International Journal of Computer (IJC)

ISSN 2307-4523 (Print & Online)

© Global Society of Scientific Research and Researchers

http://ijcjournal.org/

Upgraded Deadlock Averting Algorithms in Distributed Systems

Abdullah Jan[a*], Syed Atif Ali Shah[b], Zafar Khan[c], Sakin Jan[d], Adnan Haroon[e], Imad Khan[f], HabiburRahman Tariq[g]

[a,b,c]Northern University Nowshera, Peshawar, Pakistan

[d,e,f,g]Abasyn University Peshawar, Peshawar, Pakistan

[a]Email: abdullah@northern.edu.pk, [b]Email: atif@northern.edu.pk

[c]Email: Zafarkhan@northern.edu.pk, [d]Email: sakinjan100@gmail.com

[e]Email: safanktk44@gmail.com, [f]Email: khanimad082@gmail.com

[g]Email: habiburahmantariq@gmail.com

Abstract

Distributed system deadlock is like ordinary deadlock but it is difficult to prevent or detect when it is traced down. In the distributed system all, the related information is distributed over many machines. However, deadlock in distributed systems is tremendously serious. Therefore, it is important to understand how this deadlock is different from the ordinary deadlock and how to prevent it. To prevent deadlock in the distributed system there are two techniques to prevent it one wound-wait and other is wait-die. Therefore, the problem in these algorithms are that they just attend to the timestamp of the process but not the priority of them but in the real operating system priority of the process is very important. In this paper, we present upgraded deadlock averting algorithms and these algorithms are deal with both priority and time stamp of processes.

Keywords: Deadlock averting; Distributed Systems Deadlock averting; Distributed Systems; Deadlock averting Algorithms.

----

* Corresponding author.

1. Introduction

Distributed system is a group of independent nodes that seems to its user as a single intelligible system. This definition mention two feature of the distributed system. The first feature of the distributed system is that a distributed system is a group of computing nodes, which behave independently of each other. A node is generally referred to as a computing element that can either be hardware or software process. The second feature of the distributed system is that when a user uses distributed systems they think they are dealing with a single intelligible system [1]. Benefits of the distributed system are sharing of data mean that the distributed environment provides facilities to users if the users at one site may be able to access the data be present in at other sites, Autonomy means of data distribution each site is able to hold a degree of control over data that are stored locally. In addition, availability means that if one site crash in a distributed system the other sites may be able to continue working. Thus, a crash of one site does not essentially imply the shutdown of the whole system [2]. The weakness of distributed systems is software development cost because it is very difficult to implement a distributed database system. In addition, distributed database work in parallel it is difficult to guarantee the correctness of algorithms. Specially operation during failure of part of the system and recovery from failures. The possibility exists for extremely subtle bugs. Enhanced processing overhead problem arise in distributed systems [2]. In the past not too far away on the server, two devices combined and created first distributed system.at the very start, one computer can perform one task at a time. If we need multiple tasks to be performed at a time, so we need to run multiple computers at a time. However, running the computers in parallel at a time in not building a truly distributed system since it needs a method to build communication between different computers. Therefore, the evolution of distributed systems start form message-oriented communication and then Service Oriented Architecture and then ESB (Enterprise Service Bus) and then Application Programming Interface (API) and then Virtual machines and then Container-based application deployment [3]. Nowadays distributed system is originated everywhere and their design is simple and there are many scopes to develop more service and application. Designing distributed system many challenges arise related to design like Heterogeneity, Openness, Security, Scalability, Failure handling, Concurrency, Transparency, Quality of service, Reliability, Performance [4]. The rest of the paper organized as follow. Section 2 related work. Section 3 briefly explain what is deadlocks and the scope of deadlocks in distributed system/distributed operating system and the overview of distributed system deadlock averting algorithms. Section 4 provide the complete details of proposed algorithms for deadlock averting. Section 5 the proposed algorithms are to be implemented using C++ programming language. Section 5 provide complete summarization.

2. Related Work

Review the techniques related to deadlock averting in the first method it locks complete data item before each transaction begins to execute in the second method allocating global timestamp to each transaction to preventing deadlocks [6]. Review two deadlock-averting techniques. Wait-die is based on the non-preemptive method and the other one wound wait is based on preemptive method so both the techniques can avoid starvation.so the major problem exist in these techniques is that unnecessary rollbacks may occur[7]. Try to averting deadlock in a grid environment to standby the data consistency then improve amount by exploiting the accessibility of resource and to the potential that all the resource available in the grid are efficiently used [8].

Present a new method for averting resource deadlock in a distributed system. Deadlock averting using well-ordered and atomic multicast in a distributed system. Ordered and atomic multicast (OAM) suggestion communication services with the help of these services deliver communication or message to their endpoint with an assurance of well-ordered and atomicity so he shows that well-ordered and atomicity is top in an existing condition to prevent deadlocks [9]. Deadlocks averting methods for services oriented transaction processing. Deadlock problem is more difficult in the service-oriented architecture. Present a replication based approach to evade the local deadlocks and a timestamp based technique to indirectly release the global deadlocks. So usually, the algorithm is intended for both local and global deadlock prevention [10].

3. Distributed Deadlock System

Distributed system present a high quality of resource and data sharing, a condition in which deadlocks may happen. [6] Deadlock state arises in a distributed system. When two or more processes waiting for state incessantly for an event and which is pretentious by one of the waiting processes [11]. A deadlock can be ensure if the four conditions grip concurrently the first one is Mutual exclusion. In mutual exclusion condition in which one resource be, able to allocated to one process. The second condition is Hold and wait. In Hold and wait, the processes can grip resources and demand new resources. Third condition is Non-preemption. In Non-preemption, resources cannot be forcefully isolated from a process. The fourth condition is Circular wait. In Circular, wait condition in which individual process wait for resource that the subsequent process in chain holds [12]. There are four methods for handling deadlocks. The first method is Ignore the deadlock. The second method is Deadlock detection. In deadlock detection, first find the deadlock and then break them. The third Method is Deadlock prevention. In deadlock prevention, create the structure of a system in such a manner those three conditions for deadlock cannot occur. The forth method is Deadlock avoidance. In Deadlock, avoidance explains methods that try to control if a deadlock will happen at the time a resource is request and respond to the request in a method that avoids the deadlock. In this paper, we concentrate on deadlock prevention, which is the best usually implemented deadlock solution. The two popular techniques for deadlock presentation in distributed system is wait-die and wound-wait. Transactions are well order by timestamps. When a transaction is create, a timestamp field is attach to it. For example, let assume two transaction are Ti and Tj and these two transaction marked with timestamp Ts (Ti) and Ts (Tj). In the following wait-die algorithm [13]. if old process waiting for the young process so the old process continues to wait otherwise if the young process is waiting for the old process so the young process will expire and restarted again with the same timestamp so this algorithm just attend timestamp of the process but not the priority of the processes [5].

3.1. Wait-die

```
If (Ts (Ti) < Ts (Tj))
    Print (Ti is older than Tj, so Ti is allowed to wait)
Else
Print (Ti younger than Tj, so Ti is terminate and is resume later with the same timestamp) attend timestamp of
```

the process but not the priority of the processes [5].

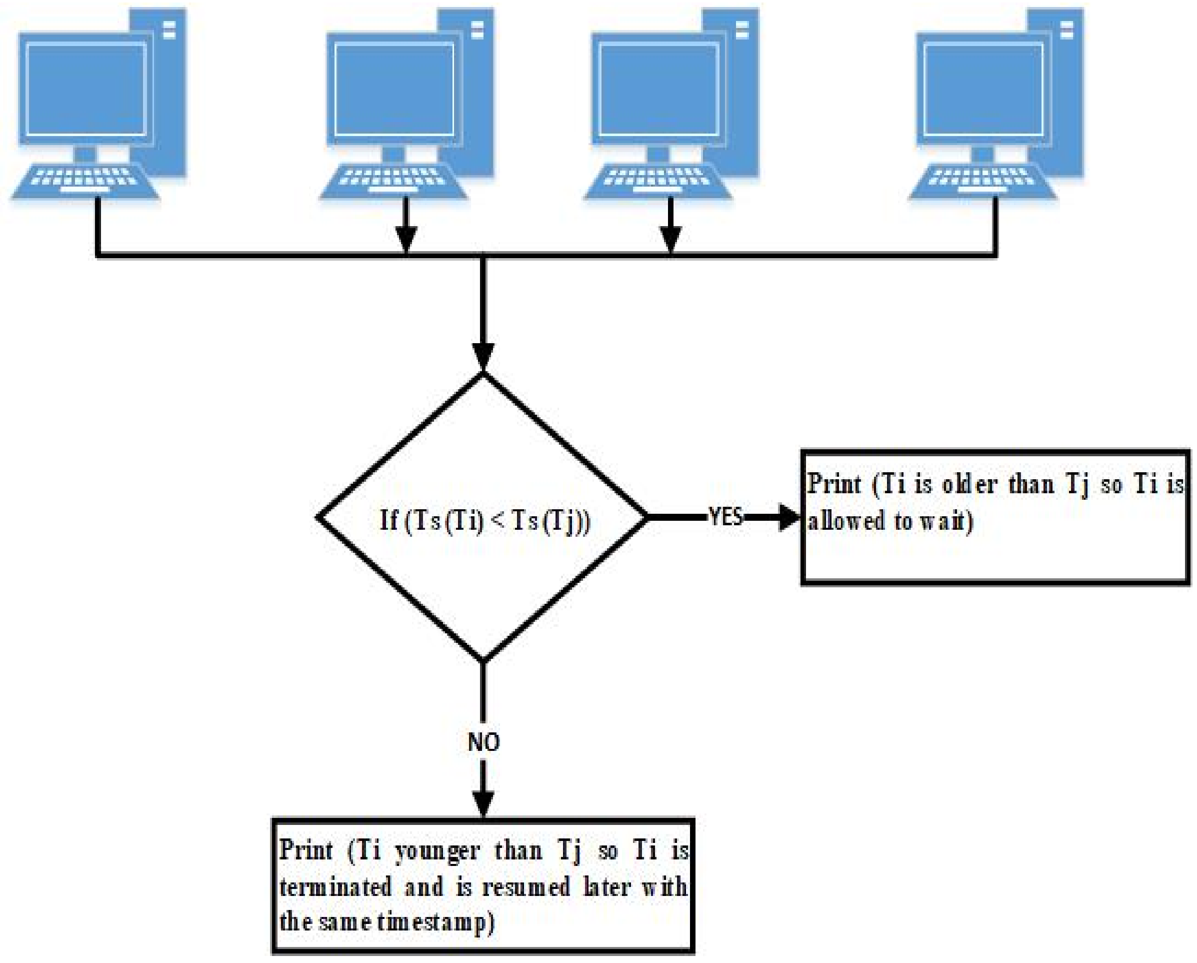

If (Ts(Ti) < Ts(Tj))
YES
Print (Ti is older than Tj so Ti is allowed to wait)
NO
Print (Ti younger than Tj so Ti is terminated and is resumed later with the same timestamp)

Figure 1: Wait-Die Algorithm

3.2. Wound-Wait Algorithm

In the following wound-wait algorithm is the old process is to be wait for young process so the old process damaged the young process and acquire it resources otherwise young process is to be wait for the old process so the young process continue to wait [5].

```
If (Ts (Ti) < Ts (Tj))
    Print (Ti is older than Tj so Ti wound Tj and acquire it resources)
Else
        Print (Ti earlier than Ti so Tj is allow to wait)
```

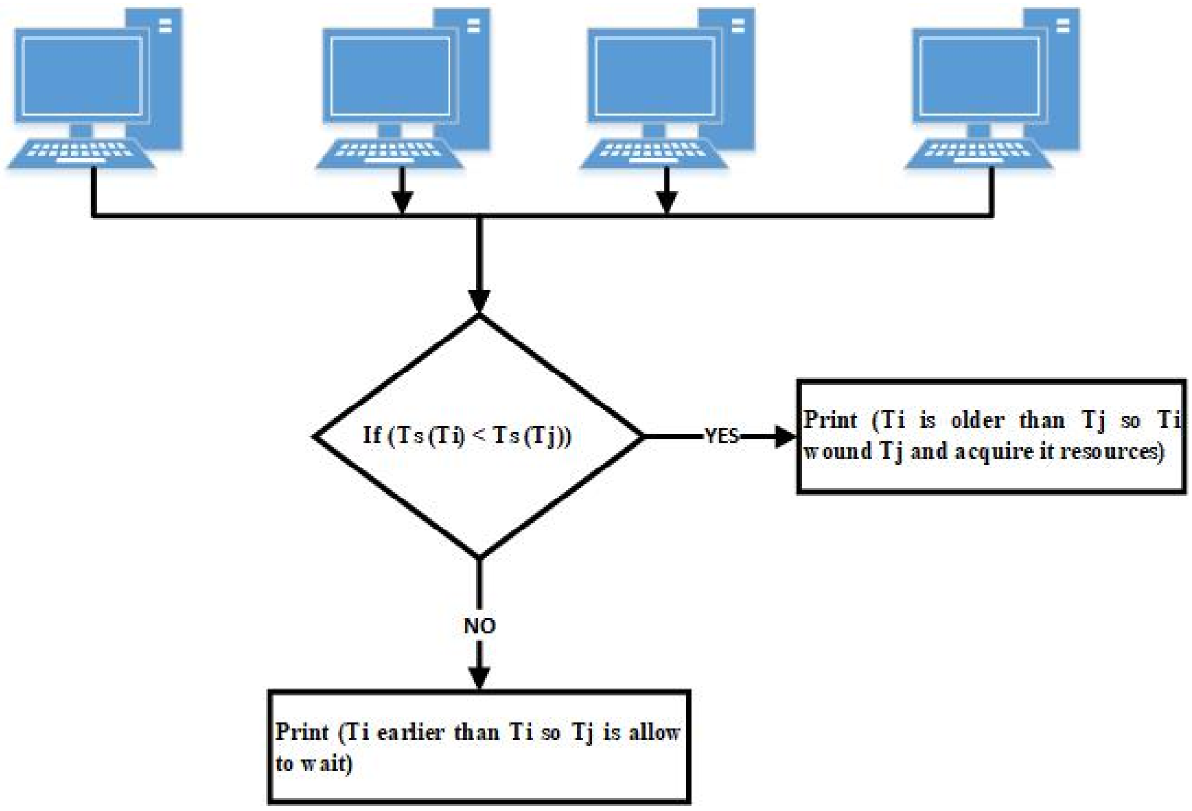

If (Ts(Ti) < Ts(Tj))
YES
Print (Ti is older than Tj so Ti wound Tj and acquire it resources)
NO
Print (Ti earlier than Ti so Tj is allow to wait)

Figure 2: Wound-Wait Algorithm

4. Proposed Algorithms

In the proposed upgraded wait die algorithm if the old process is to wait for the young process so the old process continues to wait but the priority of the older process is higher than younger process so the young process should be wait. Else, if the old process is to wait for the young process so the old process continues to wait but the priority of the young process is higher than Old process so the Young process continues to wait. In the proposed upgraded wound wait algorithm if the old process is waiting for young and the priority of the old process is higher than young process so old process wound young process and acquire its resource. Else, if the old process is to wait for young process and the priority of the young process is higher than the old process so the old process should be wait.

4.1. Proposed Wait-Die

```
If (TS (Ti) < TS (Tj) && PTi > PTj)

   Print ((Tj) is to allowed to wait because (Ti) is older than (Tj) but priority of (Ti) is greater than (Tj))

 Else If (TS (Ti) > TS (Tj) && PTi > PTj)

     Print (Tj is allowed to wait because Tj priority is less than Ti)
```

```
Else If (TS (Ti) > TS (Tj) && PTi < PTj)

    Print ((Ti) is allowed to wait because priority of (Tj) is greater than (Ti))

Else If (TS (Ti) < TS (Tj) && PTi < PTj)

    Print ((Ti) is allowed to wait because priority of (Tj) is greater than (Ti))
```

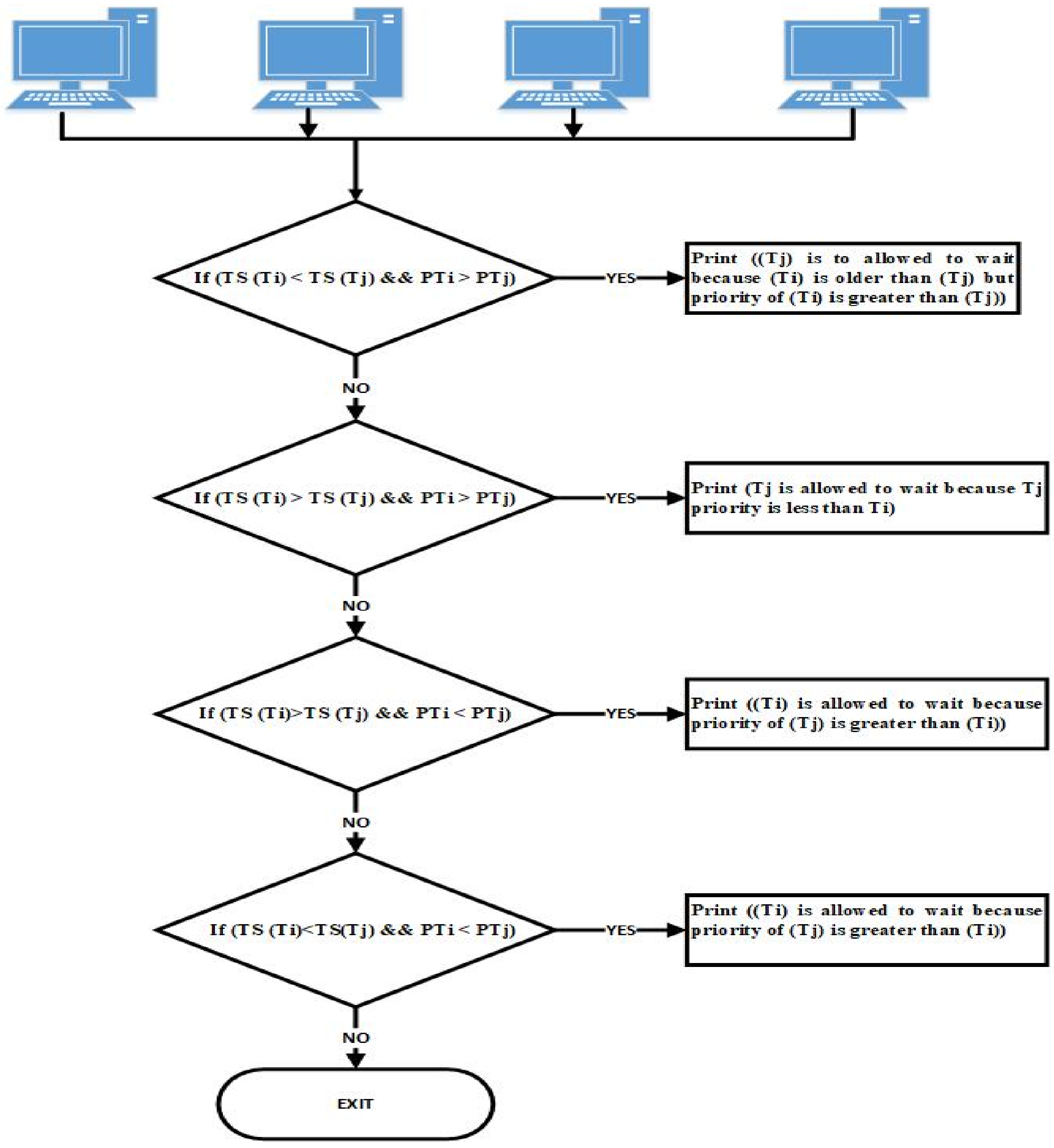

If (TS (Ti) < TS (Tj) && PTi > PTj)
YES
Print ((Tj) is to allowed to wait because (Ti) is older than (Tj) but priority of (Ti) is greater than (Tj))
NO
If (TS (Ti) > TS (Tj) && PTi > PTj)
YES
Print (Tj is allowed to wait because Tj priority is less than Ti)
NO
If (TS (Ti)>TS (Tj) && PTi < PTj)
YES
Print ((Ti) is allowed to wait because priority of (Tj) is greater than (Ti))
NO
If (TS (Ti)<TS(Tj) && PTi < PTj)
YES
Print ((Ti) is allowed to wait because priority of (Tj) is greater than (Ti))
NO
EXIT

Figure 3: Wait-Die Algorithm

4.2. Proposed Wound-Wait

```
If (TS (Ti) < TS (Tj) && PTi > PTj)
    Print (Ti wound Tj and get it resource)
  Else If (TS (Ti) < TS (Tj) && PTi < PTj)
        Print (Ti continue to wait)
      Else If (TS (Ti) > TS (Tj) && PTi < PTj)
            Print (Tj wound Ti and get it resource)
          Else If (TS (Ti) >TS (Tj) && PTi >PTj)
                Print (Tj continue to wait)
```

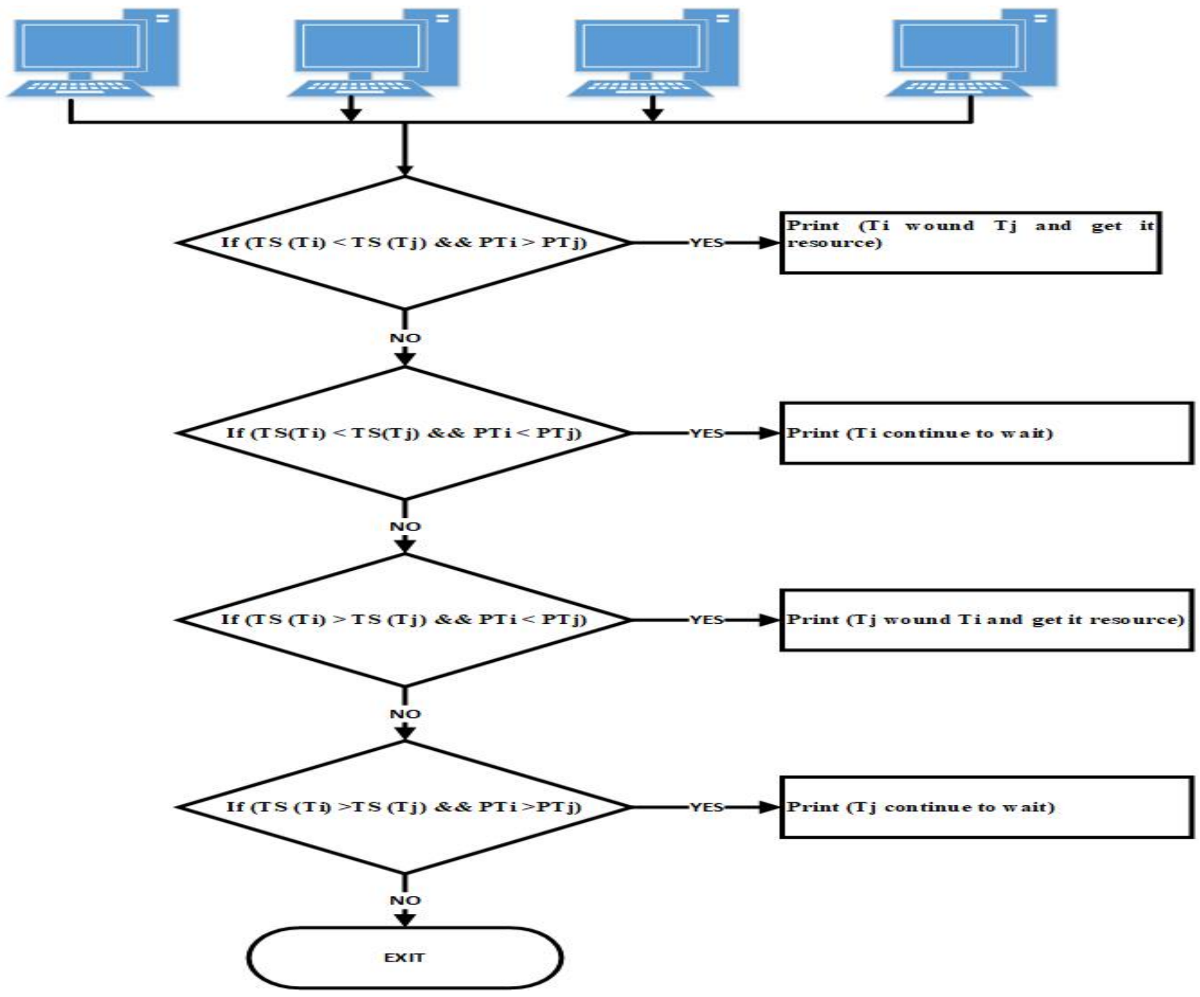

If (TS (Ti) < TS (Tj) && PTi > PTj)
YES
Print (Ti wound Tj and get it resource)
NO
If (TS(Ti) < TS(Tj) && PTi < PTj)
YES
Print (Ti continue to wait)
NO
If (TS (Ti) > TS (Tj) && PTi < PTj)
YES
Print (Tj wound Ti and get it resource)
NO
If (TS (Ti) >TS (Tj) && PTi >PTj)
YES
Print (Tj continue to wait)
NO
EXIT

Figure 4: Wound-Wait Algorithm

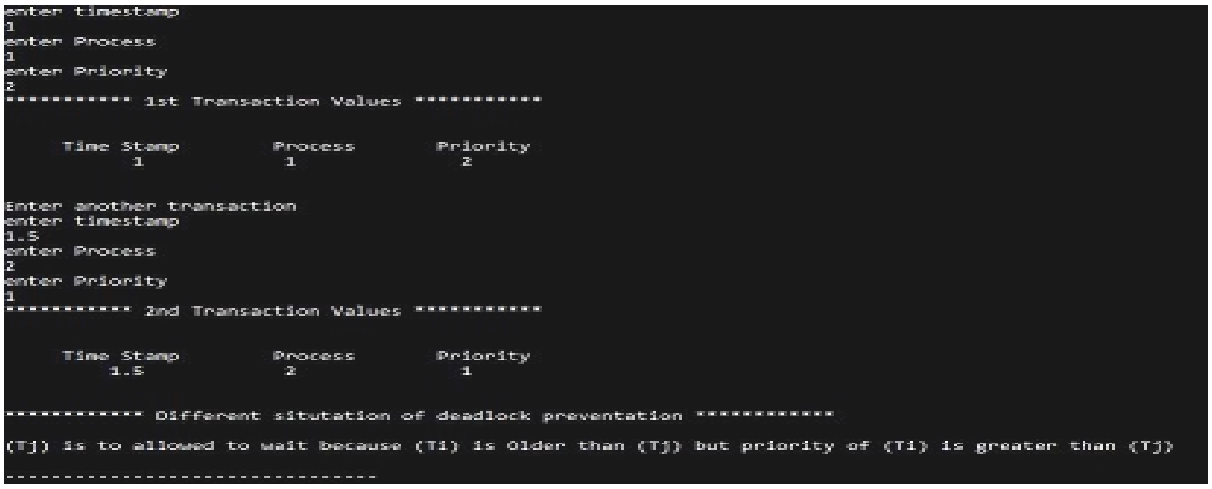
5. Implementation Of Proposed Algorithms

The proposed upgraded algorithms are implemented using C++ programming language and the tool used for coding is Dev C++ [14] and we create a small data set containing three fields like Time stamp, Process ID , Priority using these three fields to perform testing of proposed upgraded algorithms and the dataset used for testing the algorithms is show in the Table 1

Table 1: DataSet

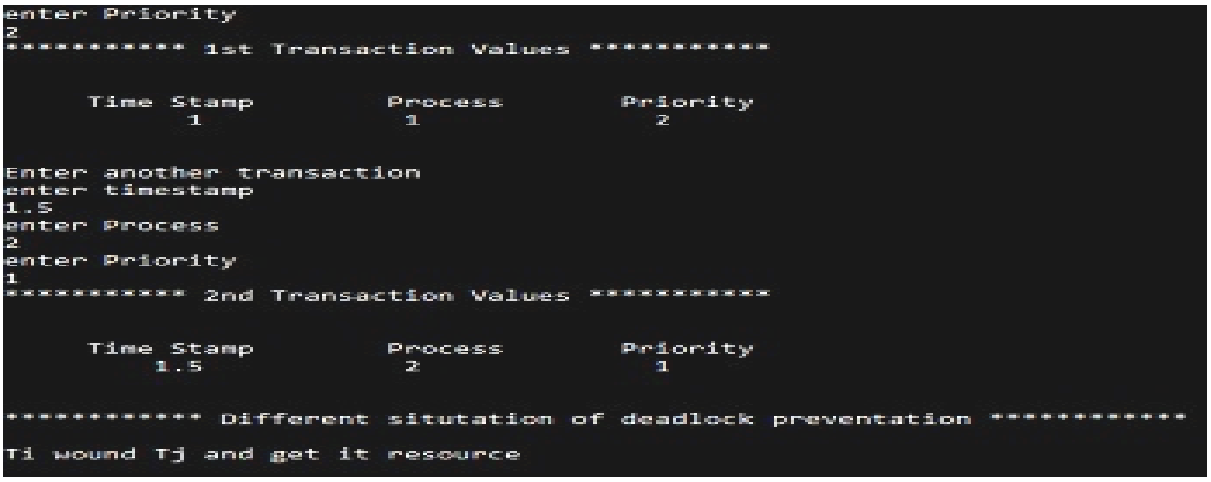
Time Stamp	Process ID	Priority
1	P1	2
1.5	P2	1
1.7	P3	3
1.9	P4	2
2.1	P5	4
2.3	P6	6
2.7	P7	2
2.9	P8	1
3	P10	3
3.1	P11	2
3.4	P12	8
3.6	P13	10
3.9	P14	11
3.2	P15	3
3.5	P16	16
3.7	P17	18
4	P18	19

Table 1 displayed Time Stamp, Process ID, and Priority. In this table, we take random values for testing the proposed upgraded algorithms. We take the float value for time stamp field and numeric values for Processes and for priorities and then put the values of these three fields one by one in the proposed upgraded algorithms and check its result. First, we put these three fields' values in the upgraded wait-die algorithm. We show the result of only one record of the proposed upgraded wait-die algorithm. Results after execution of wait-die algorithm show in the following Fig.Wait-Die Result and then we put these three fields' values in the upgraded wound-wait algorithm .we show the result of only one record of the proposed upgraded Wound-Wait algorithm. Results after execution of Wound-wait algorithm show in the following Fig.Wound-Wait Result.



```
enter timestamp
1
enter Process
1
enter Priority
2
*********** 1st Transaction Values ***********

     Time Stamp        Process       Priority
         1                1             2

Enter another transaction
enter timestamp
1.5
enter Process
2
enter Priority
1
*********** 2nd Transaction Values ***********

     Time Stamp        Process       Priority
        1.5               2             1

************ Different situtation of deadlock preventation ************

(Tj) is to allowed to wait because (Ti) is Older than (Tj) but priority of (Ti) is greater than (Tj)
--------------------------------
```

Figure 5: Wait-Die Algorithm



```
enter Priority
2
*********** 1st Transaction Values ***********

     Time Stamp        Process       Priority
         1                1             2

Enter another transaction
enter timestamp
1.5
enter Process
2
enter Priority
1
*********** 2nd Transaction Values ***********

     Time Stamp        Process       Priority
        1.5               2             1

************ Different situtation of deadlock preventation ************

Ti wound Tj and get it resource
```

Figure 6: Wound-Wait Algorithm

6. Conculusion

In this paper main, focus on the deadlock problem in the distributed system and evaluated the approaches like wound wait and wait die. The major problem exist in these algorithms is that they do not attend priority of processes. In the proposed upgraded wait die algorithm if the old process is to wait for the young process so the old process continues to wait but the priority of the older process is higher than younger process so the young process should be wait. Else, if the old process is to wait for the young process so the old process continues to wait but the priority of the young process is higher than Old process so the Young process continues to wait. In the proposed upgraded wound wait algorithm if the old process is waiting for young and the priority of the old process is higher than young process so old process wound young process and acquire its resource. Else, if the old process is to wait for young process and the priority of the young process is higher than the old process so the old process should be wait. These proposed upgraded algorithms attend to both priority and timestamp of processes figure number and caption should be typed below the illustration in 9pt and center justified.

Print References

[1] A. S. T. Maarten van Steen, “Distributed Systems,” pp. 1–596, 2017.

[2] Hardik Pandya, “Distributed Systems | Characteristics | Advantages | Disadvantages. | I’M FROSTY,” Hardik Pandya, 2018. [Online]. Available: http://www.imfrosty.com/2014/11/distributed-system.html. [Accessed: 27-Sep-2018].

[3] Chanaka Fernando, “The Evolution of Distributed Systems - DZone Cloud,” Chanaka Fernando, 2018. [Online]. Available: https://dzone.com/articles/the-evolution-of-distributed-systems. [Accessed: 28-Sep-2018].

[4] G. B. George Coulouris, Jean Dollimore, Tim Kindberg, DISTRIBUTED SYSTEMS Concepts and Design, Fifth Edit. Addison-Wesley, 2011.

[5] M. Abdoos, “Improved Deadlock Prevention Algorithms in Distributed Systems,” Int. J. Eng. Appl. Comput. Sci., vol. 02, no. 02, pp. 75–78, Feb. 2017.

[6] G. Dhiraj and V. K. Gupta, “Approaches for Deadlock Detection and Deadlock

[7] Y. Bhatia and S. Verma, “Deadlocks in Distributed Systems,” Int. J. Res., vol. 1, no. 9, pp. 1249–1252, 2014.

[8] D. Malhotra, “Deadlock Prevention Algorithm in Grid Environment,” vol. 02013, 2016.

[9] W. C. H. Cheng, “Using ordered and atomic multicast for distributed deadlock prevention,” Proc. - 1st Int. Symp. Object-Oriented Real-Time Distrib. Comput. ISORC 1998, vol. 1998–April, pp. 106–116, 1998.

[10] F. Tang, I. You, S. Yu, C. L. Wang, M. Guo, and W. Liu, “An efficient deadlock prevention approach for service oriented transaction processing,” Comput. Math. With Appl., vol. 63, no. 2, pp. 458–468, 2012

[11] S. Ghosh, “Distributed Systems: An Algorithmic Approach,” pp. 352–361, 2015.

[12] J. Wu, Distributed system design. 2017.

[13]Dev-C++ download SourceForge.net,” Slashdot Media.[Online]. Available: https: // sourceforge .net/projects/orwelldevcpp/. [Accessed: 21-Feb-2019].

[14] Formulating DNA Chains Using Effective Calculability, Syed Atif Ali Shah, INTERNATIONAL JOURNAL OF COMPUTER (IJC). http://ijcjournal.org

[15] Reengineering the Industrial CMMI, Syed Atif Ali Shah, Journal of Advances in Computer Engineering and Technology 4 (3), 1-10.