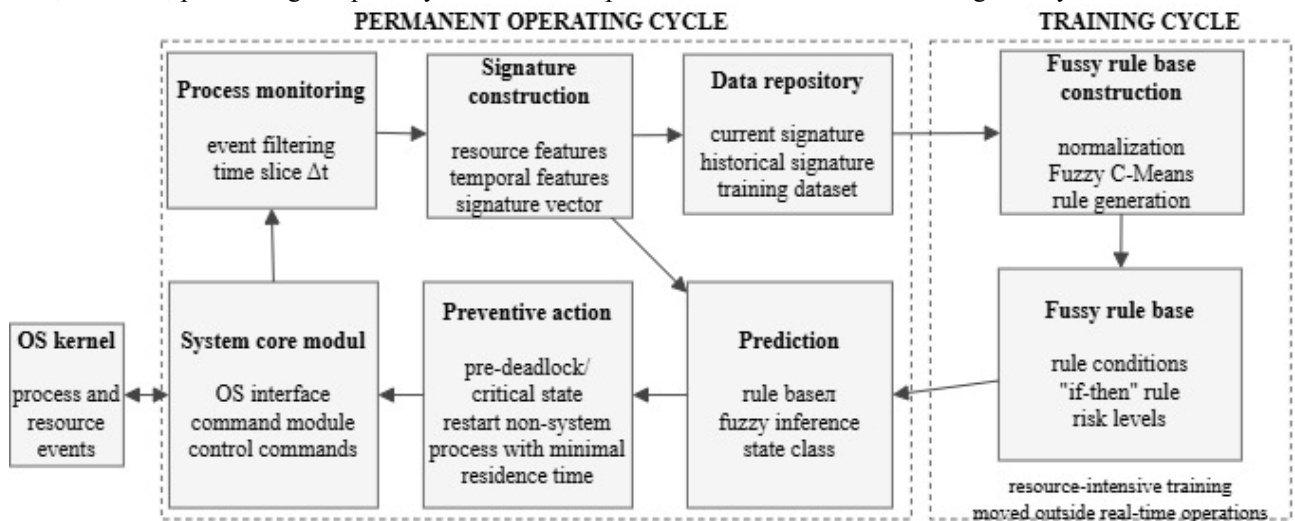


Fig. 1. Architecture of the system for predicting process deadlock states

The primary source of information is the OS kernel and the associated process-accounting, scheduling, and resource-management subsystems. The system core module obtains the list of active processes, their states, priorities, and residence times in the system, as well as information about held and requested resources. If a process has no current resource request, its state is stored for historical purposes but does not require full predictive analysis. If a request is recorded, the process becomes a candidate for real-time assessment. This selection reduces the number of calls to the prediction module because analysis is not performed for processes that do not currently form a resource dependency.

The monitoring module classifies events related to process creation, state changes, resource allocation, requests and releases, and process termination. Relevant events are accumulated during an interval Δt , after which a time slice of the system is formed. For each process, changes in parameters are checked against the previous slice; unchanged processes are not passed for repeated signature construction. The signature module transforms heterogeneous data into an ordered feature vector combining identification, resource, temporal, and behavioral characteristics. Current signatures are stored in the operational part of the repository, while completed scenarios with a known outcome are added to the historical training dataset.

After a sufficient number of training examples has been accumulated, a feature matrix is formed, followed by normalization and Fuzzy C-Means clustering. A characteristic risk level is determined for each cluster, and fuzzy “if-then” rules are generated on the basis of cluster centers and membership degrees. The rule base stores rule conditions and the corresponding risk levels. For a current signature, the real-time prediction module calculates rule activation levels, aggregates the results, and determines the process state class: normal, potentially risky, pre-deadlock, or critical.

Organization of functional module operation.

Interaction with the execution environment begins with the system-information access module. After initializing the interface, it obtains the list of active processes and, for each process, determines its identifier, current state, priority, residence time in the system, and the presence of held and requested resources. Events characterizing changes in resource interaction are recorded separately: process creation and termination, resource acquisition, generation of a new request, transition to a waiting state, and resource release. This module does not perform prediction itself; its purpose is to form a unified stream of primary data and to transmit the control command generated by the system back to the OS kernel or resource manager. This separates the analytical part from the platform-dependent method of obtaining system events.

The monitoring module operates not on a complete snapshot of the system at every cycle, but on actual changes in process states. Events are accumulated during the control interval Δt and grouped by process. For each process, previous and current parameter values are compared; if the resource state has not changed, repeated construction of a complete signature is not initiated. If a new resource request or a transition to a waiting state is recorded, the data are passed to the next stage. Such filtering is practically important for systems with a large number of simultaneously active processes because it reduces the number of repeated data-transformation operations and calls to the prediction module. At the same time, the time slice preserves the sequence of changes required to reconstruct the resource behavior of a process.

The signature-construction module transforms the obtained set of heterogeneous characteristics into a vector with a fixed structure. It includes identification, resource, temporal, and behavioral features, in particular information about occupied resources, the resource being awaited, and parameters describing process-state changes. Formalized encoding is used for resource components, making it possible to compare signatures of different processes unambiguously and apply the same normalization and fuzzy-analysis procedures to them. The generated current signature is used simultaneously for two tasks: during real-time prediction it is passed to the state-assessment module, while in the data repository it is stored to accumulate history. Thus, the same data format supports both real-time decision-making and subsequent training.

The data repository separates current and historical information. While the outcome of a resource situation is still unknown, the signature remains in the operational part. After the process terminates or the actual outcome is determined, the corresponding record is transferred to historical data and may be included in the training dataset. Before the training procedure is started, the system checks whether a sufficient number of examples has been accumulated; if the specified threshold has not been reached, the system continues to use the existing rule base for real-time prediction. This separation prevents unfinished scenarios from being mixed with examples for which it is already known whether a deadlock occurred, while also ensuring gradual expansion of the training dataset during system operation.

The training procedure is started for the accumulated set of historical signatures. First, data completeness is checked; then a feature matrix is formed and its components are normalized. Fuzzy C-Means clustering is subsequently applied, producing cluster centers and membership degrees for each training example. For each cluster, a risk level is determined from the actual outcomes of the signatures with the highest assignment to that cluster. Based on the cluster centers, assignment degrees, and established risk levels, “if-then” rules are generated automatically. Importantly, this procedure is not executed for every resource request, so its computational complexity does not directly contribute to the latency of real-time decision-making.

The fuzzy rule base is an independent information resource. For each rule, it stores the conditions, conclusion, and associated risk level. During a real-time request, the prediction module receives only the data required for fuzzy inference. If the rule base has not yet been generated or is considered outdated, full prediction must not be imitated by an artificial decision: the system records the absence of the required information basis. This behavior is important for reliability because it distinguishes an actual prediction generated using a trained rule base from a state in which the model does not have sufficient data for a justified assessment.

Real-time prediction is initiated primarily for processes that have a current resource request. The current signature is matched against the conditions of the rules in the base, after which rule activation levels are determined, their results are aggregated, and an estimate of assignment to the state classes is produced. The result includes not only a numerical risk value but also an unambiguously determined class: normal, potentially risky, pre-deadlock, or critical. Together with the class, the repository records the assignment degrees and identifiers of the activated rules. This provides interpretability of the decision: subsequent analysis can determine which features and which rules influenced the classification of a process as hazardous.

The final module converts the prediction into a control action. No intervention is performed for the normal

and potentially risky classes. If one or more processes are classified as pre-deadlock or critical, a set of candidates is formed and system processes are excluded from it. Among the remaining non-system candidates, the process with the shortest residence time in the system is selected, after which a forced-restart command is generated for it. Selecting this process limits the loss of already completed computation and breaks a potentially hazardous sequence of mutual resource waiting. After the command is executed, its result is recorded together with the predicted class and process data, allowing the resulting situation to be used subsequently for analysis and training-dataset updates.

For normal and potentially risky states, the system only records the prediction result. A control action is applied only for a pre-deadlock or critical state. System processes whose termination could disrupt OS operation are excluded from the set of related processes. Among the remaining candidates, the non-system process with the shortest residence time in the system is selected. A forced-restart command is generated for this process. This criterion does not guarantee the minimum recovery cost according to all possible measures; however, it minimizes the amount of already completed processing time lost among the available candidates and provides an unambiguous selection. The action result, process identifier, and determined state class are written to the log for subsequent training and analysis.

Thus, in practical implementation, the system does not replace OS mechanisms but operates as an analytical layer above them. The kernel remains the source of reliable information about processes and resources and the executor of the control command; the prediction system transforms this information into a predicted class and determines whether intervention is required. This organization simplifies transfer of the approach between environments because the platform-dependent part is mainly limited to obtaining system events and executing the command, while signature construction, assessment using the fuzzy rule base, and candidate-selection logic remain independent of the specific event source.

Reliability assessment and simulation modeling.

To verify operability, a discrete-event environment was created in Python using the SimPy library. Processes were specified by scenarios containing arrival times, durations of computation stages, sequences of resource requests, and resource-release delays. Resources were represented as limited-capacity objects, while the event engine separated simulation time from the actual execution time of the program. This organization made it possible to reproduce identical event sets for different methods and compare the results without the influence of random operating-system load.

The simulation experiments compared deadlock ignoring, forced resource preemption or rollback, no-preemption, the Banker's algorithm, a resource-allocation graph, wait-for-graph analysis, and the proposed prediction method. Each run involved 10,000 processes. Configurations with 30, 100, and 300 resources and 5% potentially hazardous processes were investigated, as well as configurations with 30 resources and 10% and 15% such processes. For the prediction method, 99.85–99.95% of processes completed in the five configurations; Recall was 99.00%, Specificity was 99.76–99.93%, Accuracy was 99.65–99.88%, Precision was 98.61–98.67%, and the F1-score was 98.80–98.84%. The false-positive rate among safe processes did not exceed 0.24%.

Increasing the number of resources from 30 to 300 while keeping the share of hazardous processes at 5% did not change the number of prevented or missed episodes: in each configuration, the prediction method prevented 99 of 100 hazardous groups. At the same time, the duration of a single prediction call increased because the dimensionality of the resource vector grew. This confirms the expected dependence of computational cost on the number of features, while also showing that scaling the resource description did not reduce reliability within the investigated range. With 30 resources fixed, increasing the share of hazardous processes to 15% led to a proportional increase in the number of preventive restarts; however, the advantage of the prediction method over the resource-allocation graph in terms of the average process residence time in the system remained at approximately 9%.

Organization of simulation experiment scenarios.

The set of methods for comparison was formed to cover different principles of deadlock handling. Ignoring was used as the baseline mode without special control; forced resource preemption or rollback represented reactive recovery; no-preemption and the Banker's algorithm represented preventive approaches based on restrictions on allowable actions; the resource-allocation graph and wait-for graph represented structural dependency-analysis methods; and the proposed method represented the predictive approach. All modes used the same arrival sequence of 10,000 processes and the same resource configuration within a given experiment. This allowed differences in process completion, number of interventions, and timing characteristics to be attributed specifically to the deadlock-handling method.

Each simulated process executed a specified sequence of computation segments and requests for single-instance resources. For a subset of processes, scenarios were formed in groups in which the resource-acquisition order created conditions for cyclic waiting. The share of hazardous processes determined how many such groups were included in the overall set. Increasing the number of resources from 30 to 100 and 300 while keeping 5% hazardous processes was used to assess the effect of the resource-description dimensionality. Separately, with 30 resources, the share of hazardous processes was increased to 10% and 15% to evaluate system behavior under a higher intensity of potential deadlocks. A fixed scenario and deterministic event order ensured reproducibility of the numerical results.

The obtained results demonstrated the difference between formal completion of all processes and the costs required to achieve such completion. Structural and preventive methods could achieve 100% completion; however,

some required prior restrictions or reacted only after a cycle had formed. For the prediction method, the main indicator was the ability to prevent a hazardous episode before an actual deadlock occurred. With 5% hazardous processes, the system prevented 99 of 100 generated groups, and increasing the number of resources to 100 and 300 did not change this result. As the share of hazardous processes increased, the number of preventive restarts grew together with the number of hazardous groups; however, Recall remained at 99%, and the false-positive rate did not exceed 0.24% among safe processes.

For quantitative evaluation, the prediction result was compared with the actual process state. Standard confusion-matrix elements were used: TP – hazardous processes correctly identified; TN – safe processes correctly identified; FP – false-positive results; FN – hazardous processes that were missed. The following metrics were calculated from these values:

$$\text{Accuracy} = (TP + TN) / (TP + TN + FP + FN)$$

$$\text{Recall} = TP / (TP + FN)$$

$$\text{Specificity} = TN / (TN + FP)$$

$$\text{Precision} = TP / (TP + FP)$$

$$F1 = 2 \cdot \text{Precision} \cdot \text{Recall} / (\text{Precision} + \text{Recall})$$

In addition to classification metrics, practical evaluation included the average processing time, throughput, time per analysis call, and prediction lead time relative to the moment at which the actual deadlock was formed.

Experimental validation on a real server platform.

The experimental platform was implemented on a physical HPE ML350 Gen10 server equipped with two Intel Xeon Silver 4214R processors, 128 GB DDR4 memory, an HDD/SSD storage subsystem, and Ubuntu Server 24.04 LTS. Parallel client requests were accepted by Apache HTTP Server, while the main resource interaction occurred among PostgreSQL 16.14 server processes competing for exclusive record locks. PostgreSQL created a separate process for each client connection; transactions locked shared records in different sequences, allowing cyclic dependencies to be generated in a controlled manner. Process and lock states were obtained from the `pg_stat_activity` and `pg_locks` system views and the `pg_blocking_pids()` function.

Two datasets of 10,000 processes each were formed. In the first, 500 processes (5%) were included in scenarios capable of forming a deadlock; in the second, 1,000 processes (10%) were included. Three modes were compared for each dataset: deadlock ignoring (IGN), reactive cycle detection using a resource-allocation graph (RAG), and the proposed prediction method (PM). Each mode was repeated ten times. In IGN mode, processes waited until a timeout occurred; in RAG mode, one process was terminated after a cycle had actually formed; in PM mode, signature analysis was performed before cycle formation, and in a pre-deadlock or critical state the non-system process with the shortest residence time in the system was restarted.

Table 1.

Average timing characteristics of repeated experiments

Share of hazardous processes	Mode	Completed	Avg. time, ms	Throughput, processes/s
5 %	IGN	9 500 ± 8	25.6 ± 0.38	390.6 ± 6.08
5 %	RAG	10 000	14.0 ± 0.18	714.3 ± 9.08
5 %	PM	9 990 ± 4	12.8 ± 0.18	781.1 ± 11.03
10 %	IGN	9 000 ± 15	32.4 ± 0.45	308.6 ± 4.46
10 %	RAG	10 000	17.1 ± 0.26	584.8 ± 8.42
10 %	PM	9 980 ± 6	15.6 ± 0.26	639.7 ± 10.85

For the dataset with 5% potentially deadlocking processes, the average processing time for PM was 12.8 ms, which was 8.6% lower than for RAG and 50.0% lower than for ignoring. Throughput was 781.1 processes/s, exceeding RAG by 9.4%. For the 10% dataset, the average time was 15.6 ms, which was 8.8% lower than RAG and 51.9% lower than IGN; throughput of 639.7 processes/s also exceeded RAG by 9.4%. The coefficients of variation of the timing characteristics in repeated experiments remained below 2%, indicating low measurement dispersion.

The average time required to analyze one state was $39.6 \pm 0.69 \mu\text{s}$ for RAG and $25.0 \pm 0.48 \mu\text{s}$ for the prediction method. In this experimental scenario, a single prediction call required 36.9% less time. This result is explained, in particular, by the fact that full clustering is not performed during real-time prediction; instead, the already generated rule base is used, and not all active processes are analyzed with the same intensity.

For the dataset with 5% potentially deadlocking processes, an average of 490 hazardous processes out of 500 were correctly identified; for the 10% dataset, 980 out of 1,000 were correctly identified. Recall was 98% in both cases. At the same time, Specificity exceeded 99.8%, while Precision was approximately 98.4–98.5%, meaning that the number of erroneous preventive interventions was substantially lower than the number of justified interventions. For the 5% dataset, every 98.0 ± 0.82 justified restarts corresponded to 8.0 ± 0.82 erroneous ones; for the 10% dataset, every 196.0 ± 0.82 justified restarts corresponded to 15.0 ± 0.82 erroneous ones.

A key characteristic of a prediction system is not only the correctness of the assigned class but also the time reserve before an actual deadlock. The average lead time was 3.10 ± 0.08 ms for 5% hazardous processes and 3.60 ± 0.08 ms for 10%. This interval was used to generate and execute the restart command before the system entered an actual deadlocked state. Thus, in the investigated scenarios, the system performed prediction rather than accelerated detection of an already formed cycle.

Table 2.

Average prediction reliability metrics over ten repetitions				
Metric	5 %: $\bar{x} \pm s$	95 % CI	10 %: $\bar{x} \pm s$	95 % CI
Accuracy, %	99.82 ± 0.03	99.80–99.84	99.65 ± 0.03	99.63–99.67
Recall, %	98.00 ± 0.82	97.42–98.58	98.00 ± 0.41	97.71–98.29
Specificity, %	99.92 ± 0.01	99.91–99.92	99.83 ± 0.01	99.83–99.84
Precision, %	98.39 ± 0.15	98.29–98.50	98.49 ± 0.07	98.44–98.55
F1-score, %	98.19 ± 0.34	97.95–98.44	98.25 ± 0.17	98.13–98.37
Missed detections, %	2.00 ± 0.82	1.42–2.58	2.00 ± 0.41	1.71–2.29
False positives, %	0.08 ± 0.01	0.08–0.09	0.17 ± 0.01	0.16–0.17
Lead time, ms	3.10 ± 0.08	3.04–3.16	3.60 ± 0.08	3.54–3.66

At the same time, the experiment revealed a fundamental limitation of the current implementation. Because a backup reactive mechanism was not applied after a missed prediction, FN directly reduced the number of successfully completed processes: on average, 9,990 out of 10,000 for the 5% dataset and 9,980 for the 10% dataset. Therefore, practical deployment should use a two-level scheme: the predictive mechanism should be primary, while traditional cycle detection should serve as a backup for the rare cases that were not classified in advance as pre-deadlock or critical.

Discussion of the experimental evaluation results.

The obtained classification metrics show that increasing the share of potentially hazardous processes from 5% to 10% had almost no effect on prediction sensitivity: Recall remained at 98% in repeated experiments. Precision increased slightly from 98.39% to 98.49%, while Specificity decreased from 99.92% to 99.83% because the absolute number of safe processes for which an erroneous preventive decision was generated increased. The ratio between useful and erroneous interventions is important. For the dataset with 10% potentially deadlocking processes, every 196.0 ± 0.82 justified restarts corresponded to 15.0 ± 0.82 erroneous ones; thus, erroneous interventions occurred approximately 13.1 times less frequently. This shows that the high Accuracy values are not due solely to the predominance of the safe class but are accompanied by high Precision specifically for hazardous processes.

The timing results complement the reliability assessment. Reactive analysis of a resource-allocation graph must maintain and check the dependency structure until a cycle appears, whereas the proposed method operates during real-time prediction with the current process signature and an already generated fuzzy rule base. In the investigated implementation, the average time for one analysis was 25.0 ± 0.48 μ s compared with 39.6 ± 0.69 μ s for RAG. At the level of processing the complete dataset, this reduced the average processing time by 8.6–8.8% and increased throughput by 9.4%. Thus, performing Fuzzy C-Means and rule generation within a separate training procedure not only separates the training and real-time prediction tasks but also directly reduces the computational load during decision-making.

The practical significance of the 3.1–3.6 ms lead time lies in the presence of an interval between identifying a hazardous tendency and the actual formation of a cyclic dependency. Within the experiment, this time was sufficient to select a candidate, generate a command, and forcibly restart the process. The control action was intentionally limited to a single type of intervention: restarting the non-system process with the shortest residence time in the system. This limitation simplifies interpretation of the experiment because the prediction effect is not mixed with the results of different strategies such as delaying a request, changing priority, or forcibly preempting a resource. At the same time, the minimum-residence-time criterion reduces the amount of already completed work lost when a potential mutual-waiting sequence is broken.

Nevertheless, the obtained results should be interpreted within the investigated conditions. Real-system validation was performed for a server scenario with PostgreSQL transactional locks and two levels of hazardous-process share; therefore, transferring the numerical lead-time values to other operating systems, resource types, or workloads requires separate verification. In addition, in the current experimental implementation, no backup reactive mechanism was applied after a missed prediction, so FN directly reduced the number of successfully completed processes. For practical operation, a combined scheme is appropriate: prediction is used as the primary means of early intervention, while traditional cycle detection remains a backup level. An additional direction is periodic updating of the rule base using new historical signatures, which will make it possible to account for changes in the nature of the resource load.

CONCLUSIONS AND PROSPECTS FOR FURTHER RESEARCH

The article develops an architecture for a system for predicting process deadlock states in which the complete sequence from a system event to a control decision is implemented through monitoring, signature construction, data storage, fuzzy-rule-base generation, prediction, and preventive-control modules. Separating the training procedure from real-time prediction makes it possible to perform Fuzzy C-Means and rule generation outside the critical resource-management cycle and to use the prepared rule base during real-time operation for processes waiting for a resource.

For pre-deadlock and critical states, the control action is limited to forcibly restarting the non-system process with the shortest residence time in the system. Simulation modeling confirmed that high prediction reliability is maintained when the number of resources and the share of potentially hazardous processes are varied. In the real server system, ten repeated runs yielded Accuracy of 99.65–99.82%, Precision of 98.39–98.49%, and an F1-score of 98.19–98.25%. The prediction was generated on average 3.1–3.6 ms before the actual deadlock. Compared with reactive resource-allocation-graph analysis, the average processing time decreased by 8.6–8.8%, while throughput increased by 9.4%.

References

1. Klots, Yu. P., & Mostovyi, S. V. (2015). Prediction of process deadlocks in computer systems. *Electrical and Computer Systems*, 19(95), 205–209.
2. Nicheporuk, A., Klots, Y., Yashyna, O., Mostovyi, S., & Nicheporuk, Y. (2019). Prediction of entering processes into the deadlock state. *Indonesian Journal of Electrical Engineering and Computer Science*, 14(3), 1484–1492. <https://doi.org/10.11591/ijeecs.v14.i3.pp1484-1492>
3. Mostovyi, S. (2021). Analysis of methods and models for solving the process deadlock problem in computer systems. *Herald of Khmelnytskyi National University. Technical Sciences*, 5, 27–31. <https://doi.org/10.31891/2307-5732-2021-301-5-27-31>
4. Mostovyi, S. V. (2023). Architecture of a process deadlock prediction system in a computer system. In *APKN-2023: Proceedings of the XV All-Ukrainian Scientific and Practical Conference* (pp. 343–345). Khmelnytskyi: Khmelnytskyi National University.
5. Mostovyi, S. V. (2024). Model and method for constructing a process signature for deadlock prediction in a computer system. In *APKN-2024: Proceedings of the XVI All-Ukrainian Scientific and Practical Conference* (pp. 580–583). Khmelnytskyi: Khmelnytskyi National University.
6. Mostovyi, S., & Pyrch, O. (2025). Prediction of the process deadlock state in computer systems. *Measuring and Computing Devices in Technological Processes*, 4, 284–292. <https://doi.org/10.31891/2219-9365-2025-84-32>
7. Talaat, M., Saleh, A., Moawad, M., & Zaki, J. (2023). Fog computing effective load balancing and strategy for deadlock prediction management. *Ain Shams Engineering Journal*, 14(12), 102561. <https://doi.org/10.1016/j.asej.2023.102561>
8. Zohora, M. F., Farhin, F., & Kaiser, M. S. (2024). DBDAA: A real-time approach to Dynamic Banker's Deadlock Avoidance Algorithm with optimized time complexity. *PLOS ONE*, 19(9), e0310807. <https://doi.org/10.1371/journal.pone.0310807>
9. Ye, C., Cai, Y., & Zhang, C. (2024). When threads meet interrupts: Effective static detection of interrupt-based deadlocks in Linux. In *33rd USENIX Security Symposium* (pp. 6167–6184).
10. Ji, W., Bo, L., Yuan, Y., & Sun, X. (2024). TDFix: A lightweight tool for fixing deadlocks based on templates. *Science of Computer Programming*, 233, 103073. <https://doi.org/10.1016/j.scico.2023.103073>
11. Rowicki, R. J., Francalanza, A., & Scalas, A. (2025). Correct black-box monitors for distributed deadlock detection: Formalisation and implementation. *Proceedings of the ACM on Programming Languages*, 9(OOPSLA2), 527–554. <https://doi.org/10.1145/3763069>
12. Lu, F., Wang, X., Yuan, G., Zheng, J., & Bao, Y. (2026). Deadlock detection and replay of multi-thread programs based on refined segmentation and lock graphs. *Tsinghua Science and Technology*. Online First. <https://doi.org/10.26599/TST.2025.9010085>
13. Hassan, M. H., Darwish, S. M., & Elkaffas, S. M. (2022). An efficient deadlock handling model based on neutrosophic logic: Case study on real-time healthcare database systems. *IEEE Access*, 10, 76607–76621. <https://doi.org/10.1109/ACCESS.2022.3192414>
14. Khang, T. D., Vuong, N. D., Tran, M.-K., & Fowler, M. (2020). Fuzzy C-Means clustering algorithm with multiple fuzzification coefficients. *Algorithms*, 13(7), 158. <https://doi.org/10.3390/a13070158>