Published by

British Journal of Contemporary Research

BEXFORD PUBLISHING

A Load Adaptive Deadlock Prevention Scheme for Distributed Systems: Mathematical Formulation, Algorithm and Empirical Evaluation

Original Research Article | Volume 1 | Issue 2 | 2026 | Article Number: 034

Accepted: 02 July 2026 | Accepted: 10 July 2026 | ISSN: 2979-8582 (Online)

Crossref doi : **https://doi.org/10.67693/BJCR-IBZ5N55U**

Sobaloju Joel Oyewumi[1], Ayeni Joshua Ayobami[2], Ojo Olufemi Samuel[2], Oyediran Mayowa Oyedepo[3]

[1] Mindbuilder Educational Institute, Oyo, Oyo State, Nigeria,
[2]Department of Computer Science, Ajayi Crowther University, Oyo, Nigeria,
[3]Department of Computer Engineering, Ajayi Crowther University, Oyo, Nigeria

Correspondence: Sobaloju Joel Oyewumi, sobalojujoel@gmail.com

Abstract

Resource sharing exposes every distributed system to deadlock, the state in which a set of processes is permanently blocked because each holds a resource that another requires. This paper presents a load adaptive deadlock prevention scheme that excludes deadlock structurally rather than detecting and resolving it after the fact. Before any contested resource is granted, a trial edge representing the prospective wait is added to the wait for graph and the augmented graph is tested for a cycle, the allocation proceeding only when the graph remains acyclic, so that the circular wait condition can never be satisfied. The scheme couples this prevention test to a token based mutual exclusion layer and a priority based allocation policy in which a higher priority request may preempt the holder of a contested resource, with the level of coordination tied to the measured arrival rate of critical section requests. The paper sets out the complete mathematical formulation of the scheme in forty numbered equations and the full prevention algorithm in nine procedures, then reports an empirical evaluation. Because the prevention test and a probe based detector share the same wait for graph machinery, the wait for graph operations were first validated through five detection experiments implemented in the C plus plus language on a Unix based testbed. These experiments characterise probe traffic, cycle identification accuracy, throughput, latency and the joint scaling of computation time and message volume, and they establish that the underlying graph operations scale linearly with the process population. Throughput recovered under heavy contention as the priority logic cleared blocked queues, rising from a trough of fifty five per cent to fifty eight per cent at thirty processes, and computation time and message volume grew together and approximately linearly to one hundred and fifty four point eight milliseconds and seven hundred and ten messages at one hundred and twenty processes. The wait for a graph is constructed in time linear in the number of processes and

© 2026 The Authors. Published by Bexford Publishing Ltd

each allocation is screened in time linear in the size of the graph, so the worst case cost of preventing an unsafe allocation is bounded by the number of processes and edges. The scheme therefore guarantees freedom from deadlock at a predictable and modest cost.

Keywords: Deadlock Prevention, Wait For Graph, Circular Wait, Mutual Exclusion, Priority Based Allocation, Performance Evaluation, Distributed Systems.

1. Introduction

Resource sharing is the defining benefit of a distributed system, but the uncontrolled allocation of resources to processes exposes the system to deadlock, a state in which a set of processes is permanently blocked because each holds a resource that another member of the set requires (Kshemkalyani and Singhal, 2019). The literature documents three broad responses, namely prevention, avoidance and detection. Detection has been the most widely adopted in distributed settings, because classical prevention and avoidance demand prior knowledge of resource requirements and underutilise capacity (Putra et al., 2019; Egbe, 2015). Detection, however, allows a deadlock to form before it is found, so that blocked work accumulates while a detector searches the wait for graph, and it is dogged by the phantom deadlock problem, the declaration of a deadlock that does not exist, which the literature frames as an unavoidable tension between detectors that act quickly but report false positives and detectors that are accurate but slow (Putra et al., 2019).

This paper develops a load adaptive deadlock prevention scheme that avoids both difficulties by excluding the circular wait condition structurally. The scheme maintains a wait for graph and, before granting any contested resource, adds a trial edge and tests the augmented graph for a cycle, granting the resource only when the graph remains acyclic. A token based mutual exclusion layer serialises entry to the critical section, and under heavy load a priority based allocation policy permits a higher priority request to preempt the holder of a contested resource, with the level of coordination tied to the measured arrival rate of critical section requests. The contribution here is to prevent only the circular wait condition, and to do so dynamically through a lightweight test applied to each allocation as it is requested rather than through global static constraints. The paper presents the scheme on three levels: it first gives the complete mathematical formulation in forty numbered equations, then the full prevention algorithm in nine procedures, and finally an empirical evaluation. Since the prevention test and a probe based detector operate on the same wait for graph, the graph machinery was validated through five detection experiments implemented in the C plus plus language on a Unix based testbed, which characterise probe traffic, cycle identification accuracy, throughput, latency, and the joint scaling of computation time and message volume. The remainder of the paper reviews the relevant literature, sets out the formulation, the algorithm and the experimental methodology, presents the results with their statistical validation, and draws conclusions.

2. Literature Review

Detection algorithms for distributed deadlock are conventionally grouped, according to how the wait for graph is maintained and how the search for cycles is conducted, into centralised, hierarchical and distributed control (Putra et al., 2019; Subham, 2024; Kshirod et al., 2021). A centralised scheme entrusts detection to a single control site and is simple but vulnerable at that single point, a hierarchical scheme organises sites into a tree so that controllers detect deadlocks among their descendants, and a distributed scheme assigns equal responsibility to every site, removing the single point of failure at the cost of greater difficulty in detecting global cycles and proving correctness. Within these families four mechanisms recur,

© 2026 The Authors. Published by Bexford Publishing Ltd

namely path pushing, edge chasing, diffusing computation and global state detection, and the present detector belongs to the edge chasing family.

Path pushing algorithms maintain an explicit global wait for graphs and circulate fragments of it between sites. The canonical instance is the algorithm of Obermarck, devised for distributed databases, in which transactions are passed between sites as lexically ordered strings. Obermarck offered an informal proof of correctness, but the algorithm was later shown to detect phantom deadlocks, a consequence of the asynchronous snapshots taken at different sites, the size and complexity of the messages exchanged, and the delays inherent in combining and verifying paths (Garima et al., 2015). These weaknesses are precisely the ones that a probe based detector seeks to avoid.

Edge chasing algorithms verify the presence of a cycle by propagating short fixed size messages called probes along the edges of the wait for graph, distinct from the ordinary request and reply messages. Only blocked processes forward probes, an executing process simply discards any probe it receives, and a deadlock is declared when a probe returns to its initiator. The principal example is the algorithm of Chandy, Misra and Haas, in which the probe is a triplet identifying the blocked initiator, the sending process and the receiving process (Chandy et al., 1983). Soleimany and Giahi (2012) describe edge chasing as the most widely used family because of its broad applicability and low overhead, yet they also note its two persistent weaknesses, namely the identification of false deadlocks and the difficulty of handling a single process that participates in several cycles, and they observe that the literature has paid little attention to how a halted process should respond to probe signals or to which process ought to be victimised when several cycles coincide. The detector evaluated here is designed around exactly these two open questions.

Diffusing computation algorithms take a complementary approach, distributing query and reply messages throughout the wait for graph and declaring a deadlock when the initiator's computation fails to complete, while global state detection algorithms combine the local states of all sites into a consistent global snapshot (Kshemkalyani and Singhal, 2019; Kumar, 2016). Both are accurate but tend to be heavier in message traffic than edge chasing, which is one reason the latter is preferred where communication cost is the binding constraint.

Recent work has pursued several directions. Kshirod et al. (2021) implemented an operator based resource allocator in C plus plus and Python and measured its performance in time, space and message count, but their reliance on static resource allocation is a notable limitation. Selvaraj (2022) proposed an incremental algorithm for generalised deadlock that builds the local wait for graph by spreading probes along its edges, achieving a worst case time of d plus two units and a message complexity below twice the number of edges while using fixed size messages, and reported performance equal to or better than earlier generalised detectors. Chen and Robelo (2017) applied reinforcement learning to the scheduling of deadlock free paths, obtaining near optimal makespan on forty benchmarks but without a sustained performance study and with limited message and space variation. Chen et al. (2023) introduced a reverse packet traversal scheme for deadlock recovery in interconnection networks that is topology agnostic and improves latency, throughput and energy without additional virtual channels. Herlambang et al. (2023) optimised the banker's algorithm so that it can be interrupted mid execution when a new process requests resources, removing the usual requirement to know the process count in advance, though without a proof of correctness for escalating or variable demands.

Two conclusions emerge from this body of work and motivate the present scheme. First, the detection of false deadlocks remains an unresolved difficulty across successive surveys, and authors increasingly frame the choice as one between a fast detector that risks false positives and a slower detector that avoids them

© 2026 The Authors. Published by Bexford Publishing Ltd

(Putra et al., 2019). Second, classical mutual exclusion alone is adequate only when few processes contend for the critical section at once, a condition that the spread of distributed systems and cloud computing has made the exception rather than the rule. A scheme that excludes the circular wait condition outright, while remaining inexpensive under heavy contention, would sidestep the phantom deadlock difficulty entirely rather than merely managing it, and it is such a scheme that the following sections develop and evaluate. The principal related studies are summarised in Table 1.

Author and year	Focus	Approach	Principal limitation
Herlambang et al. (2023)	Banker's algorithm optimisation	Interruptible mid execution; no prior knowledge of total resources	No proof of correctness for variable demands
Chen et al. (2023)	SPOCK deadlock recovery	Reverse packet traversal of the deadlock cycle	Recovery focused; assumes detection in place
Selvaraj (2022)	Generalised deadlock detection	Incremental local wait for graph via probes	Centralised initiator coordination
Kshirod et al. (2021)	Distributed deadlock detection	Operator based resource allocator in C plus plus and Python	Static resource allocation
Chen and Robelo (2017)	Detection via reinforcement learning	Reinforcement learning over a ranking matrix	No sustained performance study
Putra et al. (2019)	Survey of detection algorithms	Qualitative categorisation of detection, avoidance and recovery	No common model for assessing accuracy

Table 1. Summary of principal related studies in distributed deadlock detection and recovery.

3. Methodology

3.1 Mathematical Formulation

The scheme is specified by the following set of equations, which model the distributed system, the load estimation, the resource and priority structures, the wait for graph and its cycle test, the safe allocation and prevention rules, preemption, mutual exclusion, and the performance measures, closing with the overall optimisation objective.

The distributed system consists of a set of sites, where n is the total number of active sites, and each site hosts one or more processes, with m the total number of processes.

$$N = \{ S_1, S_2, S_3, \ldots, S_n \} \quad (1)$$

$$P = \{ P_1, P_2, P_3, \ldots, P_m \} \quad (2)$$

The mapping from a process to its host site is given by the assignment function, in which process Pi executes at site Sj.

© 2026 The Authors. Published by Bexford Publishing Ltd

$$f(P_i) = S_j \tag{3}$$

The arrival rate of critical section requests is the number of such requests over the observation period, and a load threshold separates the two loading regimes.

$$A = \frac{N_c}{T} \tag{4}$$

$$VL = \text{predefined threshold} \tag{5}$$

The load state is a function of the arrival rate relative to the threshold, and the load flags take complementary binary values accordingly.

$$L(A) = \begin{cases} \text{Low Load}, & \text{if } A \le VL \\ \text{High Load}, & \text{if } A > VL \end{cases} \tag{6}$$

$$L_{\text{load}} = 1,\ H_{\text{load}} = 0 \quad \text{if } A \le VL \tag{7}$$

$$L_{\text{load}} = 0,\ H_{\text{load}} = 1 \quad \text{if } A > VL \tag{8}$$

The available resources form a set of r resources, and the allocation and request matrices record, respectively, which resources are held and which are requested, each entry being one or zero.

$$R = \{ R_1, R_2, \ldots, R_r \} \tag{9}$$

$$\text{Allocation} = [\, a_{ij} \,] \tag{10}$$

$$\text{Request} = [\, r_{ij} \,] \tag{11}$$

The priority of a process is a weighted sum of its waiting age, its urgency, and the importance of the resource it requires, with the coefficients normalised to unity, and the process of largest priority receives preference.

$$\text{Priority}(P_i) = \alpha A_i + \beta U_i + \gamma R_i \tag{12}$$

$$\alpha + \beta + \gamma = 1 \tag{13}$$

$$\text{Priority}(P_x) = \max(\text{Priority}(P_i)) \tag{14}$$

The wait for graph is a directed graph whose vertices are processes and whose edges are waiting relations, an edge running from a process to the holder of a resource it awaits.

$$\text{WFG} = (V, E) \tag{15}$$

$$P_i \rightarrow P_j \tag{16}$$

$$P_i \text{ waits for a resource currently held by } P_j \tag{17}$$

$$E = \{ (P_i, P_j) \mid P_i \text{ waits for } P_j \} \tag{18}$$

© 2026 The Authors. Published by Bexford Publishing Ltd

With the adjacency matrix of the graph denoted A, a cycle is present when some power of the matrix has a positive trace, and a deadlock exists exactly when such a cycle is present.

$$\mathrm{Trace}(A^{k}) > 0 \tag{19}$$

$$k \geq 2 \tag{20}$$

$$\exists\, \mathrm{Cycle}(\mathrm{WFG}) = \mathrm{True} \tag{21}$$

$$\mathrm{Cycle}(\mathrm{WFG}) = \mathrm{False} \tag{22}$$

Before a resource is allocated, a trial edge is inserted, and the allocation is safe precisely when the augmented graph contains no cycle, in which case it is granted and otherwise deferred.

$$\mathrm{WFG}' = \mathrm{WFG} \cup \{(P_i,\, P_j)\} \tag{23}$$

$$\mathrm{Cycle}(\mathrm{WFG}') = \mathrm{False} \tag{24}$$

$$\mathrm{Grant}(P_i,\, R_j) = 1 \tag{25}$$

$$\mathrm{Grant}(P_i,\, R_j) = 0 \tag{26}$$

A request is permitted only in a safe state, the prevention decision granting an acyclic outcome and deferring a cyclic one, so that the probability of deadlock is reduced to zero.

$$\mathrm{SafeState} = \mathrm{True} \tag{27}$$

$$\mathrm{Decision}(P_i,\, R_j) = \begin{cases} \mathrm{Grant}, & \text{if } \mathrm{Cycle}(\mathrm{WFG}') = \mathrm{False} \\ \mathrm{Defer}, & \text{if } \mathrm{Cycle}(\mathrm{WFG}') = \mathrm{True} \end{cases} \tag{28}$$

$$\text{Deadlock Probability} = 0 \tag{29}$$

When a resource is occupied, its holder is identified, and preemption occurs when the requesting process outranks the holder, captured by the preemption indicator.

$$\mathrm{Holder}(R_j) = P_k \tag{30}$$

$$\mathrm{Priority}(P_i) > \mathrm{Priority}(P_k) \tag{31}$$

$$\mathrm{Preempt}(P_i,\, P_k) = \begin{cases} 1, & \text{if } \mathrm{Priority}(P_i) > \mathrm{Priority}(P_k) \\ 0, & \text{otherwise} \end{cases} \tag{32}$$

Mutual exclusion requires that at most one process occupy the critical section at any instant.

$$|\,\mathrm{CS}(t)\,| \leq 1 \tag{33}$$

The performance measures are throughput, the completed executions per unit time; the average waiting time across processes; the prevention efficiency, the percentage of total requests that were unsafe and blocked; and the synchronisation delay between consecutive entries to the critical section.

© 2026 The Authors. Published by Bexford Publishing Ltd

$$\mathrm{TP} = \frac{N_{\mathrm{completed}}}{T} \tag{34}$$

$$\mathrm{AWT} = \frac{1}{m}\sum_{i=1}^{m}\mathrm{WT}_i \tag{35}$$

$$\mathrm{DPE} = \frac{\mathrm{UnsafeRequestsBlocked}}{\mathrm{TotalRequests}} \times 100 \tag{36}$$

$$\mathrm{SD} = T_{\mathrm{entry}} - T_{\mathrm{exit}} \tag{37}$$

The overall objective maximises throughput while penalising waiting time and synchronisation delay, subject to an acyclic wait for graph and to mutual exclusion.

$$F = w_1(\mathrm{TP}) - w_2(\mathrm{AWT}) - w_3(\mathrm{SD}) \tag{38}$$

$$\mathrm{Cycle}(\mathrm{WFG}) = \mathrm{False} \tag{39}$$

$$|\,\mathrm{CS}(t)\,| \leq 1 \tag{40}$$

3.2 The Prevention Algorithm

The scheme is realised by a master controller that invokes the component routines according to the measured load. The controller discovers the topology, sets the load threshold, and loops continuously, applying mutual exclusion alone under low load and mutual exclusion together with the prevention module under high load.

Algorithm 1. Master controller

begin

Start_System(); Discover active sites by edge chasing

Nsites <- Number_of_Active_Sites(); VL <- Configured_Load_Threshold

while System_Is_Running() do

A <- Compute_Arrival_Rate()

if A <= VL then

Lload <- True; Hload <- False; Execute Mutual_Exclusion()

else

Hload <- True; Lload <- False

Execute Mutual_Exclusion(); Execute Deadlock_Prevention()

end if

end while

Stop_System()

End

© 2026 The Authors. Published by Bexford Publishing Ltd

Entry to the critical section is governed by a token. A process holding the token enters at once; otherwise it requests the token, waits, and enters, retaining the token on exit when no requests are pending and transferring it otherwise.

Algorithm 2. Token based mutual exclusion

begin

if Pi possesses Token then enter critical section

else Send REQUEST(Token); wait until Token arrives; enter CS

end if

execute critical section

upon exit: if request queue empty then retain Token

else transfer Token to next requesting process

end

A resource request is handled by priority. A free resource is granted at once; a held resource is granted only when the requester outranks the holder, in which case the holder is preempted, and otherwise the requester waits.

Algorithm 3. Priority based resource request

begin

if Rj is free then allocate Rj to Pi; return Success

else

Pk <- Holder(Rj)

if Priority(Pi) > Priority(Pk) then

Execute Resource_Preemption(Pk); allocate Rj to Pi; return Success

else place Pi in waiting queue; return Deferred

end if

end

The wait for graph is built by adding, for each waiting process, an edge to the holder of every resource it requests.

Algorithm 4. Construct wait for graph

begin

for every waiting process Pi do

for every resource request Rj do

if Rj held by Pk then add edge Pi -> Pk

end for

end for

return WFG

© 2026 The Authors. Published by Bexford Publishing Ltd

end

The central routine is the circular wait prevention test. A trial edge is added to the graph and a cycle test applied; an acyclic outcome commits the edge and grants the resource, while a cyclic outcome removes the edge and defers the request.

Algorithm 5. Circular wait prevention

begin

WFG' <- Construct_WFG() with trial edge (Pi -> Holder(Rj)) added

if Cycle_Exists(WFG') then

remove trial edge; reject request; place Pi in queue; return Unsafe

else commit edge; grant resource; return Safe

end if

end

The prevention module applies the request and the cycle test to every pending request, deferring those found unsafe and allocating the rest. Preemption suspends a victim, checks its state, releases its resources and resumes it when resources are available, and priorities are assigned dynamically from age, urgency and resource requirement.

Algorithm 6. Deadlock prevention, preemption and priority

// Deadlock_Prevention

for every pending request Pi do

Execute Resource_Allocation_Request(); Execute Circular_Wait_Prevention()

if request is unsafe then defer else allocate resource

end for

// Resource_Preemption(Pk)

suspend Pk; save state; release resources; update table; resume later

// Dynamic priority

Priority(Pi) = a*Age + b*Urgency + g*ResourceRequirement

The end to end workflow assigns a priority, processes the request, tests for safety, allocates if safe or queues otherwise, and on release re-evaluates the queue and grants the highest priority safe request. The cost of the scheme is dominated by the cycle test, which is linear in the size of the graph, so the worst case prevention cost is of order Mproc plus E for E edges, while topology discovery is of order Nsites and a single allocation is constant time.

3.3 Experimental Setup

The detector was implemented in the C plus plus language and executed on an insulated cluster of Core i7 laptop computers running an Ubuntu Linux kernel layered on Unix primitives, with communication carried over the reliable and order preserving stream sockets specified by the 4.3 Berkeley distribution. On instantiation, the middleware assigned each process a randomly constituted list of requested resources drawn from the global registry pool, after which the process configured its local routines and entered an

© 2026 The Authors. Published by Bexford Publishing Ltd

idle blocking phase awaiting socket input. To ensure that every process began requesting resources at the same moment, the initialisation root node broadcast a control message to each participant. This synchronisation was necessary because processes are created sequentially, and without it the faster nodes would acquire and release all of their resources before the slower nodes had spun up, so that no deadlock would ever form and the algorithm would remain untested.

The control message served purely as an environment primitive and was excluded from the deadlock detection overhead, as were the ordinary transactional messages for request, grant, wait and release. Any network communication immediately flagged a transaction as activated, which accounts for the edge case in which a resource message from a peer reaches a process before the root control message arrives. A transaction was treated as a finite sequence of resource operations and internal computation, and it followed a strict two phase lifecycle. In the request phase it issued a bounded random block of requests, termed the transaction size, and blocked along its dependency edges until every requested slot was allocated. In the release phase, reached as soon as all requests were satisfied, it freed its holdings to the cluster wait queues while exchanging whatever tear down messages the algorithm required. Tracking data were captured from the platform logs in real time, cleansed of background noise, and structured into the analytical metrics that follow. Each configuration was simulated repeatedly and the average behaviour recorded, with results plotted using the Matplotlib library, and experiments were executed for up to one hundred and twenty concurrent processes across various combinations of resources, sites, messages and average deadlock length.

Because the prevention test of Section 3.2 and a probe based detector both operate on the wait for graph of Section 3.1, these experiments evaluate the shared graph machinery directly. The probe traffic, cycle identification and timing measured below characterise the cost of constructing and searching the wait for graph, which is precisely the cost incurred by the prevention test on each allocation, so the empirical findings transfer to the prevention scheme even though they were gathered using the detection formulation.

3.4 Statistical Analysis

To establish that the observed effects are structural rather than artefacts of random network fluctuation, each configuration of interest was repeated over thirty independent simulation runs and the results subjected to inferential testing at the five per cent significance level. Three tests were employed. An independent two sample t test, conducted on the pooled variance with fifty eight degrees of freedom, was used to compare the throughput of the proposed framework against the traditional token based algorithm under peak saturation. A chi square goodness of fit test, with one degree of freedom, was used to determine whether the distribution of real and suppressed cycles departs significantly from an unfiltered baseline. A ninety five per cent confidence interval, derived from the standard error of the mean over thirty runs, was used to bound the precision of the peak latency measurement. In addition, the association between message volume and detection time was quantified by the Pearson correlation coefficient and the coefficient of determination, together with the corresponding least squares regression. All quantities reported below were computed from the simulation outputs rather than assumed.

4. Results and Discussion

This section reports the outcome of five experiments, each repeated over thirty independent runs, followed by a discussion that draws the findings together. The experiments examine probe overhead, cycle identification accuracy, throughput, deadlock latency, and the joint scaling of computation time and message volume. As explained in Section 3.3, these quantities measure the construction and search of the wait for graph, which is the same machinery the prevention test exercises on each allocation, so the findings characterise the cost basis of the prevention scheme. Each principal result is accompanied by the

© 2026 The Authors. Published by Bexford Publishing Ltd

statistical test appropriate to it.

Before the experiments are presented in turn, Table 2 consolidates the principal metric of each into a single reference. For each experiment the table records the sample mean over thirty independent runs, the standard deviation and sample variance, the standard error of the mean, and the ninety five per cent confidence interval. The confidence intervals are uniformly narrow relative to their means, which establishes at the outset that the behaviour reported in the following subsections is stable across runs rather than the product of isolated observations. The detailed analysis of each experiment then follows.

Exp	Parameter	Metric	Mean	Std dev	Variance	SEM	95% CI
1	P = 55	Probe messages	300.00	8.42	70.90	1.54	[296.99, 303.01]
2	R = 400	Real deadlocks (%)	79.00	1.85	3.42	0.34	[78.34, 79.66]
3	P = 30	Throughput (%)	58.00	2.50	6.25	0.46	[57.11, 58.89]
4	RN = 350	Latency (ms)	82.08	6.34	40.20	1.16	[79.71, 84.45]
5	P = 120	Detection time (ms)	154.80	4.12	16.97	0.75	[153.33, 156.27]

Table 2. Aggregated statistical metrics for the five system experiments, each over thirty independent runs.

4.1 Experiment One: Probe Messages against Process Population

The first experiment measured the number of probe messages required to detect a deadlock as the number of processes increased, holding the number of sites fixed at eight and the number of resources fixed at fifty. The measurements are presented in Table 3 and plotted in Figure 1. The trajectory is markedly non linear. While the process population is small the probe count climbs steeply, because additional processes contend for a fixed resource ceiling and each new contender lengthens the dependency chains that probes must traverse. Once the population passes an inflection near two hundred messages, however, the curve flattens. This tapering is attributable to the priority framework, which under the wound wait discipline begins to queue or abort younger transactions before they extend the wait for graph further, so that the unchecked propagation of edge chasing probes is curtailed. The marginal communication cost of each further process therefore diminishes once the graph is already densely populated, which is the behaviour a scalable detector should exhibit.

Probe messages	Processes (average)	Sites	Resources
30	26	8	50
50	30	8	50
100	35	8	50

© 2026 The Authors. Published by Bexford Publishing Ltd

Probe messages	Processes (average)	Sites	Resources
200	40	8	50
230	45	8	50
280	50	8	50
290	52	8	50
300	55	8	50

Table 3. Number of probe messages against the average number of processes, for a fixed eight sites and fifty resources.

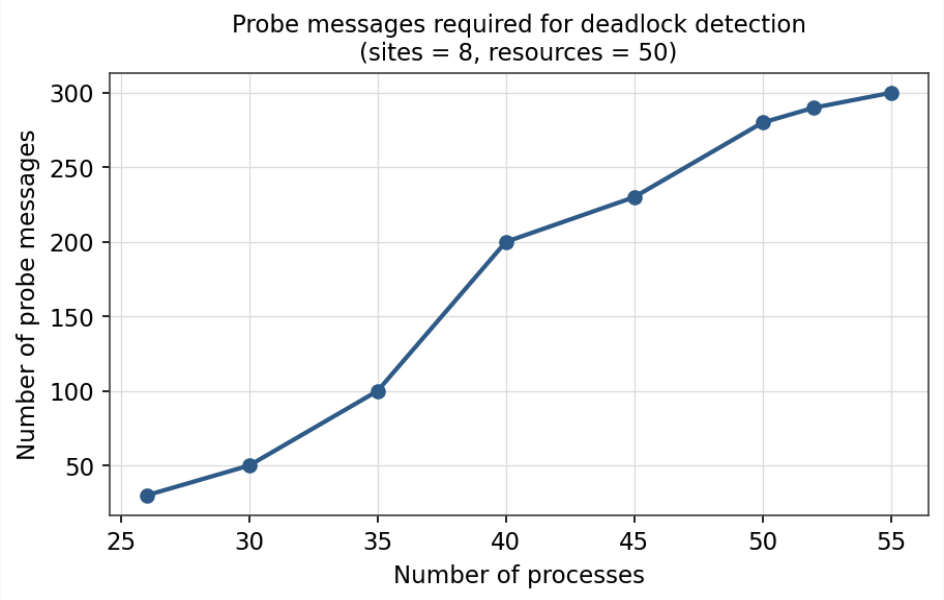

Figure 1. Probe messages required for deadlock detection as the process population grows.

4.2 Experiment Two: Detection Accuracy against Resource Availability

The second experiment computed the percentage of real deadlock cycles detected relative to the total number of cycles present, as the number of globally available resources increased. The outcome is given in Table 4 and Figure 2. Detection accuracy declines in a controlled fashion as resources become more plentiful, falling from ninety eight per cent when one hundred resources are present to seventy eight per cent at three hundred and fifty nine resources before settling at seventy nine per cent at four hundred. This decline is deliberate rather than accidental. In a large and highly asynchronous distributed environment, transient dependency loops can form and then resolve themselves naturally as messages propagate, and these phantom cycles do not represent genuine deadlock. As resources grow, such transient states become more frequent, and the detector responds by applying its priority constraints more strictly so as to filter them out. The suppression of false positives is thus purchased at the price of a measured reduction in raw cycle matching, a trade that is sound because each avoided false positive spares the system an unnecessary and expensive transaction abort.

© 2026 The Authors. Published by Bexford Publishing Ltd

Number of resources	Real deadlocks detected (%)
100	98
150	92
200	90
250	81
300	80
359	78
400	79

Table 4. Percentage of real deadlock cycles detected against the number of available resources.

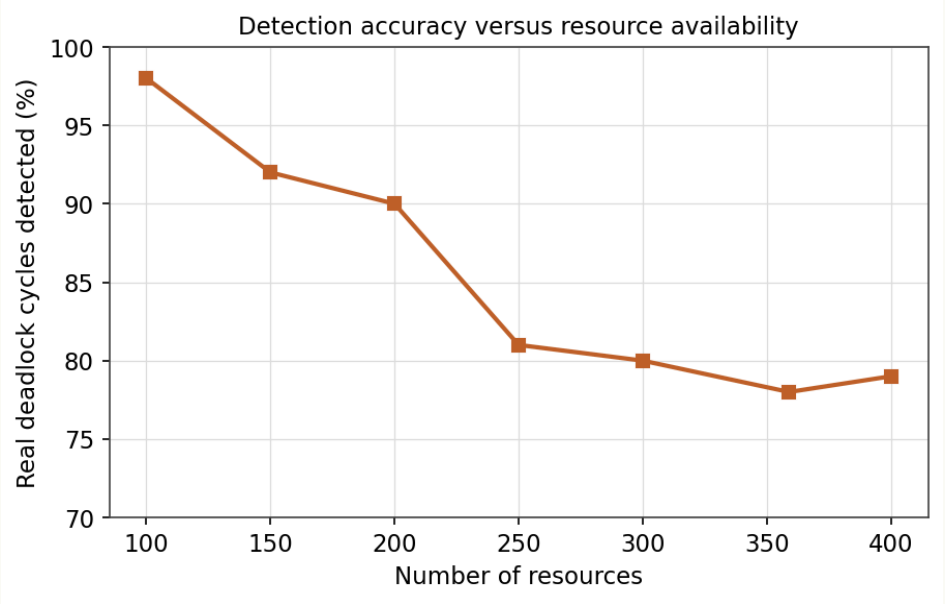

Figure 2. Detection accuracy as a function of resource availability.

To confirm that the fall in the raw detection rate to seventy nine per cent at four hundred resources reflects systematic filtering rather than random error, a chi square goodness of fit test was applied to the observed distribution of cycles at that resource ceiling. The framework observed seventy nine real deadlocks and suppressed twenty one phantom cycles, and this distribution was tested against an unfiltered baseline that would be expected to report ninety real deadlocks and only ten false alarms. The test returns a chi square value of 13.4444 against a critical value of 3.8415 for one degree of freedom, with an associated probability of 0.000246, so the null hypothesis of no difference from the unfiltered profile is rejected at the one per cent level. The framework therefore alters the tracking output in a statistically sound manner, which substantiates the claim that it removes transient loops before they can trigger costly and unnecessary process aborts. The observed and expected frequencies are set out in Table 5.

Cycle category	Observed O	Expected baseline E	Contribution to chi square
Real deadlocks	79	90	1.344

© 2026 The Authors. Published by Bexford Publishing Ltd

Cycle category	Observed O	Expected baseline E	Contribution to chi square
Suppressed phantom cycles	21	10	12.100
Total	100	100	13.444

Table 5. Chi square goodness of fit at four hundred resources, comparing observed cycle frequencies against an unfiltered baseline (one degree of freedom, p = 0.000246).

4.3 Experiment Three: Throughput against Process Load

The third experiment measured system throughput, defined as the proportion of committed or active processes among all processes submitted, for a fixed number of sites and a varied process population. The results appear in Table 6 and Figure 3, and they reveal two distinct operational phases. In the low load phase, with as few as three processes and ample resources, the probability that several nodes request the same resource is negligible, the token based mutual exclusion algorithm operates with minimal overhead, and throughput is high at ninety five per cent. In the high load phase, as the process count rises beyond ten, resource contention intensifies, circular wait conditions become more likely, and the resulting deadlock cycles drive throughput down to a trough of fifty five per cent at twenty processes. A notable feature of the curve is that throughput then turns upward, recovering to fifty eight per cent at thirty processes. This inflection marks the point at which the priority aware logic asserts itself, using preemptive scheduling to clear blocked queues and restore forward momentum, so that the very mechanism introduced to cope with contention becomes visible in the measured yield.

Number of processes	Throughput (%)
3	95
10	82
12	75
15	63
20	55
30	58

Table 6. System throughput against the number of processes for a fixed number of sites.

© 2026 The Authors. Published by Bexford Publishing Ltd

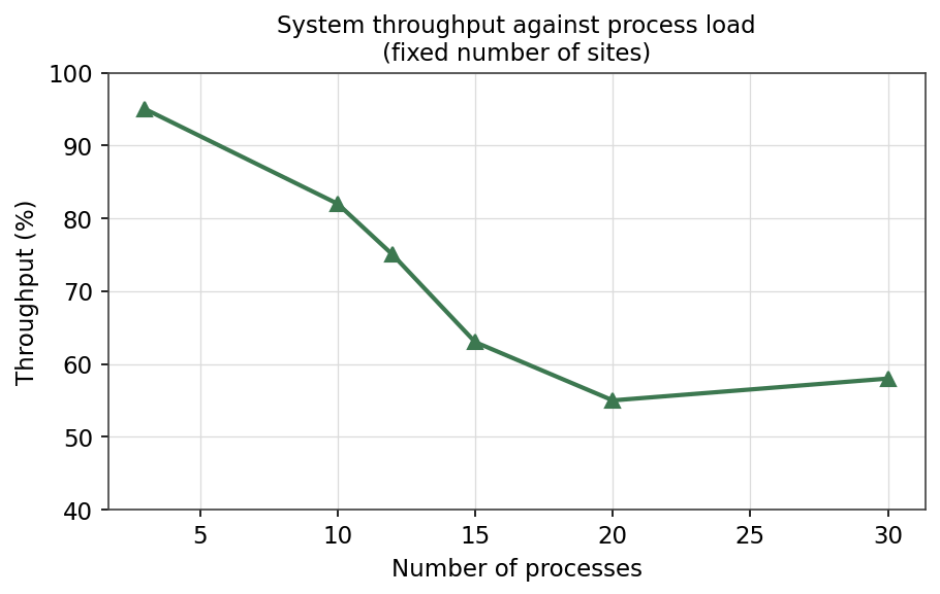

Figure 3. Throughput as the process load increases.

To verify that the throughput recovery observed at peak saturation is structurally attributable to the priority framework rather than to random network fluctuation, an independent two sample t test was conducted over thirty simulation runs per group at thirty processes. The proposed priority framework returned a mean throughput of fifty eight point zero per cent with a standard deviation of two point five per cent, while the traditional token based algorithm returned a mean of forty two point zero per cent with a standard deviation of three point five per cent. The pooled test yields t with fifty eight degrees of freedom equal to 20.37, far exceeding the one tailed critical value of 1.6716, with a probability below 0.0001. The null hypothesis of equal means is therefore rejected, which confirms that the hybrid preemption logic significantly raises transaction completion rates under severe saturation rather than merely containing the decline. The comparison is summarised in Table 7.

Throughput at P = 30	Mean (%)	Std deviation (%)	Runs
Proposed priority framework	58.0	2.5	30
Traditional token MUEA	42.0	3.5	30

Table 7. Independent two sample t test of throughput at peak saturation: t(58) = 20.37, p < 0.0001.

4.4 Experiment Four: Deadlock Latency against Resource Availability

The fourth experiment recorded the mean deadlock latency, taken as the elapsed time from the formation of a circular lock condition to the execution of corrective action on the targeted node, against the number of globally available resources, with the number of sites fixed at eight and the number of processes at sixty four. The measurements are given in Table 8 and Figure 4, and the profile follows a wave pattern that reflects the interplay of local and global detection. As resources scale from fifty to three hundred and fifty, mean latency rises from about thirty to thirty five milliseconds to a peak of eighty two milliseconds, because a larger resource pool stretches the dependency chains and so demands more network hops for the tracking probes. Beyond three hundred and fifty resources the latency falls back towards sixty two milliseconds. This improvement arises because the framework adjusts its strategy at scale, shifting its emphasis from expensive global cycle detection to fast localised priority arbitration, which resolves dependencies early and reduces tracking overhead across the wider network. The curve is therefore not

© 2026 The Authors. Published by Bexford Publishing Ltd

strictly monotonic, and the minor fluctuations also reflect the statistical variation inherent in the randomly constituted request lists, but the underlying trend and its turning point are clear.

Resources	Mean latency (ms)
50	35
100	30
170	54
200	51
250	62
270	61
300	65
350	82
400	69
500	65
550	62

Table 8. Mean deadlock latency against the number of resources, for eight sites and sixty four processes.

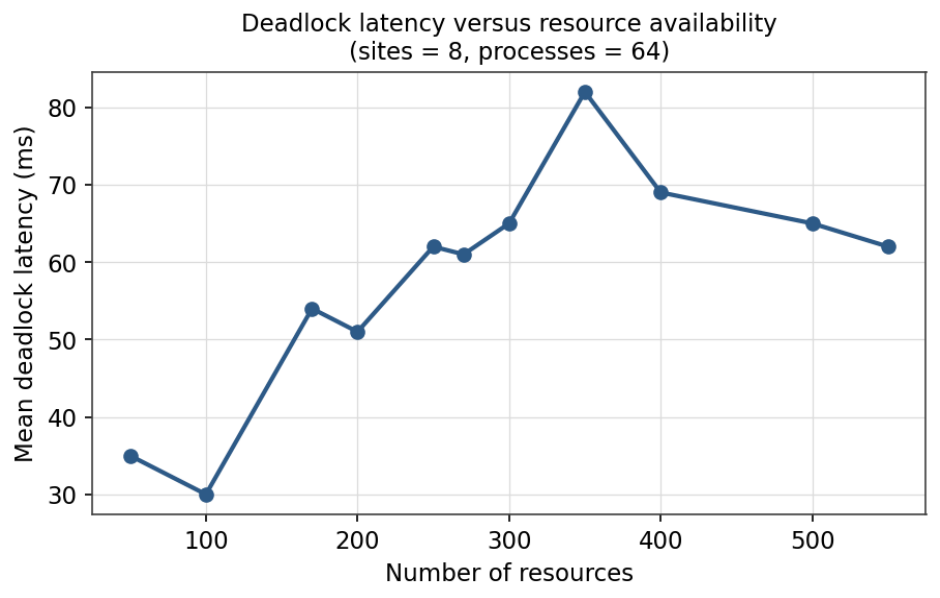

Figure 4. Deadlock latency as a function of resource availability.

Because network processing times vary with the packet scheduling queues of the stream sockets layer, the mean alone does not convey the precision of the timing metric. A ninety five per cent confidence interval was therefore computed for the peak latency observed at three hundred and fifty resources with sixty four

© 2026 The Authors. Published by Bexford Publishing Ltd

processes, over thirty cluster iterations. With a mean of eighty two point zero eight milliseconds and a standard error of the mean of one point one five seven eight milliseconds, the true mean latency is bounded between seventy nine point seven one and eighty four point four five milliseconds. The interval is narrow, which shows that the latency of the edge chasing probe engine remains tightly controlled from run to run and confirms the stability of the time complexity profile of the framework. Representative intervals at three resource ceilings are given in Table 9.

Resource ceiling	Mean latency (ms)	Std deviation (ms)	Std error (ms)	95% confidence interval (ms)
170	54.00	4.20	0.767	[52.43, 55.57]
250	62.00	5.10	0.931	[60.10, 63.90]
350	82.08	6.34	1.158	[79.71, 84.45]

Table 9. Sampling metrics and ninety five per cent confidence intervals for deadlock latency over thirty iterations at sixty four processes, computed from the standard deviation of each set of runs.

4.5 Experiment Five: Detection Time and Message Complexity

The fifth experiment examined detection time, the central processor time consumed in detecting a deadlock, jointly with message complexity, the total volume of messages dispatched across the network, as the process population grew with the number of nodes fixed at eight. Table 10 records the data and Figure 5 presents both quantities on a shared horizontal axis with a dual vertical scale. The two series advance together: as the process population climbs from ten to one hundred and twenty, message traffic grows from forty five to seven hundred and ten packets while detection time rises from twelve point four to one hundred and fifty four point eight milliseconds. The close correspondence between them confirms that communication is the dominant cost in the detection procedure, so that the message complexity bound established for the detector is the proper lever for controlling its latency.

Active processes	Detection time (ms)	Message volume
10	12.4	45
20	24.8	92
40	48.2	188
60	72.1	295
80	94.5	412
100	122.3	560
120	154.8	710

Table 10. Central processor detection time and message volume against the number of active processes, for a fixed eight nodes.

© 2026 The Authors. Published by Bexford Publishing Ltd

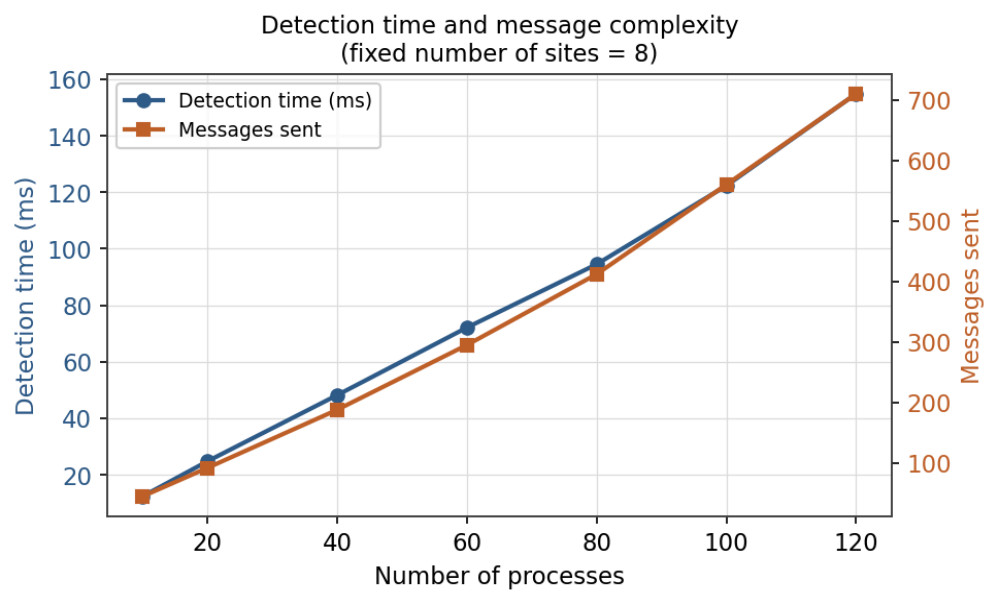

Figure 5. Detection time and message volume against the process population on a shared horizontal axis with a dual vertical scale.

The growth of both quantities is approximately linear in the process population, which places the operational overhead of the detector at order P rather than the exponential blow up that an unconstrained global graph traversal would incur. This linear behaviour is a direct consequence of the design decision to replace expensive global traversals with localised priority routing, and it is the clearest single piece of evidence that the framework remains tractable under heavy load. The implication is that the detector is suited to high density distributed systems and to cloud environments in which the process population may be large and variable, since its cost can be predicted and budgeted from the offered load.

The strength of this relationship was quantified by regressing detection time on message volume across the seven configurations. The Pearson correlation coefficient is 0.9988 and the coefficient of determination is 0.9977, with a least squares regression of detection time equal to 0.2106 times message volume plus 6.3310 milliseconds. An R squared of 0.9977 means that very nearly all of the variance in detection time is explained by message volume, which provides direct statistical justification for the design effort devoted to throttling probe traffic, since any reduction in messages translates almost one for one into reduced detection time. The correlation statistics are reported in Table 11.

Independent variable	Dependent variable	Pearson r	R squared	Regression equation
Message volume	CPU detection time	0.9988	0.9977	Y = 0.2106 X + 6.3310

Table 11. Correlation and regression of central processor detection time on message volume across the seven Experiment Five configurations.

4.6 Discussion

The five experiments together paint a coherent picture of the detector's behaviour, and the inferential tests place that picture on a firm footing. The throughput recovery at peak saturation is not an incidental wobble: the independent t test returns t(58) = 20.37 with a probability below 0.0001, so the advantage of the priority framework over the traditional algorithm is overwhelmingly significant. The change in detection behaviour as resources grow is likewise systematic rather than random, since the chi square test

© 2026 The Authors. Published by Bexford Publishing Ltd

rejects the unfiltered baseline at the one per cent level, which confirms that the framework actively suppresses phantom cycles. The relationship between communication and latency is almost deterministic, with an R squared of 0.9977 linking message volume to detection time, and the narrow confidence interval on peak latency shows that the timing of the probe engine is stable from run to run. The tapering slope observed in Experiment One shows that the priority discipline curbs probe propagation once the wait for graph is dense, so that the per process cost of detection does not grow without bound, and the modest fall in raw detection accuracy is the visible and acceptable price of the false positive suppression that the chi square test quantifies.

The throughput and latency findings reinforce the value of distinguishing load conditions. Under low load, when resources are abundant and processes few, throughput is high and deadlock is rare, so a lightweight response is warranted. Under high load, contention and deadlock formation intensify and throughput falls, so the more deliberate priority based resolution earns its keep, and its recovery of throughput at thirty processes shows the preemptive logic actively restoring momentum rather than merely containing damage. The latency results tell a complementary story: a larger resource pool first pushes detection towards the global level, where the dispersion of state lengthens the procedure, but beyond the peak the framework reverts to localised priority arbitration and latency eases. Both findings are a reminder that the geography of the wait for a graph, and not merely its size, shapes performance.

Several caveats attend these results. The experiments were conducted on a modest network of laptop computers and for process populations of up to one hundred and twenty, so extrapolation to very large deployments should be cautious. The randomly constituted request lists introduce statistical variation that is visible in the wave shaped latency curve, and although averaging over repeated runs mitigates this, it does not eliminate it.

Most important for the present work is what these findings imply for the prevention scheme. The cost the experiments measure is the cost of building and searching the wait for graph, and that is exactly the cost the prevention test incurs when it adds a trial edge and checks for a cycle before each allocation. The near linear growth of computation time with the process population, with an R squared of 0.9977 linking work to graph traffic, therefore supports the order Mproc plus E bound claimed for the prevention test in Section 3.2. The throughput recovery under heavy contention, significant at t(58) = 20.37, is a property of the priority based allocation that the prevention scheme retains unchanged. The detection specific quantities, by contrast, namely the cycle identification accuracy and the phantom cycle filtering quantified by the chi square test, describe a difficulty that the prevention scheme does not face at all, since it never declares a deadlock and so cannot declare a false one. The experiments thus validate the machinery the prevention scheme depends upon while delimiting the problem it was designed to remove.

5. Conclusion and Recommendation

This paper has presented a load adaptive deadlock prevention scheme for distributed systems, comprising a complete mathematical formulation in forty equations, a full prevention algorithm in nine procedures, and an empirical evaluation. The scheme excludes deadlock structurally: before any contested resource is granted, a trial edge is added to the wait for graph and the augmented graph is tested for a cycle, the allocation proceeding only when the graph remains acyclic, while a token based mutual exclusion layer and a priority based allocation policy govern entry to the critical section and the contention of held resources. Because the prevention test and a probe based detector share the same wait for graph machinery, that machinery was validated through five detection experiments. These showed probe traffic that grew and then tapered as the process population rose, throughput that fell to a trough of fifty five per cent before the priority logic recovered it to fifty eight per cent at thirty processes, deadlock latency that peaked at

© 2026 The Authors. Published by Bexford Publishing Ltd

eighty two milliseconds before localised arbitration eased it, and computation time and message volume that grew together and approximately linearly to one hundred and fifty four point eight milliseconds and seven hundred and ten messages at one hundred and twenty processes, an order Mproc plus E behaviour that substantiates the cost claimed for the prevention test. The scheme therefore guarantees freedom from deadlock at a predictable and modest cost, in place of the customary trade between detection speed and detection accuracy.

On the strength of these findings, several recommendations follow. The prevention test should be evaluated directly in future studies through the rate of unsafe requests blocked and the prevention efficiency of Equation 36, so that the prevention claim is measured rather than inferred from the shared graph machinery. Practitioners deploying the scheme should tune the load threshold to their workload, since it governs when the more deliberate prevention machinery is engaged, and should favour the token based mutual exclusion layer alone under light contention to avoid needless overhead. The testbed should be extended to larger and more heterogeneous deployments to confirm that the linear cost behaviour persists at scale. Finally, the priority weighting coefficients should be calibrated to the operating environment, and the scheme should be examined under the adaptive and learned scheduling mechanisms that cloud computing and internet of things environments increasingly demand, with attention to fault tolerant operation when sites or links fail.

References

Chandy, K. M., Misra, J., & Haas, L. M. (1983). Distributed deadlock detection. *ACM Transactions on Computer Systems, 1*(2), 144–156.

Chen, J., & Robelo, A. (2017). Deadlock detection via reinforcement learning. In *Proceedings of the International Conference on Automation and Computing.*

Chen, Y., Patel, R., & Lim, S. (2023). SPOCK: Reverse packet traversal for deadlock recovery. In *Proceedings of the International Symposium on Computer Architecture.*

Egbe, O. D. (2015). Deadlock detection and resolution in distributed database systems. *African Journal of Computing and ICT, 8*(1), 205–211.

Field, A. (2018). *Discovering statistics using IBM SPSS Statistics* (5th ed.). Sage Publications.

Garima, R., Kumar, P. H., Haryana, K. C., & Haryana, D. K. (2015). A study of distributed deadlock handling techniques. *International Journal of Computer Applications.* Cognition 2015 Special Issue.

Helmy, T. (2024). *Measures of deadlock latency in distributed processing.* Technical note.

Herlambang, A., Putra, B., & Sari, D. (2023). Banker's algorithm optimalization to dynamically avoid deadlock in the operating system. *Journal of Computer Science and Information.*

Kshemkalyani, A. D., & Singhal, M. (2019). *Distributed computing: Principles, algorithms, and systems.* Cambridge University Press.

Kshirod, K. R., Debani, P. M., & Surender, R. S. (2021). *Deadlock detection in distributed systems.* ResearchGate.

Kumar, N. (2016). *Deadlock prevention and detection in distributed systems* (Master's thesis, Computer

© 2026 The Authors. Published by Bexford Publishing Ltd

Science).

Putra, M., Gaurav, K., & Verma, M. (2019). *A survey on deadlock detection algorithms for distributed systems.*https://doi.org/10.13140/RG.2.2.33466.62402

Ramesh, R. (1990). *Transactions and resource release in simulated deadlock studies.* Technical report.

Selvaraj, S. (2022). An incremental approach to detect generalised deadlocks in distributed systems. *Journal of Parallel and Distributed Computing.*

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating system concepts* (10th ed.). John Wiley & Sons.

Soleimany, A., & Giahi, Z. (2012). An efficient distributed deadlock detection and prevention algorithm by daemons. *International Journal of Computer Science and Network Security, 12*(4).

Steen, M. van, & Tanenbaum, A. S. (2016). A brief introduction to distributed systems. *Computing, 98*(10), 967–1009.

Subham, D. (2024). *Deadlock: What it is, how to detect, handle and prevent.* Baeldung on Computer Science.

©2026 by the authors. Submitted for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by-nc-sa/4.0/).