A novel resource management technique for deadlock-free systems

Madhavi Devi Botlagunta¹ · Smriti Agrawal² · R. Rajeswara Rao³

Received: 5 March 2020 / Accepted: 13 April 2021 / Published online: 10 May 2021
© Bharati Vidyapeeth's Institute of Computer Applications and Management 2021

Abstract Deadlock in a shared resource system is a well-known problem. It has been extensively studied and recently a new class of resource reservation technique is researched upon for deadlock free resource management. This class of technique reserves a portion of the resources. The unreserved resources are freely allocated to any process demanding it. When the unreserved resources are not sufficient for a process demand the reserve pool resources are used such that the process completes and releases all the resources it is holding. This paper presents a new resource reservation technique resource driven DFRR. This technique estimates the optimal number of resources needed for a deadlock free resource reservation policy. The correctness is proved in the form of theorem 1. The theorem 2, suggests the resource reservation with minimal resources. The overhead of the resource pool estimation is $O(n)$ and that of resource management is $O(m)$ which is optimal for any deadlock handling technique. The effectiveness of the proposed technique is shown in the form of examples and simulation results.

Keywords Resource reservation · Banker's algorithm · Deadlock avoidance · Deadlock recovery · Operating systems · Scheduling · Safety sequence · Resource management · Deadlock-free

1 Introduction

Computing systems such as real-time [1, 2], IoT [3, 4], distributed, parallel, cluster, cloud computing systems, share resources for their effective utilization. The computing systems using shared resources, sometimes enter into a Deadlock, after which they are unable to execute further. Deadlocks adversely affect the system performance sometimes making it irresponsive so they are highly undesirable. Deadlock has been a big concern in both computing and non-computing shared resource systems and has been extensively studied for decades. Deadlocks will not always occur but as suggested by Coffman [5] it occurs if certain necessary conditions hold. These conditions are Mutual exclusion, Hold and wait, No pre-emption, and Circular wait condition. Thus, the first set of the Deadlock prevention (DP) resource management solution for a deadlock problem lies around these conditions. They suggest ensuring that one of the necessary conditions does not hold. The second set of Deadlock avoidance (DA), is a look forward technique to avoid deadlock in the future. The next set of solutions is event based where the system does not do any resource management rather waits for a deadlock event. The system periodically monitors for a deadlock occurrence and performs recovery action in case a deadlock is detected. Recently, a new class of techniques called Resource Reservation Techniques (RR) is also used for deadlock handling. These techniques are neither proactive nor reactive in the absolute sense. They reserve a

✉ Madhavi Devi Botlagunta
madhavi.polagangu@gmail.com

Smriti Agrawal
agrawal.smriti@gmail.com

R. Rajeswara Rao
rrrao.cse@jntukucev.ac.in

¹ Department of CSE, JNTU Kakinada, Hyderabad, India

² Department of IT, ChaitanyaBharathi Institute of Technology, Hyderabad, India

³ Department of CSE, JNTUK UCEV, JNTU Kakinada, Vizianagaram, Andhra Pradesh, India

set of resources and blindly allocate the remaining to processes for successful execution and also reduce the overhead usually caused by all other deadlock avoidance algorithms [6–9].

The main objective of this paper is to find optimal number of resources needed by processes to complete the execution without deadlocks in a system and also estimates how many minimal number resources of each type are sufficient. The correctness of the proposed technique can be proved in form of theorems and with the help of example and simulation results.

The rest of the paper is organized as follows: Sect. 2, provides a further look at the existing literature. Section 3 presents the proposed Resource driven-Deadlock-free technique and its effectiveness is explained with the help of example along with comparisons of other existing techniques. Section 4 illustrates experimental results and analysis. Finally, the paper concludes with Sect. 5 and Sect. 6 gives the future scope.

2 Related work

Resource allocation is a critical issue in shared or concurrent systems where resources are shared to avoid deadlocks. Authors [10] introduced a new metric Mean Normalized Blocking Time to compare locking mechanisms and also provide two different semaphore models for existing real-time systems in the automotive power environment. But these locking mechanisms with significant processing overhead reduce the system performance especially for multi core processing systems where processes execute on different cores in queues waiting to acquire a lock. So this implementation of more than one lock leads the risk of deadlock. The best solution is to eliminate locks. Removing the locks is significant overhead and propose performance based lock-free algorithms [11] in multi core systems which reduces the possibility of concurrency defects and risk of deadlock. They present a multi core communication API which enhances lock-free messaging and data exchange latency between processes. Authors [8] implemented resource request algorithm to avoid deadlocks with the help of simulating software. This algorithm grants requesting resources to processes for execution and analyze the state whether in safe or unsafe after every possible situation. But they fail to provide significant improvement in system performance and also deadlock-free. Authors [12] proposed a technique which is a combination of deadlock prevention and deadlock avoidance to detect a feasible resources and to reduce the false-positive situations by applying the Banker's algorithm. False-positive situations means it is a request, it may not actually lead to a deadlock, but still be rejected due to a failed deadlock-free test. Authors [13] proposed a new scheduling method to order products in supply

chain system. Deadlocks may arise while managing multiple supply chain flows simultaneously. To prevent capital chain ruptures and increase the suppliers money flow by adopting a famous deadlock avoidance Banker's algorithm and for this method correctness SPIN tool [14] is used. Youming Li [7, 15, 16] proposed an improved method over traditional Dijkstra's algorithm [17] in an optimal way without losing the simplicity of the data structures and also a graph-free with the cost of linear dependence on the number of the processes, with fixed number of resources and when the units of requests for resources are bounded by constants. But this is not practically possible with resource requests are constant. Begum et al. [18] proposed a safety detection algorithm over bankers algorithm [17] using linked list data structure to store resource requests and applied radix sort for comparisons to find safety sequence with the cost of linear dependence on number of the processes and number of resources. Agrawal et al. [3] proposed deadlock free resource reservation technique which is fairly allocating resources to all processes for their successful completion. But this has not give details of how to estimate resources sufficient for deadlock free system.

Another category of resource reservation policies which are improved over traditional deadlock avoidance algorithms by reserving some number of resources proposed by authors [19–21]. In this approach also considering maximum resource requirement known in advance is a major drawback. To overcome this drawback authors [22, 23] propose budget concept which is dynamically adjusts and reserve the resources for reducing the safety sequence calls overhead in traditional avoidance algorithms.

The Resource Reservation (RR) Techniques suggest to reserve some resources that can be used to avoid the deadlock. Botlagunta et al. [20] first introduced this idea and suggested to reserve resources based on a threshold. Agrawal et al. [19] suggested to use the total need of the processes to reserve the resources. The process with minimum worst case execution time reserved the resources as suggested in [21]. Kumar et al. [24], extended the technique suggested in [21] to reserve the resources based on the remaining resource need. Botlagunta et al. [22, 23] extended the RR techniques to dynamically allocate a budget of resources to each incoming process and perform resource reservation as per the technique suggested in [19, 20]. These techniques are discussed in detail in the subsequent section. All the deadlock avoidance techniques have high computation time as they must perform some test before each allocation to avoid deadlock. The Resource Reservation (RR) techniques eliminated this test for deadlock avoidance and thus, reduce the turnaround time of the processes.

Both DA and RR techniques presume that the system resource requirement is known in advance. The DA techniques perform some kind of test for ensuring that the

deadlock will not occur in the future. However, the Resource Reservation techniques do not perform any test rather use the knowledge of the resource requirement for estimating the portion of resources that must be reserved to avoid deadlock. The resource reserved by different RR Techniques existing so far is not sufficient for completely avoiding the deadlock. The present work extends the idea of the (RR) techniques to use the knowledge of the resource requirement by the processes for estimating the reserve pool strength. The proposed technique ensures that all the processes have the opportunity to complete with the reserved resources.

3 Proposed resource driven Deadlock free resource reservation (DFRR)

This work is an extension of the Deadlock-free resource reservation techniques (DFRR) [3]. The DFRR technique is free from deadlocks and its effectiveness is proved in the form of theorems. However, DFRR does not suggest how to estimate the resources that would be sufficient for a deadlock-free system. This work reexamines the DFRR technique from the systems point of view and proposes resource driven Deadlock free resource reservation (DFRR) technique. It considers a system that has a finite set of resources and manages its resources for a process set ensuring deadlock-free implementation. The system model used in this work is the same as used in [3]. The system consisting of $R_1, R_2, R_3, \dots, R_m$ resources with $\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_m$ instances of each type. These resources are portioned into $Reserve[j]$ and $Available[j]$ pools. This system executes n independent processes $P_1, P_2, P_3, \dots, P_n$ with their maximum resource requirement represented as $Max[i][j]$. The resources allocated to a process are represented as $Allocation[i][j]$, further resources needed are estimated as $Need[i][j]$ and requested are $Request[i][j]$.

The theorem 1, estimates the optimal resources used for a deadlock-free system. The DFRR further, investigates the systems with lower resources and suggests theorem 2 for a deadlock-free management. The proposed resource driven DFRR works on the same principle as DFRR but it examines the resources required by the processes and that available in the system. It suggested to reserve some resources in the reservation pool. The remaining resources are made available in the available pool. Whenever, a process requests for a set of resources, available pool resources are granted. In case the request cannot be granted only from the available pool then the reserved pool resources are also used such that the maximum resources this process can ever want are granted, giving it the chance to complete.

The theorem 1 is presented below to estimate the optimal number of resources needed for a deadlock-free system.

Theorem 1 A system with $\{\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_m\}$ instances of resource types $\{R_1, R_2, R_3, \dots, R_m\}$ can schedule a set of independent processes $\{P_1, P_2, P_3, \dots, P_n\}$ with resource requirement as $Max[i][j]$, in deadlock-free manner if $((\sum_{i=1}^n Max[i][j]) - (\eta_j - 1) * Reserve[j]) \leq \alpha_j$ where

- i. η_j is the number of processes requesting for the resource R_j i.e., $count(Max[i][j] \neq 0 \forall i = 1, 2, \dots, n)$
- ii. $Reserve[j] = \lceil \lfloor Max[i][j] \forall i = 1, 2, \dots, n \rfloor, 0 \rceil$
- iii. A process P_i can acquire either
 - a. $Max[i][j] - Reserve[j]$ resources from the available pool,
 - OR
 - b. $Max[i][j] \forall R_j$ from available and reserve pool put together such that the available pool is consumed first. That is a process must acquire all the resource it might need.
- iv. A process P_i on completion relinquishes all the resources, $Max[i][j] \forall R_j$, that are returned to the reserve and available pools as per condition i.

Proof By definition:

$$Available[j] = \alpha_j - Reserve[j], \forall R_j \quad (1)$$

In worst case as per the condition iii. a. the resources any process can acquire is as per Eq. (2).

$$Allocation[i][j] = Max[i][j] - Reserve[j] \forall R_j \quad (2)$$

Thus, each process needs $Reserve[j]$ resources to complete its execution.

Total number of resources allocated to all the processes is $\sum_{i=1}^n Allocation[i][j]$.

Ideally,

$$\sum_{i=1}^n Allocation[i][j] \leq Available[j], \forall R_j \quad (3)$$

Substituting Eq. (1) and (2) in (3)

$$\Rightarrow \sum_{i=1}^n Allocation[i][j] \leq \alpha_j - Reserve[j], \forall R_j$$

$$\Rightarrow \sum_{i=1}^n (Max[i][j] - Reserve[j]) \leq \alpha_j - Reserve[j], \forall R_j$$

$$\Rightarrow \sum_{i=1}^n (Max[i][j]) - \sum_{i=1}^n Reserve[j] \leq \alpha_j - Reserve[j], \forall R_j$$

$$\Rightarrow \sum_{i=1}^n Max[i][j] - n * Reserve[j] \leq \alpha_j - Reserve[j], \forall R_j$$

$$\Rightarrow \sum_{i=1}^n \text{Max}[i][j] - (n-1) * \text{Reserve}[j] \leq \alpha_j, \forall R_j$$

Thus, $((\sum_{i=1}^n \text{Max}[i][j]) - (\eta_j - 1) * \text{Reserve}[j]) \leq \alpha_j, \forall j = 1, 2, \dots, m$ resources are sufficient for deadlock-free execution. Hence Proved. $\square$

A system with $\alpha_j \geq (\sum_{i=1}^n \text{Max}[i][j]) \forall j = 1, 2, \dots, m$ is a generic deadlock-free system because it has more resources than it can ever consume with all the processes holding their complete set. However, a system with $\alpha_j = ((\sum_{i=1}^n \text{Max}[i][j]) - (\eta_j - 1) * \text{Reserve}[j])$ has optimal number of resources. This is actually under-utilization of resources. This value will always be lower than $(\sum_{i=1}^n \text{Max}[i][j]) \forall j = 1, 2, \dots, m$, indicating that the DFRR can achieve deadlock-free system with lower number of resources. This estimated $\alpha_j = ((\sum_{i=1}^n \text{Max}[i][j]) - (\eta_j - 1) * \text{Reserve}[j])$ is referred of α_{opt} . The overhead is only $O(n)$ for estimating the reserve pool initially. This technique as with all other reservation technique does not perform any kind of testing before allocating the resources and hence, has minimal overhead.

The following example illustrates the effectiveness of the proposed technique and compares it with other existing reservation techniques.

Example: consider a set of processes $\{P_1, P_2, P_3, P_4, P_5\}$ with the resource requirement as suggested in Table 1 The chronological sequence of resource requests as shown in the Table 2.

The resource management by all the existing techniques is examined in the following subsections

1. **Banker's algorithm (BA) [9]:** The Banker's algorithm estimates the safety sequence (SS) before each request is granted or not granted. Table 2, displays the set of request made by the process set. All the requests ($Req_1 - Req_{16}$), are granted and the safety sequence is estimated as $\langle P_1, P_2, P_3, P_4, P_5 \rangle$. The system status can be viewed in the Table 3. The unallocated resources are (2, 2, 3, 3). However, when the request Req_{17} for

Table 1 Snapshot of the system at the start time t_0

	Allocation				Maximum				Need			
	R ₁	R ₂	R ₃	R ₄	R ₁	R ₂	R ₃	R ₄	R ₁	R ₂	R ₃	R ₄
P_1	0	0	0	0	5	6	3	4	5	6	3	4
P_2	0	0	0	0	2	0	6	0	2	0	6	0
P_3	0	0	0	0	3	0	3	4	3	0	3	4
P_4	0	0	0	0	6	6	3	3	6	6	3	3
P_5	0	0	0	0	4	3	4	5	4	3	4	5
Resource pool												
	12	9	7	7								

(0, 0, 0, 1) by process P_5 is made the BA does not find a SS, implying that it is a unsafe state. The process P_5 is sent to the pending queue. Similarly, the requests of the process P_4, P_3, P_2 are also not granted and after the SS estimation they all are sent to the pending queue. Finally, all the requests ($Req_{21} - Req_{30}$) (refer Table 2) of process P_1 are granted step wise. Once the process P_1 completes it releases all the resources it was holding and processes from the queue are retrieved.

2. **Threshold based resource allocation technique (TRA) [20]:** The TRA reserve resources based on a threshold estimated using $\text{Threshold}[j] = \lceil \frac{\text{Need}[i][j]}{n} \rceil \forall i = 1, 2, \dots, n$ as (2, 3, 3, 3). All the requests (Req_1 to Req_{15}) are granted by this technique also, this is same as the BA and the system is in safe state so far. Its state is as shown in Table 4. The following request Req_{16} for (0, 1, 0, 0) by process P_1 cannot be granted from the Available pool so it is given resources from the reserve pool. The process thus, gets all the resources it need and completes and releases the resources it was holding. These resources can then be used by the remaining process likewise.
3. **Total need based resource reservation (TNRR) technique [19]:** The TNRR technique reserves the resources based on the $\text{Reserve}[j] = \lceil \frac{(\text{Total_Need}[j] - \alpha_j)}{n} \rceil$ where Total_Need is the sum of the Need of the processes. For Table 1, the $\text{Total_Need} = (20, 15, 19, 16)$. The reserve and available pools are (2, 2, 3, 2) and (10, 7, 4, 5) respectively. The requests as per Table 2 (Req_1 to Req_{16}) are all granted, the available pool is then (0, 0, 0, 1). The reserve pool is untouched. The request Req_{17} for (0, 0, 0, 1) by process P_5 can be satisfied from the available pool and is also granted. The state of the system is as in Table 5. The reserve pool resources (2, 2, 3, 2) is insufficient to satisfy the need of any of the process thus, the system in an unsafe state and will eventually end in a deadlock.
4. **Worst-case execution time based resource reservation (ETRR) Technique [7]:** The shortest execution time process is selected for the resource reservation. In the example given in Table 1, the shortest process is P_1 thus, (5, 6, 3, 4) resources are reserved. The requests (Req_1 to Req_9) are granted, however request Req_{10} for (1, 1, 0, 0) by process P_4 , cannot be satisfied from the available pool. The process is thus, assigned resources from the available and reserve pool put together. It then completes and hand over the resources it was holding. This technique works well when the resource requirement of the shortest process is high. However, if the shortest job was process P_2 then this system would have ended in a deadlock.

Table 2 Sequence of requests

Resource requests	Process requesting resources	Request (R_1, R_2, R_3, R_4)	Banker's Algorithm	TRA	TNRR	ETRR	DFRR
Req_1	P_2	(0, 0, 1, 0)	Grants	Grants	Grants	Grants	Grants
Req_2	P_4	(1, 1, 0, 0)	Grants	Grants	Grants	Grants	Grants
Req_3	P_3	(1, 0, 0, 0)	Grants	Grants	Grants	Grants	Grants
Req_4	P_2	(0, 0, 1, 0)	Grants	Grants	Grants	Grants	Grants
Req_5	P_2	(0, 0, 1, 0)	Grants	Grants	Grants	Grants	Grants
Req_6	P_3	(0, 0, 0, 1)	Grants	Grants	Grants	Grants	Grants
Req_7	P_1	(1, 1, 0, 0)	Grants	Grants	Grants	Grants	Grants
Req_8	P_5	(1, 0, 0, 1)	Grants	Grants	Grants	Grants	Grants
Req_9	P_4	(1, 1, 0, 0)	Grants	Grants	Grants	Grants	Grants
Req_{10}	P_4	(1, 1, 0, 0)	Grants	Grants	Grants	GRP**	Grants
						P_4 completes	
Req_{11}	P_1	(1, 1, 0, 0)	Grants	Grants	Grants	Grants	Grants
Req_{12}	P_5	(1, 0, 0, 1)	Grants	Grants	Grants	grants	Grants
Req_{13}	P_1	(1, 1, 0, 1)	Grants	Grants	Grants	GRP**	Grants
						P_1 completes	
Req_{14}	P_4	(1, 0, 0, 0)	Grants	Grants	Grants	P_4 complete so this request do not appear	Grants
Req_{15}	P_5	(0, 0, 1, 0)	Grants	Grants	Grants	Grants	Grants
Req_{16}	P_1	(0, 1, 0, 0)	Grants	GRP**	Grants	P_1 complete so this request do not appear	GRP**
				P_1 completes			P_1 completes
Req_{17}	P_5	(0, 0, 0, 1)	NG	Grants	Grants(unsafe state)	GRP**	Grants
			All processes except P_1 are in pending queue			P_5 completes	
Req_{18}	P_4	(0, 1, 0, 0)		Grants	NG*	Do not appear	Grants
Req_{19}	P_3	(1, 0, 0, 0)		Grants	Deadlock	Grants	Grants
Req_{20}	P_2	(0, 0, 1, 0)		GRP**		Grants	GRP**
				P_2 completes			P_2 completes
Req_{21}	P_1	(1, 0, 0, 0)	Grants	Process P_1 is already completed so these requests do not appear		Process P_1 is already completed so these requests do not appear	Process P_1 is already completed so these requests don't appear
Req_{22}	P_1	(0, 1, 0, 0)	Grants				
Req_{23}	P_1	(1, 0, 0, 0)	Grants				
Req_{24}	P_1	(0, 0, 1, 0)	Grants				
Req_{25}	P_1	(0, 0, 0, 1)	Grants				
Req_{26}	P_1	(0, 1, 0, 0)	Grants				
Req_{27}	P_1	(0, 0, 1, 0)	Grants				
Req_{28}	P_1	(0, 0, 0, 1)	Grants				
Req_{29}	P_1	(0, 0, 1, 0)	Grants				
Req_{30}	P_1	(0, 0, 0, 1)	Grants				
			P_1 completes			P_1 completes	

NG* Not Granted (moved to pending queue), GRP** Grants from Reserve Pool

5. **Proposed DFRR:** The proposed resource driven DFRR technique considers the nascent system in the Table 1 with no initial allocation. The reserve pool is estimated using equation $Reserve[j] = \lceil [Max[i][j] \forall i = 1, 2, \dots, n], 0 \rceil$ as (2, 3, 3, 3) and the remaining (10, 6, 4,

4) resources are in the available pool. The requests (Req_1 to Req_{15}) for the resources are granted and the system state can be seen the Table 4. It may be noted that the safety sequence is never calculated and reserve pool is also estimated only once. When the request

Table 3 Data structures after request Req_{16} of (0, 1, 0, 0) is granted to P_1

	Allocation				Maximum				Need			
	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4
P_1	3	4	0	1	5	6	3	4	2	2	3	3
P_2	0	0	3	0	2	0	6	0	2	0	3	0
P_3	1	0	0	1	3	0	3	4	2	0	3	3
P_4	4	3	0	0	6	6	3	3	2	3	3	3
P_5	2	0	1	2	4	3	4	5	2	3	3	3
									Resource pool			
									2	2	3	3
Safety sequence $\langle P_1, P_2, P_3, P_4, P_5 \rangle$												

Table 4 Data structures before Req_{16} of (0, 1, 0, 0) is granted to P_1

	Allocation				Maximum				Need			
	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4
P_1	3	3	0	1	5	6	3	4	2	3	3	3
P_2	0	0	3	0	2	0	6	0	2	0	3	0
P_3	1	0	0	1	3	0	3	4	2	0	3	3
P_4	4	3	0	0	6	6	3	3	2	3	3	3
P_5	2	0	1	2	4	3	4	5	2	3	3	3
									Available pool			
									0	0	0	0
									Resource pool			
									2	3	3	3
Safety sequence $\langle P_1, P_2, P_3, P_4, P_5 \rangle$												

Table 5 Data structures after Req_{17} of (0, 0, 0, 1) is granted to P_5

	Allocation				Maximum				Need			
	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4
P_1	3	4	0	1	5	6	3	4	2	2	3	3
P_2	0	0	3	0	2	0	6	0	2	0	3	0
P_3	1	0	0	1	3	0	3	4	2	0	3	3
P_4	4	3	0	0	6	6	3	3	2	3	3	3
P_5	2	0	1	3	4	3	4	5	2	3	3	2
									Available pool			
									0	0	0	0
									Resource pool			
									2	2	3	2

Req_{16} is made by P_1 the available pool is empty so the resources from the reserve pool are allocated to it to accelerate its completion lowering its turnaround time. Elimination of the SS estimation also lowers the wait time and hence, the turnaround time of all the processes. The process P_1 on its completion releases all the resources it is holding and system continues its operation.

Another interesting question to ask is what happens if number of resources in the system are lower than α_{opt_j} that

is $\alpha_j < \alpha_{opt_j}$? If the resources can be allocated as per the theorem 1, the system may end up in a deadlock. The resource driven DFRR technique suggests to modify the resource allocation policy whose correctness is proved in the theorem 2.

Theorem 2 A system with $\{\alpha_1, \alpha_2, \alpha_3 \dots \alpha_m\}$ instances of resource types $\{R_1, R_2, R_3 \dots R_m\}$ can schedule a set of independent processes $\{P_1, P_2, P_3 \dots P_n\}$ with resource requirement as $Max[i][j]$, in deadlock-free manner if $\alpha_j \geq \lceil (Max[i][j]) \rceil$ where

- η_j is the number of processes requesting for the resource R_j i.e., $count(Max[i][j]) \neq 0 \forall i = 1, 2 \dots n$
- $Reserve[j] = \lceil \lfloor Max[i][j] \rfloor \forall i = 1, 2 \dots n \rfloor, 0 \rceil$
- A process P_i can acquire either
 - $\lceil Max[i][j] - Reserve[j] - \left\lceil \frac{\alpha_{opt_j} - \alpha_j}{\eta_j - 1} \right\rceil, 0 \rceil$ where $\alpha_{opt} = ((\sum_{i=1}^n Max[i][j]) - (\eta_j - 1) * Reserve[j])$ resources from the available pool such that $\alpha_j \leq \alpha_{opt_j}$,

OR

b. $Max[i][j] \forall R_j$ from available and reserve pool put together such that the available pool is consumed first. That is a process must acquire all the resource it might need.

iv. A process P_i on completion relinquishes all the resources, $Max[i][j] \forall R_j$, that are returned to the reserve and available pools as per condition i.

Proof Proving by contradiction that total resources allocated are greater than available in the system:

$$\sum_{i=1}^n Allocation[i][j] > Available[j], \forall R_j \quad (4)$$

Substituting in (4): $Available[j] = \alpha_j - Reserve[j], \forall R_j$

$$\Rightarrow \sum_{i=1}^n Allocation[i][j] > \alpha_j - Reserve[j], \forall R_j \quad (5)$$

Substituting in (5): $Allocation[i][j] = Max[i][j] - Reserve[j] - \left(\frac{\alpha_{opt_j} - \alpha_j}{n-1} \right)$

as per the equation iii. a. from theorem 2.

$$\Rightarrow \sum_{i=1}^n \left(Max[i][j] - Reserve[j] - \left\lceil \frac{\alpha_{opt_j} - \alpha_j}{\eta_j - 1} \right\rceil \right) > \alpha_j - Reserve[j], \forall R_j$$

In worst case $\eta_j = n$ implying that the resource is being used by all the processes in the system.

$$\Rightarrow \sum_{i=1}^n \left(Max[i][j] - Reserve[j] - \left(\frac{\alpha_{opt_j} - \alpha_j}{n-1} \right) \right) > \alpha_j - Reserve[j], \forall R_j$$

$$\Rightarrow \left(\sum_{i=1}^n \text{Max}[i][j] - (n-1) * \text{Reserve}[j] - \left(\frac{n * (\alpha_{opt_j} - \alpha_j)}{n-1} \right) \right) > \alpha_j, \forall R_j$$

Substituting as suggested in the theorem 2 iii. a. ;
 $\alpha_{opt} = ((\sum_{i=1}^n \text{Max}[i][j]) - (\eta_j - 1) * \text{Reserve}[j])$

In worst case $\eta_j = n$ implying that the resource is being used by all the processes in the system. Therefore,

$$\alpha_{opt} = \left(\left(\sum_{i=1}^n \text{Max}[i][j] \right) - (n-1) * \text{Reserve}[j] \right)$$

$$\text{Thus, } \Rightarrow \left(\alpha_{opt_j} - \left(\frac{n * (\alpha_{opt_j} - \alpha_j)}{n-1} \right) \right) > \alpha_j, \forall R_j$$

$$\Rightarrow (-\alpha_{opt_j}) > -\alpha_j, \forall R_j$$

However, $\alpha_j < \alpha_{opt_j}$, hence contradiction. Therefore, $\sum_{i=1}^n \text{Allocation}[i][j] \leq \text{Available}[j], \forall R_j$, implying that the system remains feasible. Hence Proved. $\square$

In the above example if the resources are reduced from $\alpha_{opt} = (12, 9, 7, 7)$ to $\alpha = (10, 8, 6, 6)$. The total allocation possible is $\sum_{i=1}^n \text{Allocation}[i][j] = (6, 4, 2, 1)$, as per condition iii. a. from theorem 2, therefore the remaining resources are $(10, 8, 6, 6) - (6, 4, 2, 1) = (4, 4, 4, 5)$, sufficient for any process to complete hence, this system remains deadlock free with lower resources also. The following section present the simulation results.

4 Simulations results

This section presents the results of the simulations performed on the synthesized process sets generated. The performance of the proposed DFRR is compared with various existing techniques.

Resource instances Vs. Avg. turnaround time:

- Fig. 1 shows the effect of change in the number of instances (α_j) of a resource type on the average turnaround time of the process set. 100 process sets were generated with 6 processes, and one resource type and its distribution the same in all the process sets. The instances of the resource are varied from its minimum acceptable value of $\lceil (\text{Max}[i][j]) \rceil$ (as per theorem 2) to $(\sum_{i=1}^n \text{Max}[i][j])$. The DFRR performs best at the α_{opt} . The DFRR performs approximately 18% better than the BA. This is because as the number of resources is reduced the BA sends the processes into pending queue over and over again. The same is indicated by the example. All the reservation techniques promote the process with resource crunch to assist it to complete. However, the TRA, TNRR, and ETRR sometimes land into

Fig. 1 Resource instances Vs Avg. turnaround time

Fig. 2 Resource bound Vs Avg. turnaround time

deadlock increasing their average turnaround time. Resource bound Vs. Avg. Turnaround Time: a process can be resource bound or CPU bound [2, 25, 26]. A CPU bound process is a process whose major execution depends on the CPU and it has little or no resource requirement. Similarly, if the process execution depends on the resources then such a process is resource bound.

- Fig. 2, shows the effect on the avg. turnaround time as the resource dependency increases. The value '0' indicates that the process is CPU bound while value '1' indicates that the process is majorly resource bound. The avg. turnaround time for all the techniques is same as there is no resource requirement and hence, no resource management. However, as the resources dominate the process execution the BA spends more time in decision making increasing the wait time of a process. The resource reservation techniques are hence, better equipped to handle resource requests. The DFRR which is deadlock free performs the best for all the ranges as it never enters into deadlocks.

5 Conclusion

This paper presented a new resource reservation technique for resource management in shared resource systems. The proposed DFRR technique is a deadlock-free resource management technique. Unlike other resource reservation techniques, DFRR suggests resource management based on the resources in the system. It suggests a deadlock free resource management for minimal resource system. It also suggests the optimal number of resources needed for a process set to efficiently execute on a system. The correctness of the technique is proved in the form of two theorems. The theorem 1, answers, given a process set how many resources will be sufficient for its efficient execution. While theorem 2, suggest resource management based on the resources available in the system to the demanding process set. Its effectiveness is demonstrated by an example and simulation results.

The DFRR belongs to resource reservation class of deadlock handling techniques. This class of techniques eliminate the excessive overhead incurred by the Deadlock Prevention and Avoidance techniques as well as the uncertainty of deadlock occurrence in the deadlock detection and recovery techniques. In other words, the proposed technique achieved deadlock freeness as any deadlock avoidance technique with the overhead of a deadlock detection and recovery technique, which is optimal. The overhead in these technique is only for the reservation pool estimation. The proposed technique estimates the reservation pool by finding a minimum in an array of 'n' where 'n' is the number of processes in the system. Hence, the overhead for reservation pool estimation is $O(n)$. The simulation results indicate that the turnaround time of the proposed DFRR is approximately 18% better than the existing Banker's algorithm. Thus, the proposed technique is a deadlock-free technique with optimal overhead.

6 Future work

This work considers resource management based on the resources available in the system to the demanding process set. This work can be applied to study resource management during hospitalization due to the present pandemic of Covid-19.

References

1. Agrawal S, Yadav RS (2020) Transient fault tolerance patterns for real time systems with arbitrary deadline. *Int J Sci Technol Res* 9(3):6832–6848
2. S. Agrawal, R. S. Yadav, N. Das (2007) Checkpointing based fault tolerance patterns for systems with arbitrary deadlines. In *Proceedings of the 15th International Conference on Advanced Computing and Communications, ADCOM 2007* pp. 694–699, doi: <https://doi.org/10.1109/adcom.2007.96>.
3. Devi MB, Agrawal S, Rao RR (2020) Deadlock free resource management technique for iot-based post disaster recovery systems. *Scalable Comput* 2020:1. <https://doi.org/10.12694/scpe.v21i3.1734>
4. Sathish Kumar J, Zaveri MA (2019) Resource scheduling for postdisaster management in IoT environment. *Wirel Commun Mob Comput*. <https://doi.org/10.1155/2019/7802843>
5. Coffman EG, Elphick MJ, Shoshani A (1971) System deadlocks. *ACM Comput Surv* 3(2):67–78
6. Y. Wang, T. Kelly, M. Kudlur, S. Lafortune, S. Mahlke (2019) Gadar: Dynamic deadlock avoidance for multithreaded programs. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2008*
7. Y. Li (2013) A modified Banker's algorithm. In *Lecture Notes in Electrical Engineering* doi: https://doi.org/10.1007/978-1-4614-3535-8_23
8. Salim AW et al (2019) Implementation resource request algorithm in simulation of deadlock avoidance. *J Phys Conf Ser* 1230(1):012096. <https://doi.org/10.1088/1742-6596/1230/1/012096>
9. Ugwuanyi EE, Ghosh S, Iqbal M, Dagiuklas T (2018) Reliable resource provisioning using bankers' deadlock avoidance algorithm in MEC for industrial IoT. *IEEE Access* 6(August):43327–43335. <https://doi.org/10.1109/ACCESS.2018.2857726>
10. M. Alfranseder, M. Mucha, S. Schmidhuber, A. Sailer, M. Niemetz, J. Mottok (2013) A modified synchronization model for dead-lock free concurrent execution of strongly interacting task sets in embedded systems. In *International Conference on Applied Electronics*, 2013.
11. K. E. Harper, T. De Gooijer (2020) Performance Impact of Lock-Free Algorithms on Multicore Communication APIs. Accessed: May 30, 2020. [Online]. Available: <http://www.opengi.org>.
12. Cui Y, Martin U, Liang J (2017) Searching feasible resources to reduce false-positive situations for resolving deadlocks with the Banker's algorithm in railway simulation. *J Rail Transp Plan Manag* 7(1–2):50–61. <https://doi.org/10.1016/j.jrtpm.2017.05.001>
13. Z. Wu, S. Uk, J. Lee (2016) The Application of Banker's Algorithm in Order Scheduling Management for Deadlock Avoidance Software Product Line View project ICT Software Platform View project. [Online]. Available: <https://www.researchgate.net/publication/318315690>.
14. Holzmann G, Najm E, Serhrouchni A (2000) SPIN model checking: an introduction. *Int J Softw Tools Technol Transf* 2(4):321–327. <https://doi.org/10.1007/s100090050039>
15. Li Y, Greca A, Harris J (2010) On Dijkstra's algorithm for deadlock detection. In: *advanced techniques in computing sciences and software engineering*. Springer, Netherlands, pp 385–387
16. Li Y, Cook R (2007) "A new algorithm and asymptotical properties for the deadlock detection problem for computer systems with reusable resource types. *Adv Innov Syst Comput Sci Softw Eng*. https://doi.org/10.1007/978-1-4020-6264-3_88
17. Dijkstra EW (1982) The mathematics behind the banker's algorithm. In: *Selected writings on computing: a personal perspective*. Texts and monographs in computer science. Springer, New York, NY. https://doi.org/10.1007/978-1-4612-5695-3_54
18. M. Begum, O. Faruque, M. W. Rahman Miah, B. Chandra Das (2020) An Improved Safety Detection Algorithm Towards Deadlock Avoidance. *ISCAIE 2020 - IEEE 10th Symposium on*

- Computer Applications & Industrial Electronics pp. 73–78, 2020, doi: <https://doi.org/10.1109/ISCAIE47305.2020.9108818>.
19. Agrawal S, Devi Botlagunta M, Srinivasulu C (2013) A total need based resource reservation technique for effective resource management. *Int J Comput Appl* 68(18):12–17. <https://doi.org/10.5120/11678-6306>
 20. Devi BM, Agrawal S, Srinivasulu C (2013) An efficient resource allocation technique for uniprocessor system. *Int J Adv Eng Technol* 6(1):353–360
 21. M. D. Botlagunta, S. Agrawal, R. R. Rao (2014) Effective resource management technique using reservation pool. In *International Conference on Recent Advances and Innovations in Engineering, ICRAIE 2014*, doi: <https://doi.org/10.1109/ICRAIE.2014.6909221>.
 22. Botlagunta MD, Agrawal S, Rao RR (2019) “Dynamic budget-threshold based resource reservation technique. *Compusoft* 8(7):61
 23. Botlagunta MD, Agrawal S, Rajeshwara Rao R (2020) Dynamic budget-total need based resource reservation technique. *Smart Sci.* <https://doi.org/10.1080/23080477.2020.1778226>
 24. Kumar S, Chandran S (2016) Modified execution time based resource reservation (METRR) algorithm. In: Presented at ICBIM 2016. NIT Durgapur, India.
 25. Agrawal S, Yadav RS, Vijay R (2010) A speed fine tuning technique for system energy minimization of weakly hard real-time system. *Int J Comput Appl.* <https://doi.org/10.1080/1206212X.2010.11441975>
 26. S. Agrawal, R. S. Yadav, Ranvijay (2008) System level energy aware fault tolerance approach for real time system. In *IEEE Region 10 Annual International Conference, Proceedings/TENCON, 2008*, doi: <https://doi.org/10.1109/TENCON.2008.4766854>