School of Information Technology and Engineering (SITE)

Master of Computer Application (MCA)

Course Project Report

Deadlock detection using Path-Pushing Algorithm Distributed Operating System

Submitted for the Course ITA 5006: Distributed Operating System

Submitted To:-Proff. Sivakumar N.

By

SAYAN CHAKRABORTY	**18MCA0099**
AVINAY MEHTA	**18MCA0100**
ABHIJEET GIRI	**18MCA0088**
NITISH KUMAR	**18MCA0129**

APRIL 2019

TABLE OF CONTENTS

Abstract

Deadlock is a situation which occurs when two or more process are being executed concurrently in a system .In this situation one process has to wait while other process is executing ,and it can't be executed till the other process is finished executing or exist from the critical time .And detection is the method by which deadlock is detected for the prevention and solution .In the era of Inter of things every things is being executed by the systems and machines .The machines and system being used is handling multitasking ,In this way systems have to perform better and accurate with low latency .Lower the latency better the processing .There are lots of algorithm proposed for deadlock detection and prevention such as Banker's algorithm ,WFG(wait for graph),Dining philosopher algorithm ,Daemon algorithm etc. In this paper we are using path pushing algorithm.

Introduction

In distributed systems a process can request and release resources in any order, in which a process can request some resources while holding other resources. If the sequence of the resource allocation is not done in proper way there is a possibility to have a deadlock there.

Deadlock is a situation, where each of the processes waits for a resource which is being assigned to some other process. In this situation none of the process executes. In distributed system deadlock is handled with three strategies:-

- **Deadlock prevention** it is achieved by either having a process acquire all the needed resources simultaneously before execution or by preempting a process that holds the needed resource. This method has no of drawbacks like inefficiency, chances of having set of processes to be deadlock.
- **Deadlock avoidance** in this approach a resource is granted to a process if the resulting global system state is safe. It also has some drawbacks like maintaining information on global state for each site resulting requirement of huge amount of storage area and huge cost.
- **Deadlock detection** it requires an examination of the status of process-resource interaction for the cyclical wait. It has two favourable condition: 1) once a cycle is formed in the WFG, it persists until it is detected and broken and (2) cycle detection can proceed further concurrently with the removal of the activities of the system.

There are various control organization there is different algorithms for deadlock detection.

These organizations are:-

1) Centralized Control
2) Distributed Control
3) Hierarchical Control

There are different deadlock detection algorithms in Distributed Control like Path Pushing algorithm, Edge-chasing algorithm. We are using path pushing algorithm in this paper.

Path-pushing algorithm is also known as Obermarck's algorithm for path propagation in which the information of path sent from waiting node to blocking node. It is based on a database model using transaction processing. Sites which detect cycle in their partial WFG views convey the paths discovered to members of the (totally ordered) transaction. Here transactions are the processes. Transaction or process having the highest priority transaction detects the deadlock.

Obermarck's algorithm has two features:

- The nonlocal portion of the global TWF (Transaction wait for) graph at a side is abstracted by a distinguished node (called External or Ex) which helps in determining potential multisite deadlocks without requiring a huge global TWF graph to be stored at each site.
- Transaction are totally ordered, which reduces the number of messages and consequently decreases deadlock detection overhead. It also ensures that exactly one transaction in each cycle detects the deadlock.

Literature Survey

In [1] author says about the deadlock as most common problem in Distributed Data System and deadlock mainly prevents the system performance of overall Distributed Data System so it becomes necessary to detect and solve the problem of deadlock so that system work efficiently. Here author analyzed the "B.M Alom algorithm" and "Edge chasing algorithm" that are distributed algorithms. "B.M. Alom algorithm" detects and solve deadlock correctly if priorities are in order and they don't change. It detects the local and global deadlock by construction of Linear Transaction Structure (LTS) and Distributed Transaction Structure (DTS) respectively. On the other hand "Edge-chasing algorithm" detects and solves the deadlock problem with the help of updated Wait-for-graph (WFG) and sends it to the transaction that is in neighbor of it, along the edges of the graph. Probes are the messages having fixed length. There are some drawbacks of the proposed algorithms that are also mentioned by the author. B.M Alom algorithm has the problem of priorities. On the other hand Edge-chasing algorithm is unable to detect the deadlock if the initial transaction or process which initiates the propagation of probe is not the part of deadlock cycle.

Deadlock controlling [2] involves two problems: 'Deadlock Detection and Deadlock Resolution'. In the distributed systems the performance of the system gets changed drastically because of deadlock. So, it is necessary to solve the problem of deadlock detection and resolution as soon as possible. In this paper author focuses on detection of deadlocks in distributed systems and introduces many techniques in the form of survey and also give comparison table which defines the performance of different algorithms in which the author defines the length of message, delay of message and total no of messages etc. In this paper author also tells about the probe which is based on detection of algorithms in which wait-for-graph (WFG), matrix based detection algorithm, Linear transaction construction algorithm which is used for finding of local cycle and distributed transaction construction for finding of global cycle are drawn.

In [3] author presented the solution to detect the problem of distributed deadlock under the Single Resource Model. The solution is based on the synchronized hardware clocks. In deadlock cycle, single process detects the deadlock and then aborts it. The clock based technique can be extended to the more complex distributed deadlock models. In order to continue the transactions, it needs all the resource requested. In WFG deadlock corresponds like a cycle. When the detection is to be initiated, the blocked transaction sends the time stamped tokens to the data managers of the resources that are requested. The transaction that receives the token propagates it, based on the time-stamp value of the token, its current time stamp value and whether it can holds the resources that is requested by its predecessor or not, to the data managers of all of its requested sources. A cycle can exists as real time instants, if a transaction or process can receives its own token. In this paper author also states that there are many techniques are available worldwide for maintenance of accurate time in systems, but they are quietly expensive. In this paper the protocols used the leveraged availability from this rough global time base.

In this paper [4] the author discuss about the problems of 'Leader election' and deadlock detection/resolution. They are different problems. In this paper a new algorithm or a resolution is introduced that improves the cost of the communication. In this algorithm, to detect as well as solve the deadlock problem in messages of n nodes, it requires only O (n) messages. The author also states that this algorithm number of messages that are required for the system get reduced, whenever that arc appears at most ones it ensures that how many times a pair of nodes communicate

with each other. The author also tells that most of the negotiation processes are made of the leader election algorithm to cut down on messages.

In [5] author says that according to some researches when shared resources are tried to access by multiprocessing system and there is adaptation of some basic mechanisms, then deadlock can be easily there. Here author proposes a novel solution is provided for prevention of deadlock in “distributed transaction services” by “utilizing the timestamps mechanisms and advanced replicas”. In this paper author also proposed and designed high performance comprehensive mechanisms for detection and prevention of deadlock in two phase commit protocol (2PC) which is based upon “distributed transaction processing”. It is also demonstrated in the paper, the approach used in distributed transaction services is deadlock free and no more overhead if little more optimization is applied. For local nested transactions approach perfectly solves the deadlock. To distinguish among different requests resources, this design is highly depends on resource manager, it should be very strong so that it could communicate with transaction managers and it is required to keep the status of massive resources when concurrent conditions are very high. Different types of locks are also used in the proposed design, shared locks and exclusive locks. For shared locks they require more consistency and consideration, and they deserve some special kind of treatment from resource manager.

In[6] this paper author focuses on the occurrence of failures of system due to occurrence of deadlocks in the system. So design of deadlock not only viewed from the trade off performance of the system, but also it is required for prevention of the failures of systems. In this paper author formulates deadlock detection scheduling algorithm given by Ling *et al.*’s when there is a presence of system failures, and also derived “the optimal detection of deadlock time minimizing the long-run average cost per unit time”. With the introduction of algorithms for deadlock detection and resolution in message-complexities these algorithms being used. In this paper, with the help of Landau notation, asymptotic optimal frequency of detection and scheduling of deadlock in terms of number of distributed processes. The results which are analyzed are the direct extension of the works of Ling *et al.*’s but it could give the reality for occurrence of failures of system when it is subjected by the detection and resolution of deadlocks. The author also presents the number of distributed processes and the failure of the systems probably that can give a major effect to the long run average and message complexity per unit time, but not the scheduling time of deadlock. The author also mentioned the future planning about the extension of resulted probabilistic models from the different views of different cost criteria.

In this paper [7] a new technique is proposed to handle the congested network and potential deadlock anomalies. The technique which is proposed in this paper not only identifies the path of congested resources in a price manner but also disperse the detected congestion with the help of provided bubbles for the blocked packets due to congestion, in the active manner. This techniques in applicable for the deadlock recovery-based network routing algorithms such as fully adaptive routing as well as for the deadlock avoidance based network routing algorithms. In the previous case, to create the essential bubbles for the resolution of potential deadlock accelerated bubbles are implemented. There are many advantages of the proposed technique for the system and for network designers. The author also states that in future it could be very interesting to investigate the ways to reduce the average length of the traces of ping efficiently in the large networks. Moreover the design synthesis of the hardware prototype can be pursued for the better understand the cost and performance trade-offs in the terms of the complexity of hardware, operations in critical paths and the improved sustained throughput, that implements the proposed technique.

In[8] this paper, the author has discussed the "Intellectual Distributed Processing System" proposed as a new object-oriented distributed system which is used to eliminate the central managing mechanisms from a system. Therefore to understand the highly distributed managing mechanisms. Therefore the object as the global knowledge is distributed the deadlock and data consistency managing mechanism. The protocol has introduced through which we can detect the deadlock and can avoid it simultaneously called lock protocol. To execute the deadlock handling on different sites parallelly the reachable sets are used. The timestamp ordering method is used to correctly control the multiple requests for deadlock handling.

In[9] this paper the author is trying to solve the problem of deadlock generated due to voting based approach and priority-based approach. In voting based approach the process getting the majority of votes will be allowed to enter in the "critical section". But the problem with this approach is if no process will gain the majority of votes then the process will remain idle. Where in priority-based approach the process having the highest priority will get into critical section first but the problem with this approach is the processes with lower priority will not get the chance to execute for a very long time. Therefore the author in this paper is come up the combined solution for priority and voting based approach by putting some constraint like "treatment for 'non-maskable ' interrupts" and "Shortest job first" techniques. It allows creating multiple critical sections in a distributed system and it stops processes to entering in idle state.

In[10] this paper, the author has developed the algorithm to remove the deadlock generated by the dependency of two processes. The processes can be dependent on each other (Cyclic state) in different conditions so if the one process stopped working, therefore, the process will keep waiting and finally go to idle state and the deadlock will generate. Hereupon resolution of deadlock, the recovery is affected when we return the system to a consistent state. Here the distributed algorithm used for deadlock detection is classified which refers to techniques used to the dissemination of dependency information called as top down and bottom up.

In[11] this paper, the author has focused on the problems generated due to the time-out when a deadlock occurs. A deadlock can be avoided by pre allocation of resources but in some cases when the deadlock has occurred then rather than invoking deadlock handling mechanism it goes for a time out which leads to loss of data and progress. Therefore in this paper, the author has developed a suite of deadlock detection protocols through which the problem of not invoking the deadlock handling mechanism can be resolved and deadlock can be detected when generated. Here with the help of a multi-cycle detection protocol, it can resolve multiple deadlocks at the same time.

In[12] this paper, the author is talking about the problem in MPI(Message Passing Interface) as in MPI due to its multitude of communication functions it has the problem of usage error. Here the author develops the tool called MUST which helps to provide the automatic correctness check. It can check the correctness in distributed mode but not for its deadlock detection. This limitation applies only on the tools that use centralized detection algorithm and timeout approach. The algorithm developed by the author implements the scalable Message passing Interface deadlock detection. They have done about 4096 processes stress tests to demonstrate the scalability of this approach.

In this paper [13] As "multiprocessor system on a chip" (MPSOC) designs become more common, and used everywhere, therefore, the code sign engineers of hardware and software are facing new problems in operating system integration. To make the MPSOX or operating system code sign more efficient in terms of speed, In this paper the author describes the latest research in hardware and software partitioning in "real-time operating system"(RTOS). In this author focus on new deadlock detection and avoidance. Among different configuration of RTOS and MPSOC design here the author has shown the system with "deadlock detection hardware unit"(DDU) which can achieve up to 46% of the increase in speed of application running time over the system with deadlock detection in software. Also, they show another example of "deadlock avoidance hardware unit"(DAU) which not only avoids the deadlock but also increase the speed of application up to 44%.

In[14] this paper, the author has developed a new procedure for storing and forwarding the deadlock prevention. This technique is based on timestamps which make it more efficient and flexible for message transmission. It allows unrestricted packet routing. Also, for a network with one exchange buffer per node, the procedure proposed by the author is completely "independent of network reconfiguration". In this author is mentioned that it is possible to apply timestamps idea to get prevention from types of deadlock.

[15]Day by day computer performance has increased frequently in recent years and communication subsystems has become bottleneck in systems .To solve this problem current distributed system are using switched based interconnection networks . There are problem occur when use switch network reconfiguration is the reduction of performance during the change assimilation process .To avoid this problem we use in this paper we propose and

evaluate a first reconfiguration method for source routing networks that not restrict the packets during the change of assimilation process. We use up and down algorithm for routing the networks.

[16]Today's distributed database system and ready to handle more than thousand concurrent transaction at a time. In this distributed system use of wait-die and wound-wait mechanisms are deal with the deadlocks but its always not works efficiently .To avoid the deadlock limitation we use the drag-back model its works on partial rollback of transaction and also it not compromise with ACID properties.

[17]Most of the people studies bases on the diffusing computational technique, where reply of propagation required to detect deadlock. The reply carry the information between process of the initiator algorithm to determined deadlock this algorithm is not execute fast. This paper focus on the fast resolution of deadlock detection system. This is done by the concurrent execution of the algorithm if the algorithm is not execute concurrently then duplicate deadlock is aborted.

In [18] distributed system deadlock detection is very complex work in distributed system no node has complete information about the system. In this paper proposed a semi-centralized algorithm using adaptive gossip to detect deadlock in distributed systems. The inherent nature of the gossip protocol can be said to be fault-tolerance, scalable and efficient while maintaining the properties. The algorithm is performed well in term of time and message. The detection of phantom deadlocks are minimized in the algorithm and no of round are increased linearly.

In [19]distributed system debugging of large scale application in parallel is a very big issue in the real world for debugging large scale application there are many problem occur like increasing of error and impact on performance. To avoid this problem in this paper propose novel on the fly algorithm to detect the race condition that causes deadlocks. The algorithm is high effectiveness because it is scalable when apply it in to the large scale environment and it is also compatible with parallel application.

[20]Due to the rapid advance technology introduce Multiprocessor System-on-chips (MPSoCs) are commonly used in the computing platform. It is possible in feature that multiprocessor system on chip equipped with a large number of processing element for that management faces some challenges like deadlock is one of the critical issues. A parallel, hardware-oriented OSDDA, suitable for MPSoCs, is presented in this paper. During the phase of detection preparation, OSDDA handles different types of resource allocation events differently based on a classification of all resource allocation events.

The paper[21] proposed detection of deadlocks in resource shearing for distributed system. The algorithm is based on the sending message with edge using Wait-for graph. The algorithm is very efficient for avoids the false deadlock and capable of detecting deadlocks the subset of processes in the system. The algorithm also work well when multiple node are initiated in the deadlock detection algorithm.

Proposed Methodology

Deadlock detection at a site follows the following iterative process:-

Step1: The site waits for deadlock-related information from other sites like 'Ex → T_1 → T_2 →Ex'

Step 2: The site combines the received information with its local TWF graph to build an updated TWF graph. It then detects all cycles and breaks only those cycles which do not contain the node 'Ex'. These cycles are the local to this site. All other cycles have the potential to be a part of global cycles.

Step 3: For all cycles 'Ex → T_1 → T_2 →Ex' which contain the node 'Ex' (these are the potential candidates for global deadlock), the site transmits them in string from 'Ex, T_1, T_2, Ex' to all other sites where a sub transaction of T_2 at this site.

Step 4: Send the string 'Ex, T_1, T_2, $T_{3'}$, Ex' to other sites only, if T_1 is higher than T_3 in the lexical ordering.

Implementation

We are implementing the path pushing algorithm in c.

Step 1:- By making a wait for graph.

Step2:- Receives any string of nodes that are transmitted from other sites and adding them into graph. For every transaction to be identified in the string each node is to be created.

Step3: creating wait for node for Ex for receiving or visiting of node.

Step4: creating wait for node for Ex for sending of node.

Step 5: analyzing the graph and listing all cycles.

Step6: if any node is received more than once then deadlock is detected, as there would be the same loop. If all nodes are received for Ex then there is no deadlock.

Code

```
#include <stdio.h>

#include <stdlib.h>

#define FALSE 0

#define TRUE 1

#define MAX_COUNT 10


int deadlockDetected = FALSE;

struct Stack {
```

```
    int top;
    int array[MAX_COUNT];
};

void push(struct Stack *stack, int data) {
    if (stack -> top == MAX_COUNT) {
        printf("Overflow");
        exit(1);
    }
    stack -> array[++(stack -> top)] = data;
}

int pop(struct Stack *stack) {
    if (stack -> top == -1) {
        printf("Underflow");
        exit(1);
    }
    int temp = stack -> array[(stack -> top)--];
    return temp;
}

/**
 * util method to print an array
 */
void printArray(int *array, int n) {
    int i = 0;
    printf("\nArray : ");
    for (i = 0; i < n; i++) {
```

```
        printf("%d ", array[i]);
    }
    printf("\n");
}

/** * util method to print a stack*/
void printStack(struct Stack *stack) {
    int i = stack -> top;
    printf("\nPrinting stack : ");
    while (i >= 0) {
        printf("%d, ", stack -> array[i--]);
    }
}

/**
 * method to show a graph
 */
void showGraph(int **graph, int totalNodes) {
    int i = 0;
    int j = 0;
    printf("\nGraph:");
    for (i = 0; i < totalNodes; i++) {
        printf("\n");
        for (j = 0; j < totalNodes; j++) {
            printf("%d  ", graph[i][j]);
        }
    }
    printf("\n");
```

```
}

/**
 * validates if edge that has entered is correct or not.
 */
int validateEdge(int source, int sink, int numProcesses, int numResources, int *usedResource) {

    if (source < 1 || sink < 1) {
        printf("Invalid id. can't be less than 1");
        return 0;
    }

    if ((source >= 1 && source <= numProcesses) &&
        (sink >= 1 && sink <= numProcesses)) {
        printf("Both ids belong to process.");
        return 0;
    }

    if ((source > numProcesses && source <= numProcesses + numResources) &&
        (sink > numProcesses && sink <= numProcesses + numResources)) {
        printf("Both ids belong to resource.");
        return 0;
    }

    if (source > numProcesses + numResources || sink > numResources + numProcesses) {
        printf("Invalid id.");
        return 0;
    }
```

```
    if (source > numProcesses) { //means source is a resource
        if (usedResource[source - numProcesses - 1]) {
            printf("Resource %d already used. Can't assign 1 resource to 2 processes.", source);
            return 0;
        } else {
            usedResource[source - numProcesses - 1] = TRUE;
        }
    }
    return 1;
}

/**
 * this is a depth first search algorithm which keeps on pushing all visited nodes on to a stack.
 * The moment a node is visited and is already present on stack, it means that there is a deadlock.
 */
void dfs(int **graph, int *visited, int *isOnStack, int source, int totalNodes, struct Stack *stack, int *isPartOfCycle)
{
    int i = 0;
    if (isOnStack[source]) {
        if (!isPartOfCycle[source]) {
            printf("\nDeadlock detected : (");
            deadlockDetected = TRUE;
            i = stack -> top;
            while (stack -> array[i] != source) {
                i--;
            }
            while (i <= stack -> top) {
```

```
            isPartOfCycle[stack -> array[i]] = TRUE;
            printf("%d -> ", (stack -> array[i] + 1));
            i++;
        }
        printf("%d)", (source + 1));
    }
    return;
  }
  visited[source] = TRUE;
  push(stack, source);
  isOnStack[source] = TRUE;
  for (i = 0; i < totalNodes; i++) {
    if (graph[source][i] == 1) {
      dfs(graph, visited, isOnStack, i, totalNodes, stack, isPartOfCycle);
    }
  }
  pop(stack);
  isOnStack[source] = FALSE;
}

/**
 * This the method that detects a deadlock
 */
void detectDeadlock(int** graph, int n) {
  int* visited = (int *) malloc (sizeof(int) * n);
  int* isPartOfCycle = (int *) malloc (sizeof(int) * n);
  int* isOnStack = (int *) malloc (sizeof(int) * n);
  int i = 0;
```

```
    int j = 0;
    struct Stack stack;
    stack.top = -1;

    for (i = 0; i < n; i++) {
        visited[i] = FALSE;
        isOnStack[i] = FALSE;
        isPartOfCycle[i] = FALSE;
    }

    for (i = 0; i < n; i++) {
        if (visited[i] == FALSE) {
            dfs(graph, visited, isOnStack, i, n, &stack, isPartOfCycle);
            for (j = 0; j < n; j++) {
                isOnStack[j] = FALSE;
            }
        }
    }
    if (!deadlockDetected) {
                        printf("\nThere is no deadlock in the system.");
    }
    free(visited);
    free(isPartOfCycle);
    free(isOnStack);
}

/**
 * main method
```

```
 */
int main() {

    int numProcesses = 0;
    int numResources = 1;
    int numEdges = 0;
    int i = 0;
    int j = 0;
    int source;
    int sink;
    printf(" enter number of processes");
    scanf("%d", &numProcesses);
    printf(" enter number of resources");
            scanf("%d", &numResources);

     printf("Enter the matrix values");
    int *usedResource = (int *) malloc (sizeof(int) * numResources);
    for (i = 0; i < numResources; i++) {
        usedResource[i] = FALSE;
    }

    int totalNodes = numProcesses + numResources;
    int** graph = (int **) malloc(sizeof(int *) * totalNodes);
    for (i = 0; i < totalNodes; i++) {
        graph[i] = (int *) malloc (sizeof(int) * totalNodes);
    }

    for (i = 0; i < totalNodes; i++) {
```

```
        for (j = 0; j < totalNodes; j++) {

                                    scanf("%d", &graph[i][j]);

        }
    }

    detectDeadlock(graph, totalNodes);
    free(usedResource);
    for (i = 0; i < totalNodes; i++) {
        free(graph[i]);
    }
    free(graph);
    printf("\n");
    return 0;
}
```

Output

```
 enter number of processes2
 enter number of resources1
Enter the matrix values2
2
4
5
2
3
6
1
2

There is no deadlock in the system.

Process returned 0 (0x0)   execution time : 9.671 s
Press any key to continue.
```

Figure 1 having no deadlock

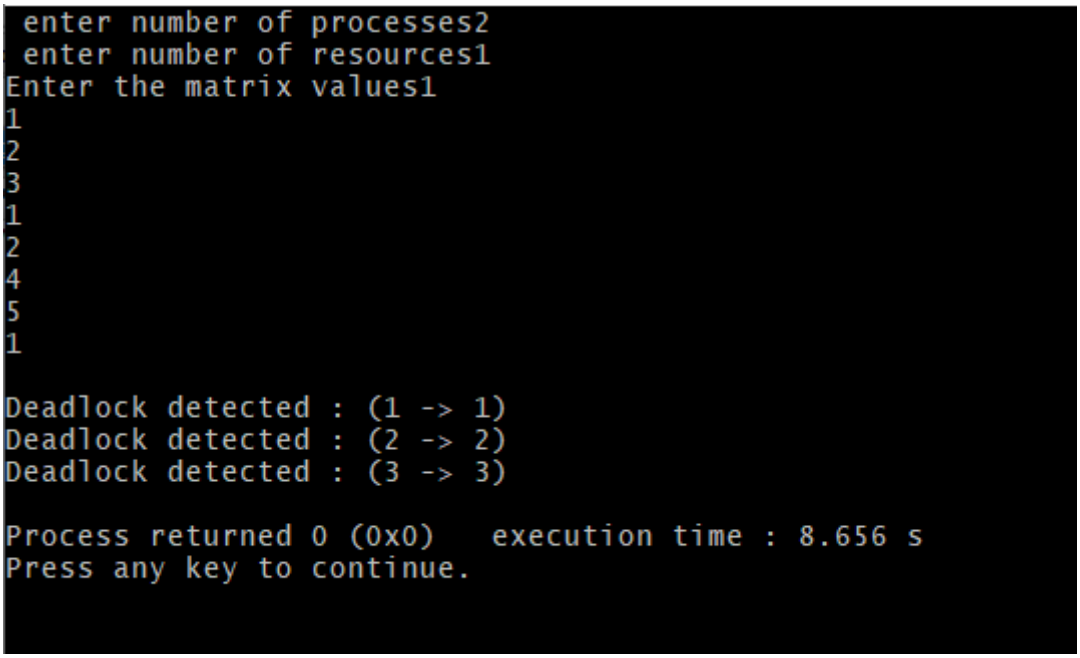

figure 2 having deadlock

Conclusion

Deadlock detection scheduling is an important, yet often over looked aspect of distributed deadlock detection and resolution. The performance of deadlock handling not only depends upon per-execution complexity of deadlock detection/resolution algorithms, but also depends fundamentally upon deadlock detection scheduling and the rate of deadlock formation. Excessive initiation of deadlock detection results in an increased number of message exchange in the absence of deadlocks, while insufficient initiation of deadlock detection incurs an increased cost of deadlock resolution in the presence of deadlocks. As a result, reducing the per-execution cost of distributed deadlock detection/resolution algorithms alone does not warrant the overall performance improvement on deadlock handling. The main thrust of this paper is to bring awareness to the problem of deadlock detection scheduling and its impact on the overall performance of deadlock handling. The key element in our approach is to develop a time dependent model that associates the deadlock resolution cost with the deadlock persistence time. It assists the study of time-dependent deadlock resolution cost in connection with the rate of deadlock formation and the frequency of deadlock detection initiation, differing significantly from past research that focuses on minimizing per-detection and per-resolution costs. there is a drawback of this algorithm and that is sometimes it detects phantom deadlocks.

References

[1] AmbikaGehlot, AkanshaJaiswal, Vandana Kate; “A survey on Distributed Deadlock and Distributed Algorithms to Detect and Resolve Deadlock” In Proceedings of Symposium on Colossal Data Analysis and Networking(CDAN) 2016, pp 1-6,2016.

[2] MamtaYadav, UdaiShanker; “Detection Techniques for Deadlock in Distributed Database System” In Proceedings of IEEE 3rd International Conference on Computing for Sustainable Global Development 2016, pp 906-911, 2016.

[3] Jean Mayo, Phil Kearns; “Distributed Deadlock Detection and Resolution Based on Hardware Clocks” In Proceedings of 19th IEEE International Conference on Distributed Computing Systems 1999, pp 208-215, 1999.

[4] Maria Castillo, Fedrico Farina, Alberto Cordoba; "A Dynamic Deadlock Detection/Resolution Algorithm with Linear Message Complexity" In Proceedings of IEEE 20thEuromicro International Conference on Parallel, Distributed and Network-based Processing 2012, pp 175-179, 2012.

[5] Lin Lou, Feilong Tang, Ilsun You, MinyiGuo, Yao Shen, Li Li, "An Effective deadlock Prevention Mechanism for distributed transaction Management" In Proceedings of IEEE Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing 2011, pp 120-127, 2011.

[6] Akikazu Izumi, Tadashi Dohi, Naoto Kaio;"Deadlock Detection Scheduling for Distributed Process in the Presence of System Failures" In Proceedings of IEEE 16th Pacific Rim International Symposium on Dependable Computing 2010,pp 133-140, 2010.

[7]Yong Ho Song, T.M. Pinkston; "Distributed Resolution of Network Congestion and potential deadlock Using Reservation-Based Scheduling" In Proceedings of IEEE Transactions on Parallel and Distributed Systems 2005,pp 686-701, 2005.

[8]Shinsuke Tamura, YasukuniOkataku, ToshibumiSeki :"Distributed deadlock avoidance and detection in intellectual distributed processing system",Proceedings of the 1988 IEEE International Conference on Systems, Man, and Cybernetics, PP 1279- 1282.

[9]KamtaNath Mishra " An efficient voting and priority based mechanism for deadlock prevention in distributed systems", Proceedings of the 2016 International Conference on Control, Computing, Communication and Materials (ICCCCM) , PP 1- 6.

[10]A.T. Amin ; M.P. Freeman " Proof techniques for distributed algorithms for deadlock handling", IEEE Proceedings of the SOUTHEASTCON '91 Year: 1991
, PP 470- 473.

[11] WaqarHaque ; Matthew Fontaine ; Adam Vezina" Adaptive Deadlock Detection and Resolution in Real-Time Distributed Environments", 2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), PP 577- 591.

[12]Tobias Hilbrich ; Bronis R. de Supinski ; Wolfgang E. Nagel ; Joachim Protze ; ChristelBaier ;Matthias S. Müller" Distributed Wait State Tracking for Runtime MPI Deadlock Detection", SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis , PP 1- 12.

[13]J.J. Lee ; V.J. Mooney" Hardware/software partitioning of operating systems: focus on deadlock detection and avoidance", IEE Proceedings - Computers and Digital Techniques , PP 167- 182.

[14] J. Blazewicz ; J. Brzezinski ; G. Gambosi" Time-Stamp Approach to Store-and-Forward Deadlock Prevention", IEEE Transactions on CommunicationsYear: 1987 , Volume: 35 , Issue: 5, PP 564- 566.

[15]A Deadlock-Free Dynamic ReconfigurationScheme for Source Routing Networks Using Close Up*/Down* GraphsAntonio Robles-Go´mez, Member, IEEE, Aurelio Bermu´dez, Member, IEEE, and Rafael Casado, Member, IEEE

[16]Avoiding unnecessary deaths Drag-Back, a deadlock avoidance model Luis Mejía-Ricart, Aspen Olmsted Computer Science Department College of Charleston Charleston, SC, United States mejiaricartlf@g.cofc.edu, olmsteda@cofc.edu

[17]Fast, Centralized Detection and Resolution ofDistributed Deadlocks in the Generalized Model

Soojung Lee, Member, IEEE Computer Society

[18]MoumitaChatterjee Department of Computer Science and Engineering University of Calcutta Kolkata, India moumitachatterji@gmail.com978-1-5090-0673-1/16/$31.00 ©2016 IEEE S K Setua Department of Computer Science and Engineering University of CalcuttKolkata, India

[19]2016 15th International Symposium on Parallel and Distributed ComputingRace Condition and Deadlock Detection for Large-scale Applications

[20]A True O(1) Parallel Deadlock DetectionAlgorithm for Single-Unit Resource Systems and Its Hardware ImplementationXiang Xiao, Student Member, IEEE, and Jaehwan John Lee, Member, IEEE

[21]"A Novel Scheduling Strategy for an Efficient
Deadlock Detection Aida". O. Abd El-Gwad, Ahmed. I. Saleh, Mai. M. Abd-ElRazikDept. of Computer and System, Faculty of Engineering, Mansoura University, Egypt

[22]**Alireza Soleimany2, Zahra Giahi1**
1Computer Engineering Department, Islamic Azad University, Lahijan Branch, Lahijan, Iran
2Computer Engineering Department, Islamic Azad University, MeshkinShahr Branch, MeshkinShahr, Iran

[23]"A Dynamic Deadlock Detection/Resolution Algorithm with Linear Message Complexity"Mar´ıa Castillo, Federico Fari˜na, Alberto C´ordobaDept. Ingenier´ıaMatem´atica e Inform´atica Universidad P´ublica de Navarra Campus de Arrosad´ıa, 31006 Pamplona (SPAIN)

[24] "Deadlock Avoidance for Interconnection Networkswith Multiple Dynamic Faults"Gonzalo Zarza, Diego Lugones, Daniel Franco, and Emilio LuqueComputer Architecture and Operating Systems Department. University Autonoma of Barcelona, Spain

[25]"A Parallel Deadlock Detection Algorithm with O(1) Overall Run-time Complexity"Jaehwan John Lee and Xiang Xiao ECE Department, Purdue School of Engineering and Technology Indiana University-Purdue University Indianapolis

[26]"A True O(1) Parallel Deadlock Detection
Algorithm for Single-Unit Resource Systems and Its Hardware Implementation"Xiang Xiao, Student Member, IEEE, and Jaehwan John Lee, Member, IEEE

[27]"A Semi-Centralized Algorithm to Detect and Resolve Distributed Deadlocks in the Generalized Model" Zhi Tao,Hui Li*, Bing Zhu, Yunmin Wang Shenzhen Eng. Lab. of Converged Networks Technology Shenzhen Key Lab. of Cloud Computing Tech. and App Peking University Shenzhen Graduate School, Guangdong, China