Optimization of Banker Algorithm

M. D. Chavan[1] A. R. Nistande[2]

Abstract— In this paper we have implemented the optimization of banker algorithm with the help of some good searching and sorting technique such as merge sort, hashing mapping and so on. This technique has a very higher potential to best utilization of resources in our computer in quadratic time. We have used greedy approach to optimize this algorithm. This algorithm is not only used for banking but also useful for many organizations such as library and many more.

Key words: Greedy Approach, Banker Algorithm

I. INTRODUCTION

Banker algorithm is widely used into industry specially into banking sector. It also has lot of significant coming to real world entity and operating system. This algorithm works only when processes claim their maximum need in advance.[1] When all requests mead by processes in advance, algorithm grants the resources if system remain in safe state. Even in worst scenario if process mead their maximum need and if their exist some schedule to grant the resources without going in unsafe state, then the resources will be granted. When their n process and m process, the time require to complete it execution will be O (n^2m).[1]-[2]

As efficiency of algorithm is major concern in practice, another important concern is effective utilization of resources and time require by processor for execution. As we know the maximum need of resource in advance, it is possible that request will be denied if system goes into [2] unsafe state.

A system going into a unsafe state mean, a set of process is said be in deadlock when every process in the set waits for an event that can cause by another process in set. We have four necessary conditions to happen deadlock which are mutual exclusion, hold and wait, circular wait and no pre-emption. If we somehow disable one of above four conditions, then we can definitely [3] say that system is safe state or deadlock free. But this is not true always.

As disabling one of the necessary condition is not possible all the time, hence we go for some strategies to handle deadlock. We have mainly four strategies,

- Deadlock ignorance
- Deadlock prevention
- Deadlock avoidance
- Deadlock detection and recovery

In deadlock ignorance approach, we completely ignore deadlock as if it will never happen and sometimes called as ostrich approach. We disable at least one of the necessary condition of deadlock in deadlock prevention strategy. We have banker algorithm [3]-[4] in deadlock avoidance strategy which will work only when we know the requirement of process in advance.

Data structure used in this algorithm is vector array; following are some of the data structure used.

- Let n = number of processes, and m = number of resources types.
- Available: Vector of length m. If available [j] = k, there are k instances of resource type R_j available
- Max: n x m matrix. If Max [i,j] = k, then process P_i may request at most k instances of resource type R_j
- Allocation: n x m matrix. If Allocation[i,j] = k then P_i is currently allocated k instances of R_j
- Need: n x m matrix. If Need [i,j] = k, then P_i may need k more instances of R_j to complete its task.

  Need [i,j] = Max[i,j] – Allocation [i,j]
- Request = An n x m matrix indicates the current request of each process. If Request [i][j] = k, then process P_i is requesting k more instances of resource type R_j. [4]

II. LITERATURE SURVEY

Various scientist has propose various solution to overcome problem of deadlock such as ensuring that the system will never enter a deadlock state called as deadlock prevention approach, Allow the system to enter a deadlock state and then recover called as deadlock detection and recovery, ignore the problem and pretend that deadlocks never occur in the system called as [6] Ostrich approach. This all requires has some additional a priori information available.

Deadlock Avoidance is simplest and most useful approach. It requires that each process declare the maximum number of resources of each type that shall be required during the execution of process. The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition. Resource-allocation state is defined by the number of available and allocated [7] resources, and the maximum demands of the processes.

We have mainly two strategies to optimize any algorithm, which are nothing but the greedy and dynamic programming respectively. In greedy algorithm we are greedy about something in same way human being.[7]-[8]

We have some necessary condition to apply dynamic programming algorithm such as optimal substructure, overlapping and recursive equation. Optimal substructures mean dividing problem into small problem in such a way that it cannot divide furthermore. It must be smallest and solution to small problem must be solution for bigger problem. Overlapping manifest that there must be repetition of some optimal substructure as we don't need it again and again. Recursive equation means by aggregating optimal substructure and overlapping, we must be in stand to form some equation and transform them into algorithm. Last point is that there should not dependency [8] between optimal substructure.

In our case we can form optimal substructure, there would be overlapping but there is dependency between optimal substructures, as we are calculating need and available matrix depending upon various factor/attribute of process. Therefore dynamic programming methodology cannot be used to optimize our algorithm.

Greedy strategy is used, in greedy approach we have to greedy about something and in our case is time. In proposed work I don't give the requirement of process i.e. input to algorithm in the same way they come, before giving it to algorithm. I mead process to go through some filter

which will organized process in ascending order depending upon their need.[9]

At entry section, I used hashmap sort library function which will internally call system call and will return sorted order of requirement of process depending upon their requirement. Time required by hashmap sort is O(nlogn) and hence if we do aggregate analysis the total time required for execution of banker [9]-[10]algorithm is O(n)+O(nlogn) is O(nlogn).

III. PROPOSED WORK

Our previous algorithm work as given below. It divides the process into two step, resource request allocation and safety algorithm. In resource allocation step. It pretends as if resources are allocated, but it never was allocation. It is doing future analysis in safety algorithm that; if we allocate resources to process it should not go into deadlock. If, after allocating resources to the process, the [11] system is going into the deadlock state then they declare it as unsafe state and resources could not be allocated.

A. Resource-Request Algorithm for Process P_i

Request = request vector for process P_i. If $Request_i$ [j] = k then process P_i wants k instances of resource type R_j

- If $Request_i \leq Need_i$ go [12] to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim
- If $Request_i \leq Available$, go to step 3. Otherwise P_i must wait, since resources are not available
- Pretend to allocate requested resources to P_i by modifying the state as follows:

$$Available = Available – Request;$$
$$Allocation_i = Allocation + Request_i;$$
$$Need_i = Need_i – Request_i;$$

If safe ⇒ the resources are allocated to Pi
If unsafe ⇒ Pi must wait, and the old resource-allocation state is restored

1) Safety Algorithm:

- Let Work and Finish be vectors of length m and n, respectively. Initialize:

Work = Available
Finish [i] = false for i = 0, 1, …, n- 1

- Find an i such that both:

Finish [i] = false
$Need_i \leq Work$
If no such i exists, go to step 4

- Work=Work+$Allocation_i$
  Finish[i]=true
  go to step 2
- If Finish [i] == true for all i, then the system is in a safe state.

The time complexity require to execute above algorithm is $O(n^2d)$. Where 'n' is the number of resources and'd' be a time require to allocate resources to process. In the proposed system, I had optimize the complexity of algorithm to O(nlogn) as shown below.

B. Propose Work

- Step 1: Get the requirement of process
- Step 2: Put them it into our new data structure which is tree map.
- Step 3: Sort the process using hashmap sort, depending upon the sum we get, after adding all element array.
- Step 4: give these all process to Banker algorithm.
- Step 5: If need of first process is valid then proceed furthermore, otherwise display error.{ conditional checking }
- Step 6: If need of the first process could not satisfy then Declare deadlock otherwise proceed furthermore.
- Step 7: If all need of the process get satisfied and every Element in Boolean array become zero or false then system will go into safe state otherwise System is unsafe.

IV. PERFORMANCE ANALYSIS

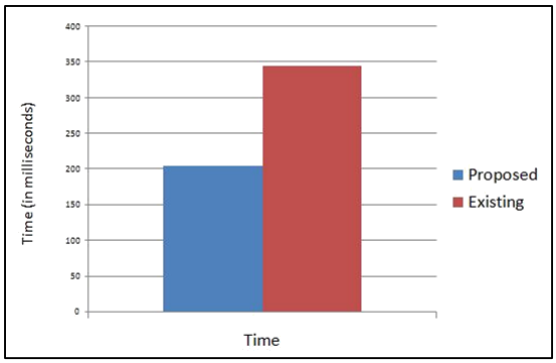

Fig. 1: Performance Analysis

Above graph illustrate time required by proposed and existing system. Data set given for both the system is same which nothing is but some raw data separated by comma, also called as comma separated value (CSV). If we take a set of input consist of 100 process and resources which, when given to a proposed and existing system simultaneously then proposed system take a time of 3384msec and existing system take a time of 5042msec . Time could also vary depending upon sequence of process which comes before hashmap sort. As we are not sorting process in existing system, safe sequence could also change as given in proposed system.

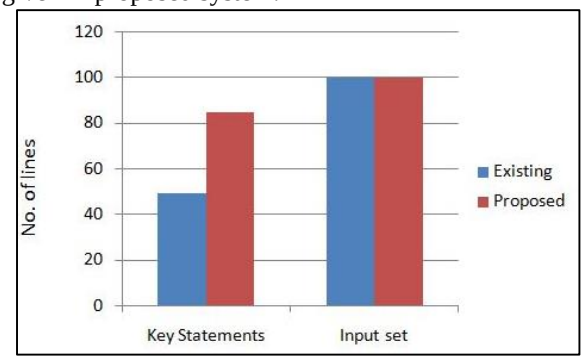

Fig. 2: Performance Analysis

We also have some performance measure coming to number of key statement executed in proposed and existing system. We are imposing some filter before process given to proposed system, which is hashmap sort. Hence number of key statement increases in proposed system as compare to existing system.

V. CONCLUSION

The Banker's Algorithm for Deadlock Avoidance has no systematic approach for those processes which are in

waiting state. We proposed that if process is going to waiting state then the consideration of number of allocated resources and/or number of instances as well as need of resources in order to select a waiting process for the execution will make Banker's Algorithm more efficient. As the number of process into system increases, the system becomes more and more efficient. Conventional safety algorithm used to produce the safe sequence of the processes so that the system will not stuck in the deadlock. Safe sequence cannot be calculated using dynamic programming, as there are n! Ways to find safe sequence.

REFERENCES

[1] Silberschatz/Galvin/Gagne Sixth Edition "Operating System Concepts".
[2] Dijkstra, Edsger W. The mathematics behind the Banker's Algorithm (EWD-623). E.W. Dijkstra Archive. Center for American History, University of Texas at Austin.
[3] http://en.wikipedia.org/wiki/Banker%27s_algorithm.
[4] R.C. Holt, "Some Deadlock Properties of Computer Systems,"ACM Computing Surveys, vol. 4, no. 3, pp. 179-196, Sept. 1972.
[5] published as pages 308–312 of Edsger W. Dijkstra, Selected Writings on Computing: A Personal Perspective, Springer-Verlag, 1982. ISBN 0-387-90652-5
[6] A.N. Habermann, "Prevention of System Deadlocks," Comm.ACM, vol. 12, no. 7, pp. 373-377, 385, July 1969.
[7] R. Finkel and H.H. Madduri, "An Efficient Deadlock Avoidance Algorithm," Information Processing Letters, vol. 24, no. 1, pp. 25-30,Jan. 1987.
[8] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, The Design and Analysis of Computer Algorithms. Reading, Mass.: Addison-Wesley, 1974.
[9] E. Horowitz, S. Sahni and S. Rajasekaran, "Computer Algorithms/C++," second edition, ISBN: 81 7371 611 0, 2008.
[10] E.W. Dijkstra, "Cooperating Sequential Processes," ProgrammingLanguages, F. Genuys, ed., pp. 103-110, New York: Academic Press,1968.
[11] T. Minoura, "Deadlock Avoidance Revisited," J. ACM, vol. 29, no. 4,pp. 1,023-1,048, Oct. 1982.
[12] S.-D. Lang, "An Extended Banker's Algorithm for Deadlock Avoidance," IEEE Trans. Software Eng., vol. 25, no. 3, pp. 428-432, May/June 1999.
[13] Y. Wang and P. Lu, "DDS: A Deadlock Detection-Based Scheduling Algorithm for Workflow Computations in HPC Systems with Storage Constraints," Parallel Computing, vol. 39, no. 8, pp. 291-305,Aug. 2013.
[14] Ferenc Belike, "An efficient Deadlock Avoidance Technique," IEEETrans. ON COMPUTERS**,** VOL. 39, NO. 7, pp. 882-888, JULY 1990.
[15] Y. Wang and P. Lu, "Dataflow Detection and Applications to Workflow Scheduling," Concurrency and Computation: Practice and Experience, vol. 23, no. 11, pp. 1261-1283, 2011.
[16] S. Reveliotis and M.A. Lawley, "Efficient Implementations of Banker's Algorithm for Deadlock Avoidance in Flexible Manufacturing Systems," Proc. Sixth Int'l Conf. Emerging Technologies and Factory Automation, Mar. 1997.

