# A Total Need based Resource Reservation Technique for Effective Resource Management

Smriti Agrawal
Department of Information Technology,
JB Institute of Engineering and Technology, Hyderabad, India

Madhavi Devi Botlagunta
Department of Information Technology,
JB Institute of Engineering and Technology, Hyderabad, India

Chennupalli Srinivasulu
Department of Information Technology,
JB Institute of Engineering and Technology, Hyderabad, India

## ABSTRACT
In multiprocessor environment when processes content for system resources, Deadlocks may occur. Deadlock is highly undesirable as it degrades the system performance largely. This paper aims to present technique to facilitate the resource allocation decision. It also strives to reduce the time cost for making this decision. It presents a Total Need Based Resource Reservation (TNRR) that suggests reserving some resources so as to ensure that at least one process will complete after it. The motivational example illustrate that the proposed technique is capable of performing resource allocation without checking the safety sequence as proposed by existing Banker's algorithm. The overhead for this decision for proposed TNRR is merely $O(m)$ as compared to Banker's algorithm of the $O(mn^2)$. The simulation results indicate that the frequency of deadlocks has reduced by approximately 75% for higher load (above 80%) as compared to the Deadlock Recovery technique, while for lower load it tends to be zero. The turnaround time of the TNRR is approximately 9% better than the existing Banker's algorithm.

## General Terms
Algorithms, Deadlock, Deadlock avoidance, Deadlock Recovery

## Keywords
Banker's Algorithm, Deadlock, Deadlock avoidance, Deadlock Recovery, Operating systems, Scheduling, Safety sequence.

## 1. INTRODUCTION

In a system, processes may be interleaved in time to give the user a feel that they are running simultaneously. They are being executed on a single processor, who switches back and forth between them. A process apart from the processor may require other resources for its successful completion. These resources may be sharable or non-sharable. A *sharable* resource is one that can be shared by more than one process at any given time. Resources that cannot be acquired by more than one process at any time are referred to as *non-sharable*. Sharable resources do not lead to any conflict of interest between any two processes. However, when two or more process demand for a non sharable resource the decision of to whom the resource is granted or not granted at all becomes crucial. This decision if not wisely taken may lead the system into a deadlock state. Formally, Deadlock can be defined as the permanent blocking of a set of processes that demand for a set of non sharable resources [1, 2, 3, 8, 13].
Deadlocked processes never terminate their executions and the resources held by them are not available to any other process. It is a permanent situation as none of the demand imposed by any of the process can ever be met. This in turn leads to poor resource utilization, lower throughput, i.e., performance degradation. Deadlock is a common problem where conflicting demand for the resources by two or more processes occur including multiprocessing systems [18, 19, 22], parallel computing [5, 6], cloud computing [14, 15, 16, 17, 22] and distributed systems [4, 8, 9, 21]. The present paper discusses the resource reservation technique for single processor system; however, it can be adapted for the multi-processor systems, i.e., parallel, cloud and distributed systems.

For systems such as automated manufacturing systems, distributed systems, control systems etc. deadlock is highly undesirable, because it may lead to catastrophic losses. Coffman [1, 8, 9] studied and suggested necessary conditions for a deadlock to occur. These conditions are: Mutual exclusion, Hold and wait, No preemption and Circular wait condition. The authors [1, 8, 9], suggested the deadlock prevention, deadlock avoidance and deadlock detection techniques for handling the deadlocks.

The deadlock prevention technique can prevent deadlock if and only if, one of the four necessary conditions stated above fails to hold. However, Mutual exclusion, Hold and wait, and No preemption conditions are system dependant and may not be prevented [1, 8, 9]. Thus, preventing circular wait from occurring is the best way for preventing deadlock. This can be achieved by using a hierarchy to determine a partial ordering of resources [8, 9]. However, it may not be possible to implement it in most cases.

Some prior knowledge about the upcoming resource requirement if available can be used for taking wise decisions so as to avoid deadlocks. A well known deadlock avoidance algorithm used in operating systems is the Banker's algorithm which was proposed by Dijkstra to handle a single resource type [4, 5]. This Banker's algorithm required $O(n^2)$ time and $O(n)$ space for handling requests of 'n' processes. It was extended by Habermann to handle multiple resource types [4, 5, 6, 7, 13, 18] leading to the time requirement of $O(mn^2)$ and space as $O(mn)$.

Banker's algorithm assumes that the maximum resource requirement at any given time by a process in the system is known in advance. The resource demand of a process is granted if the system remains in a "safe" state. In other words, Banker's algorithm does a forward calculation keeping in account the pending demands of the present process set, such that no deadlock would occur.

The Deadlock Prevention and Deadlock Avoidance strategies, lead to lower device utilization and throughput. Authors [8, 9] suggested Deadlock Recovery technique wherein corrective

measures are taken only when the deadlock actually occurs. System constantly performs a self Deadlock Detection Test to ensure that it is deadlock free. Deadlock Recovery technique is most efficient till a deadlock does not occur. When a deadlock actually occurs the system is in blocked state as no process is capable of executing. The system throughput and resource utilization tends to zero. Deadlock Recovery technique strategically does a forceful preemption of the resources or partial/complete termination of processes executing. The system in recovery phase does not respond to user requests for some finite amount of time.

This paper presents a Total Need based Resource Reservation (TNRR) Technique for effective resource management. It proposes to reserve some instances of a resource based on the total demand placed (i.e., need) for it by all the processes in the system. The remaining resources can be allocated to any process requesting them as in Deadlock Recovery technique. When a process requests for resource instances available only in the reserve pool then the proposed technique will grant it only if its total need can be satisfied. In other words, the reserved pool of resource instances is allocated to only those processes that are likely to complete. The proposed technique reduces the overhead incurred by the Deadlock Avoidance. But it does not guarantee that a deadlock will never occur. On the other hand, as compared to the Deadlock Recovery strategy, it reduces the frequency of deadlock occurrence in the system. The simulation results indicate that the system performance is improved with respect of both the frequency of deadlocks as well as in terms of turnaround time as the overhead incurred during both the Deadlock Avoidance and Recovery is reduced.

The rest of the paper is organized as follows: section 2, illustrates a motivational example while section 3 describes the system model. Section 4 elaborates the proposed approach followed by results and analysis in section 5. Finally, paper concludes with section 6.

## 2. MOTIVATIONAL EXAMPLE

This section presents a motivational example which will illustrate the limitations of existing techniques.

***Example [9, 13]:*** Consider a system consisting of five processes $P_1$ through $P_5$ and three resource types $R_1, R_2$ and $R_3$ with instances 10, 5 and 7 respectively. This is illustrated in the table 1. Where Execution Time is the worst case execution time that a process requires to complete its execution;
Allocation $R_j$, represents the number of instances of a resource type $R_j$ allocated to a process $P_i$;
Maximum $R_j$, represents the at most demand for resource type $R_j$ by process $P_i$ during its entire execution;
Request $R_j$, represents the demand for resource type $R_j$ by process $P_i$ when it starts execution;
Available $R_j$, is the number of resource type $R_j$ available in the system at any time.

Thus, after allocation of the requested resources, suppose, at time $t_1$, the snapshot of the system has been taken as given in table 2, where Need $R_j$, represents the remaining need for resource type $R_j$ by the process $P_i$;
Suppose at time $t_1$ a process $P_2$ request for one additional resource type $R_1$, and two instances of resource type $R_3$, i.e., (1, 0, 2), the decision that whether this request can be granted immediately by the existing techniques is done as follows:

**Table 1: Snapshot of the system at the start time $t_0$**

| | Execution Time | Allocation | | | Maximum | | | Request | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $R_1$ | $R_2$ | $R_3$ | $R_1$ | $R_2$ | $R_3$ | $R_1$ | $R_2$ | $R_3$ | $R_1$ | $R_2$ | $R_3$ |
| $P_1$ | 2 | 0 | 0 | 0 | 7 | 5 | 3 | 0 | 1 | 0 | 10 | 5 | 7 |
| $P_2$ | 10 | 0 | 0 | 0 | 3 | 2 | 2 | 2 | 0 | 0 | | | |
| $P_3$ | 11 | 0 | 0 | 0 | 9 | 0 | 2 | 3 | 0 | 2 | | | |
| $P_4$ | 12 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 1 | 1 | | | |
| $P_5$ | 24 | 0 | 0 | 0 | 4 | 3 | 3 | 0 | 0 | 2 | | | |
| Total_Need[j]= $\sum_{i=1}^{n} Max[i][j]$ | | | | | 25 | 12 | 12 | | | | | | |

**Table 2:Snapshot of the system at time $t_1$**

| | Allocation | | | Maximum | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $R_1$ | $R_2$ | $R_3$ | $R_1$ | $R_2$ | $R_3$ | $R_1$ | $R_2$ | $R_3$ | $R_1$ | $R_2$ | $R_3$ |
| $P_1$ | 0 | 1 | 0 | 7 | 5 | 3 | 7 | 4 | 3 | 3 | 3 | 2 |
| $P_2$ | 2 | 0 | 0 | 3 | 2 | 2 | 1 | 2 | 2 | | | |
| $P_3$ | 3 | 0 | 2 | 9 | 0 | 2 | 6 | 0 | 0 | | | |
| $P_4$ | 2 | 1 | 1 | 2 | 2 | 2 | 0 | 1 | 1 | | | |
| $P_5$ | 0 | 0 | 2 | 4 | 3 | 3 | 4 | 3 | 1 | | | |

**Deadlock Avoidance (Banker's Algorithm) *[4, 5, 9, 19, 20]*:** At time $t_1$, when $P_2$ request for (1, 0, 2) resources, then the Banker's algorithm will calculate the safety sequence, i.e., a sequence with worst case resource allocation that will lead to completion of all processes. Thus, in case the requested resources are granted the snapshot of the system can be seen in the table 3a. Banker's algorithm will estimate the safety sequence (refer section 3) as $\langle P_2, P_4, P_5, P_1, P_3 \rangle$. Thus, it will allocate the requested resources to process $P_2$ at time $t_2$, causing an overhead of $t_2 - t_1$. Suppose at time $t_3$, process $P_1$ request for (0, 2, 0) resources then the snapshot can be seen in the table 3b. Banker's algorithm will again estimate the safety sequence causing overhead. However, this time it is unable to find a single process whose need is less than available, hence, no process will be able to complete itself as the need for all the process as can be seen in the table 3b is greater than the available. Thus, the Banker's algorithm declines this request.

**Deadlock Recovery:** Deadlock Recovery technique does not do any pretesting, it will simply grant the requests as and when made by processes $P_2$ and $P_1$. However, immediately the system may not be in deadlock state, but over time all the processes will eventually ask for the resources mentioned in their need column of table 3b without releasing any resources up to their completion. Thus, a deadlock will eventually occur. Hence, the preemption of the resources or process needs to be done causing an overhead and degraded performance.

The motivational example clearly demonstrate that the above approaches either perform rigorous testing or no testing leading to no or frequent deadlocks. This paper strives to balance between the two and suggest an approach which will produce higher performance by performing a lower cost test. The following section presents the assumption and the terminologies used.

## 3. SYSTEM MODEL

This paper deals with resources allocation technique that allocates the resources to the requesting processes. The system is assumed to have $m$ resource types, i.e., $R_1,\ R_2,\ R_3 \ldots R_m$, with $\alpha_1,\ \alpha_2,\ \alpha_3 \ldots \alpha_m$ instances of each type. Further, it consists of $n$ independent processes $P_1,\ P_2,\ P_3 \ldots P_n$ where, each process $P_i$ has the attributes$(a_i,\ e_i)$, that is the arrival and worst-case execution time respectively. The processes are scheduled using Round Robin scheduling [9, 10, 11]. The following data structures are used for maintaining the state of the resource allocation in the system:

- ***Available:*** An array of $m$ elements, indicating the number of instances available for each resource type. Thus, Available$(R_j)$, is the number of resource type $R_j$ available in the system.
- ***Maximum:*** A two dimensional array $n \times m$, defining the maximum resource demand of each process. If Max[i][j] equal k, then process $P_i$ may request at most k instances of resources type $R_j$ in its life time.
- ***Allocation:*** A two dimensional array $n \times m$, defines the number of resources of each type currently allocated to each process. If Allocation[i][j] equals k, then process $P_i$ is currently allocated k instances of resources type $R_j$.
- ***Need:*** A two dimensional array $n \times m$, indicates the remaining resource need of each process. If Need[i][j] equals k, then process $P_i$ may need k more instances of resources type $R_j$ to complete its execution. It can estimated as $Need[i][j] = Maximum[i][j] - Allocation[i][j]$.
- ***Total_Need:*** An array of $m$ elements, indicating the total maximum resource requirement for all the processes. If Total_Need[j] equals k, then sum of the maximum requirement of any resource by all the processes is k. Mathematically, $Total\_Need[j] = \sum_{i=1}^{n} Max[i][j]$
- ***Request:*** A two dimensional array $n \times m$, indicating the number of resource requested by process $P_i$ during its execution. If Request[i][j] equals k, then process $P_i$ request k instances of resource type $R_j$ for current execution.
- ***Reserve:*** A two dimensional array $m$, where Reserve[j] equals $k$ indicate that $k$ instance of the resource of type $R_j$ are reserved.

***Average Turnaround Time:*** is the difference in time between the submission of a process to its completion.

***Safe State [8, 9, 18]:*** The system is said to be in Safe State, if allocation to each process can be made in some order (Safety Sequence) and still avoid a deadlock.

***Safety Sequence [9]:*** A sequence of process $\langle P_1,\ P_2,\ P_3 \ldots P_n \rangle$ is safe sequence for the current allocation state if, for each $P_i$, the resource requests that $P_i$ can still make can be satisfied by the currently available resources plus resources held by all $P_j$, $j < i$. It can be estimated as suggested in the Banker's Algorithm as follows:

**Table 3: a)Snapshot of the system, after the probable allocation to $P_2$ (1, 0, 2)**

| | Allocation | | | Maximum | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $R_1$ | $R_2$ | $R_3$ | $R_1$ | $R_2$ | $R_3$ | $R_1$ | $R_2$ | $R_3$ | $R_1$ | $R_2$ | $R_3$ |
| $P_1$ | 0 | 1 | 0 | 7 | 5 | 3 | 7 | 4 | 3 | 2 | 3 | 0 |
| $P_2$ | 3 | 0 | 2 | 3 | 2 | 2 | 0 | 2 | 0 | | | |
| $P_3$ | 3 | 0 | 2 | 9 | 0 | 2 | 6 | 0 | 0 | | | |
| $P_4$ | 2 | 1 | 1 | 2 | 2 | 2 | 0 | 1 | 1 | | | |
| $P_5$ | 0 | 0 | 2 | 4 | 3 | 3 | 4 | 3 | 1 | | | |

Request Granted by Banker's Algorithm and safety sequence is $\langle P_2, P_4, P_5, P_1, P_3 \rangle$

**b) Snapshot of the system, after the probable allocation to $P_1$ (0, 2, 0)**

| | Allocation | | | Maximum | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $R_1$ | $R_2$ | $R_3$ | $R_1$ | $R_1$ | $R_2$ | $R_3$ | $R_1$ | $R_1$ | $R_2$ | $R_3$ | $R_1$ |
| $P_1$ | 0 | 3 | 0 | 7 | 5 | 3 | 7 | 2 | 3 | 2 | 1 | 0 |
| $P_2$ | 3 | 0 | 2 | 3 | 2 | 2 | 0 | 2 | 0 | | | |
| $P_3$ | 3 | 0 | 2 | 9 | 0 | 2 | 6 | 0 | 0 | | | |
| $P_4$ | 2 | 1 | 1 | 2 | 2 | 2 | 0 | 1 | 1 | | | |
| $P_5$ | 0 | 0 | 2 | 4 | 3 | 3 | 4 | 3 | 1 | | | |

Request Not Granted by Banker's Algorithm, since no safety sequence exist

***Safety Sequence Algorithm [9]:***
***Begin***

1. ***Let Work and Finish be vectors of length m and n respectively. Initialize Work=Available and Finish[i]=false for i=1,2, … n***
   *// Finish indicates if a process has completed or not*
2. ***For x= 1 to n***
   ***do***
   ***a. For i= 1 to n***
   ***do***
   ***i. If (Finish[i] = true) then goto step 2 a.***
   ***ii. For j= 1 to m***
   ***do***
   ***1. If (Need[i][j] ≤ Work[jj]) then***
   ***a. Can_exe=1;***
   ***Else***
   ***b. Can_exe=0;***
   ***c. Goto step 2 a.***
   ***End for***

   ***iii. For j= 1 to m***
   ***do***
   ***1. Work[j]=Work[j] + Allocation[i][j]***
   ***2. Finish[i]=true;***
   ***3. Goto step 2***
   ***End for***
   ***b. If (Finish[x] =true) for all x, then system is in safe state***
   ***End for***

***End***

The following section illustrates the proposed resource reservation technique for effective utilization of the resources.

# 4. PROPOSED TOTAL NEED BASED RESOURCE RESERVATION (TNRR) TECHNIQUE

The motivational example in section 2, demonstrate that Banker's Algorithm will always estimate the safety sequence by considering the requirement for all the processes in the system. However, this cross checking incurs as overhead $O(mn^2)$.

This paper proposes the resource reservation techniques in which the system reserves a pool of resources. These reserved resources can be allocated to only those processes those total resources need can be satisfied by them, i.e., a process who will not require any resources further and will finish its execution. The remaining resources are freely available and can be allocated to any requesting process. Thus, when a process demands for resources which are not freely available then the system must release the reserve pool resources provided the total need of the process can be satisfied. Hence, it will complete and relieve all the resources it is holding.

The proposed resource reservation technique will reserve the resources based on total need of resources requested by processes seen so far. Mathematically, $Reserve[j] = \lceil (Total\_Need - \alpha_j)/n \rceil$ where $Reserve[j]$ is the number of instances of a resource type $R_j$ reserved. This technique will only considers the requesting process detail along with the system data structure to take the decision for granting or not granting of the resources incurring an overhead of mere $O(m)$. The proposed Total Need based Resource Reservation (TNRR) technique can be summarized as follows:

***TNRR Algorithm:***
***Input: Process Priority Queue***
***begin***

1. ***Initially***
   a. $\boldsymbol{Need = Maximum}$ */*whenever a process arrives it has no resource allocated to itself hence, the need is same as the maximum*/*
   b. $\boldsymbol{Reserve[j] = \lceil (Total\_Need - \alpha_j)/n \rceil}$
   c. $\boldsymbol{Available[j] = \alpha_j - Reserve[j]\ \forall j = 1, 2 \ldots m}$
   d. ***Allocation=0***

2. ***Till no request***
   *a.* ***Wait*** *// Do nothing*

3. ***If an $i^{th}$ process requests for "Request[i][j]", a vector of size m for each resource then***
   a. ***If*** $(\boldsymbol{Request[i][j] \le Available[j]}) \forall j = 1, 2 \ldots m$ ***then***
      1. ***Allocation[i][j]= Allocation[i][j] + Request[i][j]*** $\forall j = 1, 2 \ldots m$
         *// grant the request*
      2. ***Available [j]=Available [j] – Request[i][j]*** $\forall j = 1, 2 \ldots m$
         *// since the resources are allocated hence, they are no*
         *// more available*
      3. ***Need[i][j]=Maximum[i][j]– Allocation[i][j]*** $\forall j = 1, 2 \ldots m$

      ***Else***
   b. ***If*** $\left(\boldsymbol{Available[j] + Reserve[j] \ge Need[i][j]} \quad \forall j = 1, 2 \ldots m\right)$
      ***then***
      1. ***Allocation[i][j]= Allocation[i][j] + Need[i][j]*** $\forall j = 1, 2 \ldots m$
         */* grant the request by allocating maximum number of resource it may need so that this process will definitely complete */*
      2. ***Available [j]=Available [j] –Need[i][j]*** $\forall j = 1, 2 \ldots m$
         *// since the resources are allocated hence, they are no*
         *// more available*
         a. ***Need[i][j]=0*** $\forall j = 1, 2 \ldots m$

      ***Else***
         *//decline the request*
         a. ***Goto step 2***
   ***End for***

4. ***If a process $P_i$ completes then***
   a. ***For j= 1 to m***
      ***do***
      i. ***Available [i][j] = Available[i][j] + Allocation[i][j]***
      ***End for***
   b. ***Remove $P_i$ from the queue, goto step 2.***

***End***

The effectiveness of proposed Total Need based Resource Reservation (TNRR) Technique can be seen by the motivational example in section 2. Here, for the same example resource allocation is done using the proposed approach. Thus, considering the system snapshot as illustrated in table 1. The Total_Need can be estimated as 25, 12, 12 for the resources $R_1, R_2$ and $R_3$ respectively. The number of instance of $R_1, R_2$ and $R_3$ resources reserved by the system are thus, (3, 2, 1) respectively leaving the available resources as (7, 3, 6).

At time $t_0$, the request made by all the processes is also indicated in the request column of the table 1. Since, the request made by all the processes is not more than the available resources in the system, they all are granted and the snapshot of the system can be seen in the table 2. However, the available resources will only be (0, 1, 1) while those reserved are still unused and (3, 2, 1).

At time $t_1$, when $P_2$ request for (1, 0, 2) resources. The resources available are (0, 1, 1), which are insufficient to cadre the request (step 3a. of the proposed TNRR algorithm). However, the system has a reserve pool of (3, 2, 1). The proposed TNRR algorithm releases the reserved pool resources, only to a process that promises to complete and return the reserved resources. Thus, instead to granting the requested amount of (1, 0, 2) TNRR will grant the total needed resources by $P_2$, i.e., (1, 2, 2) and ensure that it will complete. The snapshot thus, attained can be seen in the table 4. If the safety sequence test is performed on table 4, the safety sequence turns out as $\langle P_2, P_4, P_1, P_3, P_5 \rangle$. This safety sequence check is not required by the proposed TNRR algorithm. However, it is used to demonstrate that the system is in safe state. It may be noted that as per the Banker's algorithm the average turnaround time as per the safety sequence of $\langle P_2, P_4, P_5, P_1, P_3 \rangle$ (estimated in section 2) will be 38.6 while that of the proposed TNRR algorithm is 30 which is approximately 22.3% lower.

**Table 4: Snapshot of the system, after the allocation to $P_2$ (1, 2, 2)**

| | Allocation | | | Maximum | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $R_1$ | $R_2$ | $R_3$ | $R_1$ | $R_2$ | $R_3$ | $R_1$ | $R_2$ | $R_3$ | $R_1$ | $R_2$ | $R_3$ |
| $P_1$ | 0 | 1 | 0 | 7 | 5 | 3 | 7 | 4 | 3 | 0 | 0 | 0 |
| $P_2$ | 3 | 2 | 2 | 3 | 2 | 2 | 0 | 0 | 0 | | | |
| $P_3$ | 3 | 0 | 2 | 9 | 0 | 2 | 6 | 0 | 0 | Reserved | | |
| $P_4$ | 2 | 1 | 1 | 2 | 2 | 2 | 0 | 1 | 1 | $R_1$ | $R_2$ | $R_3$ |
| $P_5$ | 0 | 0 | 2 | 4 | 3 | 3 | 4 | 3 | 1 | 2 | 1 | 0 |

At time $t_3$, process $P_1$ request for (0, 2, 0) resources, by this time the process $P_2$ may or may not have completed. This lead to following two cases:
Case1: If process $P_2$ hasn't completed then as can be seen from the table 4, no resources are in the available pool. The resources in the reserved pool can be used only and only when a processes total need can be satisfied. The process $P_1$'s need as indicated in the table 4 is (7, 4, 3) while reserve pool contains only (2, 1, 0) resources. Hence, no resource can be allocated to process $P_1$.

**Table 5: Snapshot of the system, after the completion of process $P_2$**

| | Allocation | | | Maximum | | | Need | | | Available | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $R_1$ | $R_2$ | $R_3$ | $R_1$ | $R_2$ | $R_3$ | $R_1$ | $R_2$ | $R_3$ | $R_1$ | $R_2$ | $R_3$ |
| $P_1$ | 0 | 1 | 0 | 7 | 5 | 3 | 7 | 4 | 3 | 2 | 1 | 1 |
| $P_2$ | Completed | | | | | | | | | | | |
| $P_3$ | 3 | 0 | 2 | 9 | 0 | 2 | 6 | 0 | 0 | Reserved | | |
| $P_4$ | 2 | 1 | 1 | 2 | 2 | 2 | 0 | 1 | 1 | $R_1$ | $R_2$ | $R_3$ |
| $P_5$ | 0 | 0 | 2 | 4 | 3 | 3 | 4 | 3 | 1 | 3 | 2 | 1 |

Case2: If process $P_2$ has completed and relinquishes the resources. This scenario can be seen in the table 5. Still, the resources in the available pool are (2, 1, 1) when request is for (0, 2, 0) resources. The available pool and reserve pool put together has (5, 3, 2) resources while the process $P_1$'s need as many as (7, 4, 3) resources. Thus, the request by $P_1$cannot be granted as number of resources are not sufficient.

The proposed TNRR algorithm does not allocate any resource to $P_1$ under any circumstances. Thus, the system remains in the safe state and the safety sequence also remains unaltered.
The following section present the results obtained by implementation of the technique discussed in this section.

## 5. SIMULATIONS RESULTS

In this section simulation on synthesized process sets is performed to evaluate the performance of the proposed Total Need based Resource Reservation (TNRR) technique. Comparison is done with the Banker's Algorithm (BA) and Deadlock Recovery (DR). The key parameters used for comparison are the *Frequency of Deadlock and Average Turnaround time*, defined as follows:
The Average Turnaround Time is the difference in time between the submissions of a process to its completion.
The Frequency of Deadlock is the frequency of the deadlocks occurring in the system.

Processes where generated by an exponential distribution using with inter arrival time (1/λ) and service time (1/μ) with parameters λ and μ, simulation is run for 1000 processes. The resources are also picked from a pool, randomly generated at the beginning.

The effect of increase in the process load over the frequency of the deadlock can be seen in the figure 1. The Banker's algorithm is complete Deadlock Avoidance approach, wherein it ensures that system is always in safe state, hence, no deadlock ever occurs. The Deadlock Recovery technique simply grants the resources requested, hence, as the process load increases the frequency of deadlock also increases. The proposed technique TNRR performs resource reservation to ensure that at least one process will always complete. However, the resources released by a process on its completion may not be sufficient for the remaining processes and a deadlock may occur. It may be noted from the figure that the frequency of deadlock has decreased approximately by 75%. Further, the system is able to survive for longer time without deadlocks.

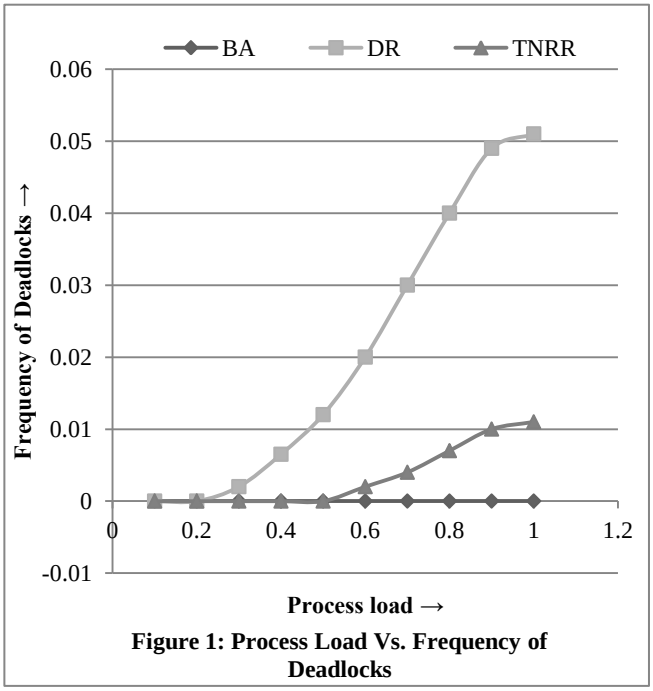


**Figure 1: Process Load Vs. Frequency of Deadlocks**

Figure 2, depicts the effect of increase in the process load over the average turnaround time. The average turnaround time increases for all the techniques as the load increases. This is because, more loads leads to higher contention for the resources and more frequent deadlocks. However, the performance of the proposed TNRR is approximately 9% better for all ranges, especially for higher ranges, because the overhead involved for resource allocation is much lower than that of the Banker's algorithm and hence, it is able to handle more processes efficiently. Thus, each process completes earlier, relinquishing the resources, leading to lower average turnaround time.

## 6. CONCLUSION

The existing techniques either perform costly tests (both with respect to time as well as space) or do no test at all. This imply that a deadlock will never occur or will occur very frequently. This paper presents intermediate solution by not performing the test but still reserve some resources so as to ensure that at least one process always has the requisite number of resources to complete. The proposed technique is referred to as Total Need based Resource Reservation (TNRR).

The proposed TNRR technique maintains the resource needed by all the processes seen so far in a data structure. The allocation to the requesting process is made based on it. The motivational example illustrate that without performing the safety sequence check, the proposed algorithm is capable of taking a decision for

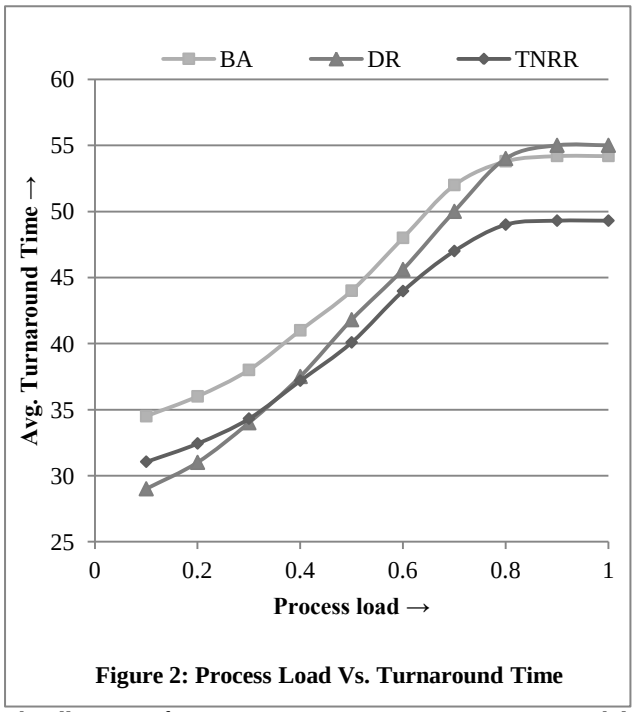


**Figure 2: Process Load Vs. Turnaround Time**

the allocation of resource instances to a requesting process while maintaining the system in safe state. The overhead incurred for this decision is mere $O(m)$ for the proposed TNRR as compared to existing Banker's algorithm of the $O(mn^2)$. The simulation results indicate that the the system is able to survive for longer time without deadlocks. Further, the turnaround time of the TNRR is approximately 9% better than the existing Banker's algorithm.

## 7. REFERENCES

[1.] Goswami, Vaisla and Ajit Singh, "VGS Algorithm: An Efficient Deadlock Prevention Mechanism for Distributed Transactions using Pipeline Method" International Journal of Computer Applications (0975 – 8887) Volume 46– No.22, May 2012

[2.] U. Kapasi, W. Dally, S. Rixner, J. Owens, and B. Khailany, "The Imagine stream processor", Proc. International Conference of Computer Design, 2002, 282–288.

[3.] D. Zobel, "The Deadlock problem: a classifying bibliography", ACM SIGOPS Operating Systems Review, vol. 17, October 1983.

[4.] Sheau-Dong Lang, "An Extended Banker's Algorithm for Deadlock Avoidance", IEEE Transactions On Software Engineering, VOL. 25, NO. 3, MAY/JUNE 1999

[5.] E.W. Dijkstra, "Cooperating Sequential Processes," Programming Languages, F. Genuys, ed., pp. 103-110, New York: Academic Press, 1968.

[6.] A. N. Habermann, "Prevention of System Deadlocks," Comm. ACM, vol. 12, no. 7, pp. 373-377, 385, July 1969.

[7.] R.C. Holt, "Some Deadlock Properties of Computer Systems", ACM Computing Surveys, vol. 4, no. 3, pp. 179-196, Sept. 1972.

[8.] William Stallings, "Operating Systems: Internal and Design Principles", Fifth Edition, Pearson Publications, 2008.

[9.] A. Silberschatz, P. B. Galvin and G. Gagne, "Operating System Principle", Seventh Edition, Wiley India.

[10.] N. Ramasubramanian, Srinivas V.V., Chaitanya V, " Studies on Performance Aspects of Scheduling Algorithms on Multicore Platforms," International Journal of Advanced Research in Computer Science and Software Engineering, Vol 2, Issue 2, February 2012.

[11.] H. S. Behera, Ratikanta Pattanayak, Priyabrata Mallick, " An Improved Fuzzy-Based CPU Scheduling (IFCS) Algorithm for Real Time Systems," International Journal of Soft Computing and Engineering (IJSCE) ( 2231-2307), Volume-2, Issue-1, March 2012

[12.] Saroj Hiranwal, Dr. K.C.Roy, "Adaptive Round Robin Scheduling using Shortest Burst Approach Based on Smart Time Slice," International Journal of Data Engineering (IJDE), Volume 2, Issue 3 2012.

[13.] B Madhavi Devi, Smriti Agrawal, Ch. Srinivasulu, "An Efficient Resource Allocation Technique for Uni-Processor System" International Journal of Advances in Engineering & Technology (IJAET) Volume 6 Issue 1, March 1, 2013.

[14.] W. Lin and D. Qi, "Research on Resource Self-Organizing Model for Cloud Computing," in Proc. IEEE International Conference on Internet Technology and Applications, pp. 1–5, 2010.

[15.] H. Shi and Z. Zhan, "An optimal infrastructure design method of cloud computing services from the BDIM perspective," in Proc. IEEE Computational Intelligence and Industrial Applications, vol. 1, pp. 393–396, 2009.

[16.] X. Nan, Y. He, and L. Guan, "Optimal resource allocation for multimedia cloud based on queuing model," in IEEE MMSP. pp. 1–6, Oct. 2010.

[17.] Nan, Xiaoming," Optimal resource allocation for multimedia cloud in priority service scheme ", in IEEE International Symposium on Circuits and Systems (ISCAS), 2012.

[18.] Finkel and Madduri, "An Efficient Deadlock Avoidance Algorithm", Information Processing Letters 24 (1987) 25-30 North-Holland 15 January 1987.

[19.] Ewa KLUPSZ, "A Linear algorithm of a deadlock avoidance for nonpreemptible resources," Information Processing Letters 19 (1984) 87-94

[20.] Wojciech Cellary, "A New Safety Test For Deadlock Avoidance," Information Processing Letters, Volume 8, number 8 , March 1979

[21.] Micha Hofri, "On timeout for global deadlock detection in decentralized database systems", Information Processing Letters 51 (1994) 295-302

[22.] Yee Ming Chen1 Shin-Ying Tsai, Optimal Provisioning of Resource in a Cloud Service," IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 6, November 2010 ISSN (Online): 1694-0814