Modifications in Lamport Algorithm for Distributed Computing System

Taskeen Zaidi
Research Scholar, Deptt. of Computer Science
Babasaheb Bhimrao Ambedkar University
Vidya Vihar, Rae Bareli, Lucknow

Vipin Saxena
Professor, Deptt. of Computer Science
Babasaheb Bhimrao Ambedkar University
Vidya Vihar, Rae Bareli Road, Lucknow

ABSTRACT
In the current scenario, distributed approach of computing is very popular over the centralized approach of computing due to faster execution of processes; cut off the execution time of processes and cost. In the year 1978, Lamport [6] has proposed an approach for synchronization of processes under distributed environment which has the limitations for reordering and executing the events of the processes by using time, ordering of events and clock conditions. The important limitation is that the algorithm does not cover the process execution in reflexive, symmetric and transitive manners when the unidirectional or bidirectional ring is appearing in the distributed network for executing the processes and sharing the common resources under distributed environment.

The present work will focus on these aspects and processor can execute the events of processes either on its node called as computer system in the reflexive manner and if the current node is busy for other tasks then it can use the next promising node under the defined topology and can be executed by using symmetric property and if further second node is busy then events can be transmitted to next promising node and these are executed in the transitive manner and the output is transferred to the first node. These aspects are demonstrated by proposing a new kind of topology called as step topology in which numbers of computer systems are attached in the distributed network. Algorithms are designed for all these three cases by considering the definitions of process and thread. Since one computer system can interact with another computer system with message passing technique under distributed environment, therefore, message complexities in all these cases are also measured and compared with Lamport and other similar kinds of algorithms available for distributed computing system.

Keywords
Distributed approach, Bidirectional ring, Message Complexity, Step topology and Lamport algorithm.

1. INTRODUCTION
Due to evolution of multithreaded systems, the executions time of processes to be executed by using processors have been drastic reduced in comparison of the old centralized computing approach. Many of the computing labs have been converted towards the distributed computing approach to take benefits of distributed systems over the centralized systems. In the distributed computing system, a task or process is divided into number of subtasks or sub processes and executed in the mutually exclusive manner [1-4]. In the mutual exclusion if one process is using the critical section with well defined memory boundaries called as critical section and it will not allow the other process till the completion of previous process. The necessary conditions for mutual exclusion of processes are well defined in [5]. Under the distributed systems, the processes are arranged into a sequence of execution by using the scheduling algorithms. These follow the time, clock and ordering events which are well explained by Lamport [6]. Later on, the Lamport algorithm for mutual exclusion is further modified by Ricart and Agarwala [7] and processes are handled with the help of queue in which processes are arranged in a sequence of execution. In a reference of Maekawa [8], a distributed algorithm is designed for symmetric execution of processes and allows fully parallel operation used to solve the mutual exclusion by using the sets. If a process wants to execute by in the critical section then it will take permission from all other processes which are available in its sets [9]. A token based ring approach for the distributed algorithm is suggested by Suzuki and Kasami [10] and they have discussed that if any process wants to enter in its critical section then it will send a message to all other processes and also to that process which is currently holds the token and then that process sends the token to the requester processes.

The concept of graph theory is also used by the various researchers for the distributed algorithm and a tree concept is used by Raymond [11] in which it is discussed that if any process wants to enter in its critical section then first it submits its request to the parent node and then the request will be forwarded to the root and the token will be passed to the requester child, if it is at the top of the request queue. An efficient token based algorithm for mutual exclusion; in the distributed system is explained by Kawsar et al. [12] and they have explained that if any process desires to enter in its critical section then it will match its request to the stored copy of token, it is called as token ring approach having token timestamps, request hosts and node numbers. Many of the researchers have done the work on the unidirectional ring of attached computer systems but limited work is available for the bidirectional ring. Chakraborty and Yaprak [13] have suggested an approach to improve the reliability of token ring in bidirectional ring and found on the detection of ring breakage if it is related to single link failure.

In the present work, authors have proposed a new kind of network topology in the form of steps and called as static step

topology which can be used to establish static interconnection of computer systems called as nodes. Therefore, it is necessary to describe some of the important references related to network topology. Various kinds of network topology like star, tree, bus, mesh, hierarchical, etc are well explained by Hwang [4]. Minar [14] has explained various kinds of topologies like centralized, decentralized and hybrid topologies used for distributed computing for static internetworking and models. Androutsellis and Spinelis [15] have explained a survey related to point to point protocol used for network topology. Habib [16] has simulated the results for analysis of server placement for the execution of tasks under distributed environment. Zhang et al. [17] have explained the performance analysis of network topology in agent based connectivity architecture for the decision support system.

Many of the researchers have used modeling concepts to model the various aspects of distributed systems. In the current scenario and due to evolution of object-orientation, a well known Unified Modeling Language (UML) has also applied for distributed computing systems. It is a modeling language and used to model the various kinds of research problems. It contains various kinds of notations used to present real time structure of research problem. First time, Pllana and Fahringer [18-19] have suggested the various UML profiles for modeling the high performance applications and proposed various performance metrics for the parallel and distributed applications. A well known author Gomma [20] has used the concepts of UML for designing the concurrent, distributed and real time applications. Later on Saxena et al. [21-22] have used UML to propose various designs architecture for distributed system like multiplex system for parallel computation and mutual exclusion establishment for parallel tasks. Martinez et al. [23] have explained the modeling of communications protocols in details. The major performance attributes for the distributed system is the reduction in the execution time; Drozowski [24] has estimated the execution time for the various distributed applications using parallel processing and applied mathematical concepts. The performance of large multiagent system for distributed application has also been measured by Helsinger et al. [25].

In the present work, Unified Modeling Language is used to model a step topology for static interconnection of computer system and later on a segment is considered in the form of the triangle with the bidirectional ring and proposed the modification in Lamport algorithm. The processes or tasks are executed by taking the triangle in reflexive, symmetric and transitive manner. These three cases are explained in the form of algorithm; message complexities in all three cases are measured and compared with Lamport algorithm and other similar kinds of algorithms under distributed environment.

2. BACKGROUND

2.1 Distributed System

In the current scenario, distributed system is called as the collection of various heterogeneous devices like mobile system, computer system, i-pads, pda's, laptops, etc and they don't share the global clock. In the comparison of centralized computing system, distributed computing system has reduced the cost of infrastructure as well as usage of resources in the minimum time. The devices connected in the distributed system can be interacted each other by using the well known message passing technique supported by object-oriented technology and this is due to evolution of graphical user interface. The best example of distributed system is collection of above devices through wide area network and on the basis of this, Government of India has set up a National Knowledge Network (NKN) under Next Generation Networ (NGN) in India in the year 2009.The above concept of distributed system is demonstrated in figure 1 as shown below:

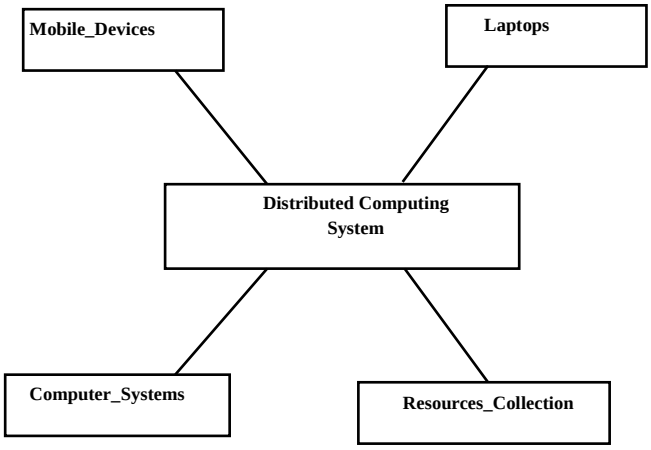

Fig. 1: A distributed system

2.2 Process

A process is defined as a collection of subprocesses, subtasks, subprograms, macro's, subroutines or a unit of task to be executed by the use of processor. The task can be executed by a unit called as a processing unit. The unit is used to execute the process and can be controlled by the object under object-oriented technology. Let us first define the object-oriented class of a process in terms of lines of code as given below:

```
Class process
{
public:
int process_id, process_size;
string process_in_time, process_out_time;
process_create();
process_delete();
process_update();
process_join();
process_suspend();
process_synchronize();
process_ack();
};
```

The class of process is controlled by an object and when it is called many times then the multiple instances are handled by the object.

2.3 Thread

The process in object-oriented programming language which consists of number of subprograms is controlled by threads. The threads are executed in a concurrent manner; follow the sequential flow of instructions and controlled by synchronization. The threads run simultaneously for the process and they have number of attributes and operations that are used during the execution of thread as described below:

```
Class thread
{
public:
int thread_id, thread_size, thread_priority;
string thread_in_time,thread_out_time;
thread_create();
thread_start();
thread_interrupt();
thread_terminate();
thread_resume();
thread_join();
thread_synchronize();
thread_execute();
};
```

The above class thread is again accessed by the object and if it is called many times then multiple instances are also controlled by the object.

2.4 Mutual Exclusion and Critical Section

A UML class diagram for accessing the critical section is shown in the figure 2 which consists of six UML classes namely Process, Thread, Communication_lines, Synchronize, Critical_section and Memory. In the proposed work, a new kind of step topology is considered and in the proposed topology, process may be executed through thread by using the message passing technique. Firstly, thread is assigned to a process and sends the request to access the critical section as shown by class name Critical_section. Then accessing of Critical_section is performed by the Thread class as shown in the figure. From the diagram, one can observed that when one process is inside the Critical_section through class Thread then it will not allow for accessing of the Critical_section by other processes and it is controlled by the Synchronize class. Threads assigned to processes move on the Communication_lines on network by using two techniques called as Point to Point (P2P) and Broadcast communication styles.

At the top of the line, the relation between the two classes is shown by the 1..1 and 1..* which may be one to one and one to many, respectively.

2.5 Process Migration

The process migration is related to transmit the process from one computer system to another computer system according to availability of the processor for execution of the process or sharing the resources under distributed computing system. If a data is available on one computer system and wants to migrate to another computer system by using the write back or write through cache then inconsistency occurs. For maintaining the cache consistency, write update and write invalidate policies are used for faster execution of the task and all these are controlled by using the message passing techniques.

2.6 Lamport Research work

In the year of 1978, Lamport [6] has described an algorithm for the execution of the processes for mutual exclusion under the distributed computing environment. The entire research work done by Lamport is based upon the distributed computing approach by considering the two processes P_i and P_j and recorded the start timings T_i and T_j in its own queue which is controlled by system clock and used the message passing technique for execution of the processes. The P_i process records its timestamp T_i in its own request queue and the process P_j records its timestamp T_j and puts the received message in its own queue. On the receipt of message, P_i acts if the request is at the front of request queue and P_j receives the messages from P_i, simply say $T_i<T_j$. Upon the receipt of the message, P_i and P_j release all resources from their own queue. The steps of the algorithm are shown below:

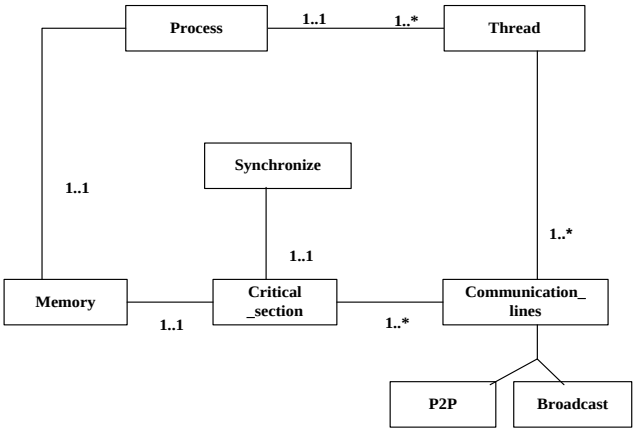

Fig. 2: A UML class diagram for mutual exclusion of process

The P_i process records its timestamp T_i by message passing technique in its own request queue. The algorithm proposed by Lamport [6] is given below:

1. The process P_j records its timestamp T_j by message passing technique and puts the received message in its own queue;

2. On the receipt of message, P_i acts according to following condition:

(a) P_i request's are at the front of request queue;
(b) P_j receives the messages from P_i after giving the timestamp reply to P_i i.e. Ti<Tj.

3. Now P_i and P_j release all the requests from their own queues.

From the above algorithm, it is observed that the algorithm does not cover when the resources are available on the other

computer system, and how it will receive the resources or incoming task can be executed on the other computer system called as node when one node is busy with some other tasks. In the proposed algorithms, all these aspects are covered and demonstrated in the subsequent sections.

3. PROPOSED MODIFICATIONS IN LAMPORT ALGORITHM

The Lamport algorithm is based on total ordering of events and it permits only one process has to access the resources at a time and has some limitations as described above. In the proposed work, the events of processes are executed in reflexive, symmetric and transitive manner i.e. first the events of process are executed on its own node called as computer system, if the current node is busy for some other processes then it use the next promising free node and executed the events of process and return the output in symmetric manner and if further, second node is busy then events of process can be transmitted to next promising free node and output is transferred to first node in transitive manner. Let us first describe some important assumptions taken into consideration for the proposed modifications in the Lamport algorithm. These are given below:

3.1 Ordering of Events of Task/Process

1. Let us consider e_1 and e_2 are two events of a process P, if e_1 comes before e_2 then $e_1 \longrightarrow e_2$;
2. Let e_1, e_2 and e_3 are three events of a process P, if e_1 comes before e_2 ($e_1 \longrightarrow e_2$) and e_2 comes before e_3 ($e_2 \longrightarrow e_3$) then e_1 comes before e_3 i.e. $e_1 \longrightarrow e_3$.

3.2 Clock Conditions for a Task/Process

i. If $C_1(e_1) < C_2(e_2)$, where, e_1 and e_2 are the events of process P then e_1 event will happen first in comparison of e_2, e_1 e_2, C_1 and C_2 are clock time of e_1 and e_2, respectively;

ii. If e_1 is the sending event of process P and e_2 is receiving event of process P then $C_1(e_1) < C_2(e_2)$, where C_1 and C_2 are clock time of e_1 and e_2, respectively.

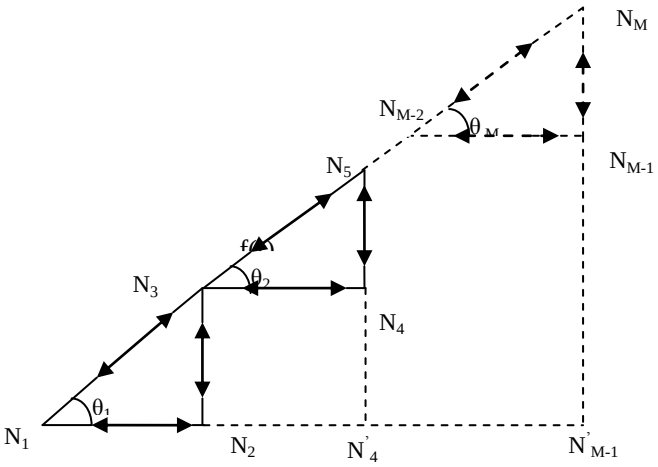

Fig 3: Representation of step topology

In addition to the above, the proposed work is based upon the new static kind of topology called as step topology as shown in the figure 3 which consists of the M computer systems called as nodes and a segment is considered and represented in the figure and it consists of the three computer systems represented as N_1, N_2 and N_3 as shown in the figure.

For the N_M computer systems, consider the length of segments as described below:

$N_1N_2=\alpha_1$, $N_1N'_4=\alpha_2$, …… $N_1N'_{M-2}=\alpha_{M-1}$, $N_1N'_{M-1}=r$

$\beta_1=\tan\theta_1$, $\beta_2=\tan\theta_2$,…….. $\beta_M=\tan\theta_M$

f(r) is the radial distance measuring the diameter of the step topology as described below:

$f(r)=1+\beta_1 r+\beta_2(r-\alpha_1)+\beta_3(r-\alpha_2)+\ldots\ldots+\beta_M(r-\alpha_{M-1})$

where r is the horizontal distance of the cable controlling the computer systems and measured in meters, while α_1, α_2, ……$=\alpha_{M-1}$ are the cable segments as shown in the above figure.

The other assumptions are given below:

1. Let us consider the computer systems N_1, N_2, N_3…....N_M are the finite number of computer systems arranged under distributed environment as shown in figure 3 with each one has local memory, cache and processor;

2. The above computer systems donot share the global clock and follow the structure of step topology of static interconnection as defined earlier;

3. Consider the subnet of three computer systems N_1, N_2 and N_3 from the above computer systems arranged under the step topology;

4. Let us consider the series of events of process P like e_1, e_2…..e_J, with timestamp tp_1, tp_2…….tp_J are recorded in the queue Q_1 and series of events of process Q say q_1, q_2…….q_K having timestamp tq_1, tq_2…..tq_K recorded in the queue Q_2 and series of events of process R say r_1, r_2…..r_L having timestamp tr_1, tr_2…….tr_L recorded in Q_3.

5. The above events are arranged by using the pipelining property as given below:

$e_1 \longrightarrow e_2 \longrightarrow \ldots e_J \longrightarrow e_J$ then $tp_1 < tp_2 < \ldots\ldots < tp_J$

$q_1 \longrightarrow q_2 \longrightarrow q_K \longrightarrow q_K$ then $tq_1 < tq_2 < \ldots. < tq_K$

$r_1 \longrightarrow r_2 \longrightarrow \ldots r_L \longrightarrow r_L$ then $tr_1 < tr_2 < \ldots\ldots < tr_L$

6. Consider the systems P, Q, R are idle at the initial stage and queues Q_1, Q_2 and Q_3 are the ready queues.

7. The processes P, Q and R create the logical ring in bidirectional manner as shown in figure.

8. The events of new incoming processes may appear on the computer systems and can join the logical ring according to the status of queue.

On the basis of above assumptions, the following cases arise for the execution of events of P, Q and R processes on nodes N_1, N_2 and N_3.

Case 1: Reflexive Execution of a Process

reflexive()
{
1. Create an object obj1 on class process (process obj1);
2. Assign obj1.process_id to a process P on node N_1(P ← obj1.process_id);
3. Set clock time for events of process P through obj1.process_in_time and recorded in queue Q_1 (e_1 ← obj1.tp_J …….. e_J ← obj1.tp_J);
4. Create an object obj2 on class thread (thread obj2);
5. Assign obj2.thread_id to a process P on node N_1(P ← obj2.thread_id)
6. for (i=1;i<=J;++i)
7. {
8. Synchronize the thread with processor (obj2.thread_synchronize());
9. if processor on self node N_1 is free then
10. {
11. Execute the process through thread (obj2.thread_execute());
12. Synchronize the thread for completing e_1,e_2,…. e_J (obj2.thread_synchronize());
13. After completing execution e_1,e_2,…….. e_J, terminate the thread (obj2.thread_terminate());
14. Acknowledge by the thread to process (obj1.process_ack());
15. Record process_out_time in queue Q_1 through obj1.process_out_time;
16. Empty the queue Q_1;
17. }
18. Repeat steps 1-17 for processes Q and R for N_2 and N_3 at the same time, respectively;

19. else goto Case 2: Symmetric Execution of a Process

20. }
}

Interpretation: The above case shows that the process P is executing on the self node N_1 and all the events e_1, e_2,……..,e_J of the process P execute on the node N_1. This case supports the ***reflexive property*** and events are executed through thread which synchronizes with the processor attached under the distributed environment as represented segment of nodes N_1, N_2 and N_3 as shown in figure 3. These events are executed by using the pipelining property as described above. The message complexity of the above case is O(N).

Case 2: Symmetric Execution of a Process

symmetric()

{
1. Create an object obj1 on class process (process obj1);
2. Assign obj1.process_id to a process P on node N_1(P ← obj1.process_id);
3. Set clock time for events of process P through obj1.process_in_time and recorded in queue Q_1 (e_1 ← obj1.tp_1…….. ….e_J ← obj1.tp_J);
4. Create an object obj2 on class thread (thread obj2);
5. Assign obj2.thread_id to a process P on node N_1(P ← obj2.thread_id)
6. if(N_2/N_3 is free) then
7. {
8. for(i=1;i<=J;++i)
9. {
10. Synchronize the thread with processor of next promising node N_2/N_3 (obj2.thread_ synchronize());
11. Execute the process through thread (obj2.thread_execute());
12. Synchronize the thread for completing e_1,e_2,…….. e_J (obj2.thread_synchronize());
13. After completing execution e_1,e_2,…….. e_J, terminate the thread (obj2.thread_terminate());
14. Acknowledge by the thread to process P to N_1 (obj1.process_ack());
15. Record process_out_time in queue Q_1 through obj1.process_out_time;
16. }
17. Repeat steps 1-15 for processes Q and R for N_1/N_3 and N_1/N_2 at the same time, respectively;
18. }

19. else

20. {goto next transitive case}

21.**}**

Interpretation-the above case shows that if node N_1 is busy, then process P is executed on next promising node N_2/N_3 and all the events $e_1,e_2,….e_J$ of process P are executed on node N_2/N_3 and after completing the execution of all the events, it returns the acknowledgement to node N_1. This case supports the ***symmetric property*** and all the events are executed through the thread which synchronizes the processor attached with the distributed environment as represented in figure 3.These events are executed using pipelining property as described above. The same is applicable for the other processes Q and R on nodes N_2 and N3, respectively which selects the next promising node N_1/N_3 and N_1/N_2, respectively. The message complexity of the above case is observed as O(N).

Case 3: Transitive Execution of a Process

transitive()

{

1. Create an object obj1 on class process (process obj1);
2. Assign obj1.process_id to a process P on node N_1(P ← obj1.process_id);
3. Set clock time for events of process P through obj1.process_in_time and recorded in queue Q_1 (e_1 ← obj1.tp_1…….. e_J ←obj1.tp_J);
4. Create an object obj2 on class thread (thread obj2);
5. Assign obj2.thread_id to a process P on node N_1(P ← obj2.thread_id)
6. if (N_1 is busy) then
7. {**goto symmetric case }**
8. else
9. {if (N_1 and N_2 are busy) then
10. {search for N_3 node
11. if (N_3 is free) then
12. {for(i=1;i<=J;++i)
13. {Synchronize the thread with processor (obj2.thread_synchronize());
14. Execute the process through thread (obj2.thread_execute());
15. Synchronize the thread for completing $e_1,e_2,$…….. e_J (obj2.thread_synchronize());
16. After completing execution $e_1,e_2,$…….. e_J, terminate the thread (obj2.thread_terminate());
17. Acknowledge to N_1 through thread to process (obj1.process_ack());
18. Record process_out_time in queue Q_1 through obj1.process_out_time;
19. }}}}
20. }

Interpretation- the above case shows the execution of the process for the transitive property which shows that if N_1 is busy then process will execute on the next promising node either N_2 or N_3 by using the symmetric property, otherwise if N_1 and N_2 are busy then it will search for the next node N_3 and execute the events $e_1,e_2,$…….. e_J of the processes in transitive manner and return the output to the node N_1. This case supports the transitive property and all the events are executed through the thread which synchronizes the processor attached with the distributed environment as represented in figure 3.These events are executed using pipelining property as described above. The message complexity of the above case is observed as O(N).

The comparison of the message complexity is summarized below in the following table and found that message complexities of the present work is matching with the existing algorithm as represented below:

Table 1. Comparison of the Message Complexity

Activity	Order of Message
Lamport [6]	O(N)
Ricart Agarwal [7]	O(N)
Maekawas [8]	$O(N^{1/2})$
Token Ring [9]	O(N)
Suzuki Kasamis [10]	O(N)
Raymonds [11]	O(Log N)
Fahims [12]	O(N)
Author's Algorithm (Reflexive/Symmetric/Transitive)	O(N)

The authors have also computed the message complexity for 24 computer systems attached through step topology as shown in the figure 1. In all the three cases i.e. reflexive, symmetric and transitive, it is of order of N. The computed results are shown in table 2 for all angles of 45^0 and horizontal cable segment is 90 meter; middle segments are controlled by $\alpha_1= \alpha_2=$ ….. α_{M-1} and taken as uniformly 10 meter. These results are also shown in the figure 4 and it is observed that message complexity is increasing as more computer systems are added in the network as shown in figure 3.

Table 2. Message Complexity of Proposed Algorithms ($\beta_1 =\beta_2 =,…. \beta_8 =1, \alpha_1= \alpha_2= …..= \alpha_8=10,r=90$)

Number of Nodes	Message Complexity Reflexive/Symmetric/Transitive
3	91
6	172
9	242
12	302
15	362
18	402
21	432
24	452

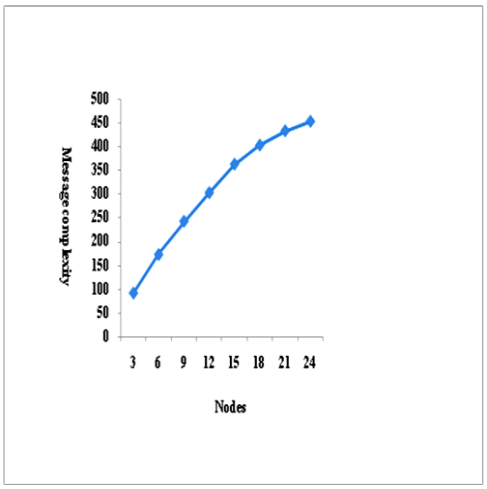

Fig 4: Representation of message complexity

4. CONCLUSIONS

From the above work, it is concluded that the Unified Modeling Language is a powerful tool and can be used to model the complex research problem. In the above mutual exclusion of execution of a process can be represented by the use of UML. It is observed that a well known researcher Lamport has published an algorithm for mutual exclusion for the distributed computing system which was applicable only for the self node or computer system. In the reference of this, the above work is extended when the numbers of the computer systems are attached according to the bidirectional ring and especially for the newly developed static step topology under distributed environment. The work is extended from the self node to the next promising node and execution of process is according to the symmetric property and further extended for the next promising node by using the transitive property. In this manner, the process may be executed inside the static step topology and in all the cases message complexity is of linear order. The sharing of resources for execution of a process is also done in the same manner.

The present work can be extended in many directions like for finding the node failures during the transferring of the messages from one computer system to another computer system. The loading, balancing and resilence issues are also the major areas for extending the proposed algorithms.

5. ACKNOWLEDGMENTS

The authors are very thankful to University Grants Commission for providing financial assistance to the Department of Computer Science, Babasaheb Bhimrao Ambedkar University, Lucknow to carry out the above research work.

6. REFERENCES

[1] Siberschatz, A. and Galvin, P.B., 2000, Operating Systems Concepts, 5th Edition, John Wiley and Sons, Inc. D.D.L.L.D.

[2] Siberschatz, A. and Peterson, J. L., 1988, Operating System Concepts, Addison–Wesley, Alternate Edition.

[3] Andrew S. Tanenbaum, 1995, Distributed Operating Systems, Prentice Hall.

[4] Hwang, K. 1993, Advanced Computer Architecture, McGraw-Hill Series in Computer Engineering, Inc Publishing.

[5] Milenkovic, M., 1997, Operating Systems: Concepts and Design,Tata Mcgraw-Hill.

[6] Lamport, L., 1978, Time, Clocks and Ordering of Events in a Distributed System, Communications of ACM, Vol. 21, No.7, pp.558-565.

[7] Ricart, G. and Agrawala, A., 1981, An Optimal Algorithm for Mutual Exclusion in Computer Networks, Communications of the ACM, Vol.24, No.1, pp.9-17.

[8] Maekawa, M., 1985, A sqrt(n) Algorithm for Mutual Exclusion in Decentralized Systems, ACM Transactions on Computer Systems,Vol.3, No.2, pp.145-159

[9] Agrawal, D. and El Abbadi, A., 1991, An efficient and fault tolerant solution for distributed mutual exclusion, ACM Transactions on Computer Systems, Vol.9, No.1, pp.1-20.

[10] Suzuki, I. and Kasami ,T.,1985, A Distributed Mutual Exclusion Algorithm, ACM Transactions on Computer Systems,Vol.3,No.4, pp.344-349.

[11] Raymond K.,1989,A Tree Based Algorithm for Distributed Mutual Exclusion, ACM Transactions on Computer Systems,Vol.7,No.1, pp. 61-77.

[12] Kawsar, F., Shaikot, S. H., Saikat, S. and Mottalib, M., A., 2002, An efficient Token Based Algorithm for Mutual Exclusion in Distributed System, Proceedings of 5th International Conference on Computer and Information Technology (ICCIT 2002), pp.93-96, Dhaka, Bangladesh.

[13] Chakraborty ,R. N. and Yaprak, E., 1993, Improve-ment in Reliability of the Token Ring Network by Reversal of Token in case of a Single Component Failure, Circuits and Systems, IEEE, Proceedings of the 36th Midwest Symposium on vol.2,Issue,16-18, pp.1152-1154.

[14] Minar N, 2002, Distributed System Topologies Part 1 and Part 2, Retrieved on June 19, 2012 from http://www.open2p.com/lpt/a/1461.

[15] Androutsellis –Theotokis S and Spinelis D, 2004, A Survey for peer to peer Content Distribution Technologies, ACM Press, New York ,U.S.

[16] Habib, S.J, 2005, Simulated Analysis of Server Placement on Network Topology Design, Proceedings of the 3rd ACS/IEEE International Conference on Computer Systems and Application, Cairo Egypt, pp.80-87.

[17] Zhang, H.L., Leung, H.C. and Raikundalia, G.K., Performance Analysis of Network Topologies in Agent Based Open Connectivity Architecture for DSS, Available online on Google Search Engine, Retrieved on June 19, 2012.

[18] Pllana, S. and T. Fahringer, 2002, On Customizing the UML for Modeling Performance Oriented Applications. In <<UML>>, Model Engineering Concepts and Tools, Springer-Verlag, Dresden, Germany.

[19] Pllana, S. and T. Fahringer , 2002,UML Based Modeling of Performance Oriented Parallel and Distributed Applications, Winter Simulation Conference, Retrieved on June 19, 2012.

[20] Gomma, H., 2001, "Designing Concurrent, Distributed, and Real-Time Applications with UML", Proceedings of the 23rd International Conference on Software Engineering (ICSE'01), IEEE Computer Society.

[21] Saxena, V. and Arora, D., 2008 "UML Modeling of a Protocol for Establishing Mutual Exclusion in Distributed Computer System, International Journal of Computer Science and Network Security,Vol.8, No.6, pp.227-235.

[22] Saxena, V., Arora, D. and Ahmad S., 2007, Object Oriented Distributed Architecture System through UML, Conference IEEE, International Conference on Advances in Computer Vision and Information Technology, ACVIT-07, ISBN 97881-89866-74-7, pp.305-310.

[23] Martinez Jesus, Merino Pedro and Solmeron Alberto, 2007, "Applying MDE Methodologies to Design Communication Protocols for Distributed Systems", IEEE Transactions of Software Engineering, April.

[24] Drozowski ,M, 2002, Estimating Execution Time of Distributed Application, Parallel Processing and Applied Mathematics, 4th International Conference PPAM, LNCS 2328, Springer-Verlag, pp 137-142.

[25] Helsinger, A., Lazarus, R., W., Wright, W. and Zinnky, J., 2003, Tools and Techniques for Performance Measurement of Large Distributed Multi Agent System, Proceedings of AAMAS 03 Conference, Australia, pp.843-850.