IOSR Journal of Mobile Computing & Application (IOSR-JMCA)
e-ISSN: 2394-0050, P-ISSN: 2394-0042.Volume 3, Issue 6 (Nov. - Dec. 2016), PP 01-06
www.iosrjournals.org

An Analysis of Resolution of Deadlock in Mobile Agent System through Different Techniques.

Rashmi Priya[1] , R. Belwal[2]
[1]*TMU, Research Scholar, Moradabad, India*
[2]*AIT, Professor and Head, Haldwani, India*

Abstract: *Mobile Agents are the set of processes that can move from one host to another host. The request granted by the client machine is executed by movement of mobile agents from a host machine to another host machine. The processes called by client are executed on a machine. The host machine must have all the resource needed to implement the service. This paper deals with allocation of resources through a proposed technique. In order to implement the tasks offered by the client, the mobile agents resume execution at the new machine. The extension of client and server model is followed in the development of Mobile agents. This paper also analyses the resolution of deadlock through different methods of study. The client only performs the operations provided by the server. There is scope of network scalable system when a particular server does not provide request desired by the client.*
Keywords: *Mobile Agents, deadlock, client, server, resource*

I. Introduction

As both sever and client keeps moving and interacting, there is a call for distributed applications. This is the reason an improvised technique for traditional approach of distributed computing is needed. Movements of agent are limited by imposing traditional distributed solutions into mobile agent systems as traditional solutions limit the movement of agents. There are many assumptions on which traditional distributed algorithms depend. Some of these are data location, communication mechanisms or network organization i.e. agent host location, number of nodes and connections between hosts.

Mobile agents are becoming more popular in areas of application development. Because of movement of both clients and servers freely through the network. Some of the fundamental assumptions of deadlock detection does not hold true. Hence new techniques and solutions and techniques are to be developed for mobile agents to solve their distributed coordination problems.

It is providing competitive environment to traditional distributed computing techniques.

II. Distributed Deadlock Detection ("Traditional Approach")

Deadlock detection properties are similar in case of Single processor Systems and Distributed Systems. The deadlocks are harder to detect, avoid, prevent and resolve because of the non-availability of centralized information and resource control [20]. These deadlocks are harder to detect. It becomes the responsibility of server to detect and resolve deadlocks in these cases of distributed systems. As the transactions are distributed and a number of servers are interconnected across multiple servers, detection process of deadlock becomes a cumbersome problem. In such situation detection of deadlock becomes complex problem as number of servers and transactions are interconnected and distributed across network through multiple servers. Under such scenario, distributed deadlock detection solutions can be divided into five categories: centralized, path-pushing, edge-chasing, diffusing computation, and global state detection. A number of solutions have been proposed to detect and resolve deadlocks in distributed systems. A number of solutions are achieved by optimizing number of messages or frequency of detection. Distributed deadlock detection is implemented simply using a centralized server to maintain the global wait-for graph. A local copy of the wait-for graph is added which is analysed on a periodical basis for each server in the network. In this technique, periodically each server in the network adds to the central server its local copy of the wait-for graph, which is analyzed for deadlocks [20]. As the single server can fail to handle many transactions this method does not follow fault tolerance. A local wait for graph helps to achieve deadlocks and it is extended to the global graph as per the situation in demand. . The global graph has no coordination at the central point as it is an extension to the local graph.

Based on the principle of a spanning tree of the global wait for graph Diffusing computation techniques are present. They represent processes and their links. The root process sends messages to its connected processes and if a deadlock exists it eventually receives a message [20].

Based on the similar concepts to edge chasing global state detection algorithms category are present. It holds little difference in the ways to detect deadlocks. In this technique the global wait-for graph is constructed from local graphs without blocking the computation. In such systems processes are linked by the

communication paths. Messages can be issued only by root processes; any non-root processes can not issue messages. This makes the root process special. All the processes, other than the root must wait until all the communication has been received by them.The deadlock detection technique based on the concept of global graph are path-pushing and edge-chasing.
The techniques of Path pushing show similar aspects. They differ in the manner in which information is sent to neighbouring nodes [22].
The other two categones of distributed deadlock detection are diffusing computation and global state detection. The technique developed by [20] is based on this paradigm.

There are two common criteria to evaluate Detection schemes .The two criterias are cited as 1) if an actual deadlock exists, it must be detected in a finite amount of time; 2) phantom deadlocks are not detected, i.e. the scheme must not find a deadlock that does not exist. As per the techniques used for single processor system there is no distributed deadlock detection which is universally detected., there is no universally accepted and deployed distributed deadlock solution, that is consistent with techniques used for single processor deadlock detection. Depending on the properties of an environment the two evaluation criteria must be applied and the most suitable scheme selected. Edge chasing [20] has an assumption that standard deadlock avoidance techniques are used for deadlock detection. These techniques are two phase transactions and resource locking.. There is a link between servers and the interconnected hosts, for which local wait- for graph is created and then converted to global graph and for each connection a permanent edge is created. In case of edge chasing whenever a transaction waits for another transaction which in turn is waiting with a resource on remote server, a probe is initiated. The probe contains the local wait-for graph of the server. The probe is then sent to the remote server which holds the transactions placed on the resource present on remote server. In case a distributed deadlock is present, a cycle is found in the global graph, which needs to be broken.

III. Distributed Deadlock Solution

3.1 Deadlock Shadow Agent Technique

The foundation and assumptions of this algorithm are presented first, followed by the detection and resolution phases of the algorithm. Finally, an example deadlock detection process is described in the section below. The following sections deals with topic of the thesis as proposed i.e., it deals with the solution to distributed deadlock detection.

Table 1. Agents Detection Table

Agent	Blocked Resource	Primary Locks
A11	Accounted	User1.db
A22	User1.db	Test1.db, Payroll1.db
A33	Test1.db	Account1.db
A66	Account1.db	Foo1.db

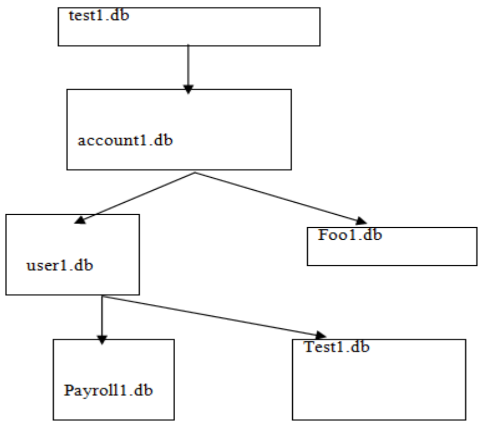


Fig.1. Detection Graph of Agents

The above criteria represents that there a cycle is detected in the graph.
The detection table creates the detection graph.
1. The root node is the resource where detecting agent is found.
2. This is account1.db. The primary locks entry from the next nodes in the graph..
3. These nodes create entry to the agents.
4. This is used to locate the root node.
For each child node as it is added to the graph. The above step is repeated.

Table 2. Deadlock Information

Agent Name	Blocked on Resource	Primary Locks
A1	R2(R2,A3,E2,2)	R1(R1,A1,E1,1),R4(R4,A1,E4,4)
A3	R1(R1,A1,E1,1)	R2(R2,A3,E2,2)

3.2 Premise and Assumptions

The solution follows the toplogy to be static in the beginning of the working of algorithm. A Mobile Ad-hoc Networks (MANET) type topology update protocol must execute in the background to keep routing tables synchronized w ith the actual topology. This rnovement is accomplished via node or resource based routing and the agent requests to be routed to a particular environment or resource. It is assumed that the host environment to route the agent on a best effort basis. This technique is similar to the routing scheme used in Intemet Protocol (IP) routers. The mobile agents of all types are independent from the network topology. The agents can move independent of nodes with no detailing of number of nodes and their connections. They can move independently through out the network.. This property allows host environments to cornrnunicate the details of Agent C. The assumption is taken that the Agent C can only lock and unlock the resources. It is assumed that the resource should be at the same environment physically. The agents manipulate the resource requests to the counterparts which detects the deadlock.
This solution assumes that agents must inform the host environment whenever any resource is blocked by them . Moreover ,here the properties of a mobile agent solution are presented. These properties are responsible for deadlock detection.
This assumptions provides the environment for the distributed deadlock detection in a mobile agent system.

The state of agent is communicated to deadlock detection agents. The additional requests desired by the blocked agents is not granted. The blocked agents during the deadlock detection process are available in the agent system . The agents are assigned identifier preceding the blocking if any done by Agent C. The agent identifiers can be assigned even when a resource is locked.

The host environment approves the agents request to unblock. A unique identifier is needed by the Agent to identify them uniquely. The agents can block other agents only when their request is granted by the host environment. It is assumed that two-phase commit or priority transactions are used . These are standard deadlock avoidance technique. An agent during the detection process will not unlock any resource instantaneously. These properties allow the agent to hold a resource. It helps to avoid any phantom deadlock.

According to this property locking of resources can be locked or unlocked when its physical presence is in the same environment. The locking and unlocking of resources are done by Agent C. The property stated above helps in preventing phantom deadlock detection significantly.

There should be strong understanding and cooperation between the agents and shared resources to allow the deadlocks to occur. As the agents perform their tasks resources can be "locked",that means their exclusivity is made to an individual consumer agent .

In addition the host is the ultimate authority and can allow or deny access to a resource .The assumption that during the locking process the Agent C must communicate with the host environment holds.

The host environment can deny the lock request depending on the tasks granted by an agent. It may block or wait on the resource .It may continue with the processing and its movement through the network. It is assumed that the blocking of agent is not done automatically. As the dynamicity of mobile agent could be disturbed by this.

IV. Proposed Solution Theoretical Measurements

Due to the asynchronous nature of the distributed deadlock detection technique, the events suggested in the example do not execute in a sequential manner. Many of the migrations and processing required during the deadlock detection process occur in parallel and exploit locality of reference. This property means that estimating the total run time of the technique should not be based on the number of migrations, since many occur at the same time. Factors such as host processor speed, deadlock implementation, network congestion and host environment efficiency have a significant impact on the tirne required to detect and resolve a deadlock.

Assuming that a deadlock condition exists and the assumed properties are true, the following steps must occur to detect the deadlock:
Detector 1 migrates to agent environment 2 and checks the locks for resource 1.
Detector 2 migrates to agent environment 1 and checks its locked resource, resource.
Detector 1 learns that Agent 2 is blocked on resource 1 and returns to its parent shadow agent. Detector 2 learns that Agent 1 is blocked on resource 2 and returns to its parent shadow agent.
Shadow Agent 1 checks the returned information, determines that there is no deadlock and adds Agent 2 (and its info) to its deadlock table. Shadow Agent 2 checks the returned information, determines that there is no deadlock and adds Agent 1 (and its info) to its deadlock table.
Shadow Agent 1 and Shadow Agent 2 re-initialize their detectors and restart the process.

Detector 1 migrates to agent environment 2 and checks the locks for resource 1.
Detector 2 migrates to agent environment 1 and checks its locked resource, Detector 1 learns that Agent 2 is blocked on resource 1 and Agent 1 is secondarily blocked on resource 1. Detector 2 that Agent 1 is blocked on resource 2 and Agent 2 is secondarily blocked on resource 2. Both detector returns to their parent shadow agents.
Shadow Agent 1 checks the retumed information, notices that Agent 1 was returned in the list of blocked agents and flags the deadlock. Shadow Agent 2 checks the returned information, notices that Agent 2 was retumed in the list of blocked agents and flags the deadlock.
Both Shadow Agents used a defined technique to determine that resource 1 must be unlocked.
Shadow I initializes its detector to unlock resource 1 and to notify Agent 2 when the resource is unlocked.
Detector 1 migrates and unlocks the resource. Agent 2 is notified. Detector 1 returns to its parent and notifies it of unlock completion.

Assuming a fault free environment, it is possible to obtain theoretical values for the number of agent migrations and number of lock checks during the deadlock detection process. In-order or sequential execution must be maintained to predict these theoretical values. For the purposes of this discussion, consider a simple distributed deadlock situation. In this situation two agents (Agent 1, Agent 2), resources (Resource I,Resource 2) and environments (environment 1, environment 2) are interacting to create a distributed deadlock.

Before starting an analysis of the preceding sequence of events, several key concepts must be defined. A migration is defined as the movement of a mobile agent from one environment to another. Migrations are unidirectional; therefore at least two migrations are required for an agent to check a lock and return.

A detector agent is defined as the number of messages required to visit all of the resources locked by a consumer agent and return. The number of migrations in a trip is network topology dependent, but is always greater than the number of resources to visit.

The migration of agent to a resource , is named as a visit. In a similar manner to a trip, the number of migrations required to complete a visit is network topology dependent. The minimum number of migrations in a visit is two but normally this value is higher.

The following variables can be defined to represent the various elements of the suggested distributed deadlock detection technique: Number of Agents in a Deadlock (NA), Number of Repetitions (NR), Detection Migration Volume (MD), Resolution MigrationVolume (MR) Messages in a Trip (MT)and Messages in a Visit (Mv).
Analysis of the steps required to complete the example deadlock detection case leads to the following general statements.
NR=NA
MD=NAXNRXMT
MR=Mv

In the example deadlock, NA is 2 and MT and M, are 2. Applying these values gives: 2 repetitions, 8 migrations for detection and 2 migrations for resolution. By checking the event sequence, these values match the actual numbers found in the example.

Additional analysis shows that since NR equals NA the detection migration volume can be restated as: N* x (MT), where N is the number of agents involved in the deadlock. It should be noted that these values are upper bounds for the volume of migrations and hence network load. They are not execution time estimates, since many factors not considered influence the time required to detect and resolve a deadlock using the suggested technique.

Lemma: The algorithm detects a deadlock within 2d time units.
Proof: When the initiator initiates the algorithm, it sends the CALL messages to all its successors which in turn propagates CALL message to its own successors. Consequently, a CALL message must travel from the initiator to the farthest leaf process of DST that requires at most d time hops in the worst case. Similarly, the leaf processes of DST must propagate the REPORT message that must travel along the edges traversed by CALL messages. As a result, a REPORT message must travel at most d hops before reaching the initiator in the worst case. Thus, the worst case time complexity of the algorithm is 2d, where d is the diameter of the WFG.

Proof: A process sends at most one CALL message on any of its outgoing edge and one REPORT message on any one of its incoming edges.
Therefore, the message complexity of proposed algorithm in the worst case is 2e, where e is the number of edges in the WFG.
Let us now measure the length of the control messages. The DSA carries the fixed number of process identifiers. Hence, the message length is a constant. Given Table compares the performance of proposed DSA with all distributed algorithms in the literature in terms of deadlock duration, message complexity, message size, and

resolution overhead in terms of number of messages. With all performance measures mentioned, it clearly shows that the performance of proposed algorithm is better or equal to the existing algorithms.

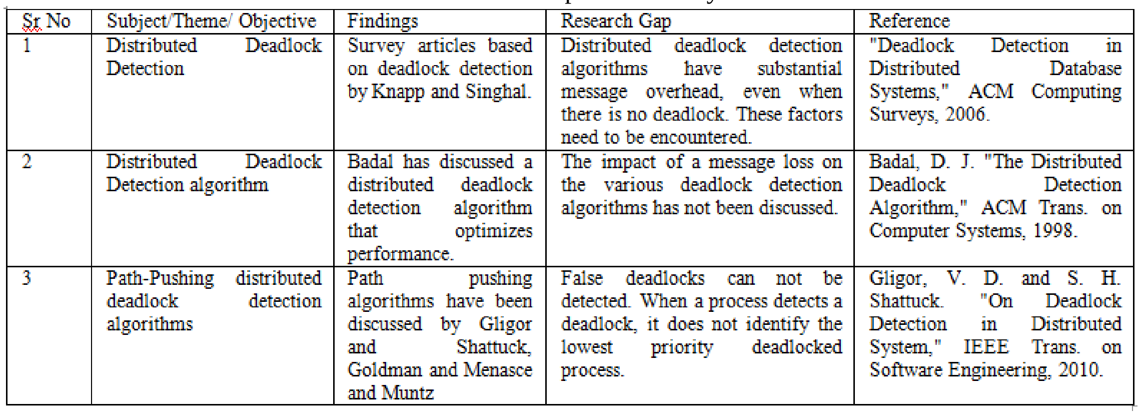
Table 3. Performance Comparison of Distributed algorithms for Detecting Generalized Deadlocks

Algorithms Comparison Factor	Bracha-Toueg's algorithm (1987)	Wang.et al's algorithm (1990)	Kshemkalyani and Singhal's algorithm (1994)	Kshemkalyani and Singhal's algorithm (1999)	DSA Deadlock Detection
Message Complexity	4e	6e	4e-2n+2l	2e	2e
Message Size	O(1)	O(1)	O(1)	O(e)	O(1)
Deadlock Resolution	No Scheme Message	No Scheme Message	No Scheme Message	No Scheme Message	1 Message

Table 4. Recent Research

Sr.No.	Papers	Objective/Research Gap
1	Deadlock Detection Based on Resource Allocation Graph(IEEE2009) Qinqin Ni , Sen Ma:	Uses principle of adjacency matrix. Random deadlock is difficult to detect
2	Formal Specification and Verification of the SET/A Protocol with an Integrated Approach (ACM2009)Vitus S.W. Lam and Julian Padget	SET/A protocol which is an agent-based payment protocol for credit card transactions in UML state chart diagrams is used. All properties are not satisfied when agent crashes.
3	Deadlock Avoidance Method for Multi-Agent Robot System Using Network of Chaotic Elements (IEEE2010) Yoichiro Maeda , Takayasu Matsuura:	Deadlock can be avoided by changing singleton values of fuzzy rule. The efficiency of proposed algorithm by using multi agent robot simulator has been shown needs to be tested.

Table 5. Comparative Study



Sr No	Subject/Theme/ Objective	Findings	Research Gap	Reference
1	Distributed Deadlock Detection	Survey articles based on deadlock detection by Knapp and Singhal.	Distributed deadlock detection algorithms have substantial message overhead, even when there is no deadlock. These factors need to be encountered.	"Deadlock Detection in Distributed Database Systems," ACM Computing Surveys, 2006.
2	Distributed Deadlock Detection algorithm	Badal has discussed a distributed deadlock detection algorithm that optimizes performance.	The impact of a message loss on the various deadlock detection algorithms has not been discussed.	Badal, D. J. "The Distributed Deadlock Detection Algorithm," ACM Trans. on Computer Systems, 1998.
3	Path-Pushing distributed deadlock detection algorithms	Path pushing algorithms have been discussed by Gligor and Shattuck, Goldman and Menasce and Muntz	False deadlocks can not be detected. When a process detects a deadlock, it does not identify the lowest priority deadlocked process.	Gligor, V. D. and S. H. Shattuck. "On Deadlock Detection in Distributed System," IEEE Trans. on Software Engineering, 2010.

V. Conclusion

The idea of distributed system is much broader and effective than centralized systems. There are many examples of distributed systems such as distributed databases, computer networks , real time process control systems and distributed information processing systems .There are many areas expanding the middleware due to the implementation of application-level protocols for communication. In case of distributed systems the tasks to control the operating systems are cumbersome; this is because database systems deadlock, mutual exclusion and concurrency control are difficult in case of centralized systems.

As the processes have used resources at various sites to improve throughput distributed systems is more vulnerable to deadlock occurrence. Hence there is a need to resolve deadlock by proper resource allocation. Further research needs to be initiated on detection of deadlock that requires lower cost and is more time efficient.

References

[1]. Almes, G.T., A.P.Black, E.D Lazowska,"The Eden System: A Technical Review", IEEE Transactions on Software Engineering,vol. 11, (1998) .

[2]. Bacor,J.M.,and K.G.Hamilton,Distributed Computing with RPC:The Cambridge Approach, " ACM Transaction on Computer Systems,vol.5,(2003) .

[3]. Nelson,B.J., "Remote Procedure Call", PhD thesis, Computer Science, Carnegie Mellon University, (2002)

[4]. Cheriton, D.R., "The V Distributed System", Communications of the ACM, Vol.30, (2000)

[5]. Lampson, B., “Remote Procedure Calls”, Lecture Notes in Computer science,vol.105,Springer Verlag,New York, (2004)
[6]. Badal, D. J. "The Distributed Deadlock Detection Algorithm," ACM Trans. on Computer Systems, (1998).
[7]. Bracha, G., and S. Toueg, "Distributed Deadlock Detection," Distributed Computing,(2000) .
[8]. Bracha, G., and S. Toueg, "A Distributed Algorithm for Generalized Deadlock Detection," Proc. Of the ACM Symposium on Principles of Distributed Computing, Aug. 1999.
[9]. Chandy, K. M., and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems," ACM Trans. on Computer Systems, 2003.
[10]. Chandy, K. M., J. Misra, and L. M. Haas. "Distributed Deadlock Detection," ACM Trans. on Computer Systems, 2005.
[11]. Chang, E., "Echo Algorithms: Depth Parallel Operations on General Graphs," IEEE Trans. on Software Engineering, July 2007.
[12]. Choudhary, A. L., W. H. Kohler, J. A. Stankovic, and D. Towsley, "A Modified Priority Based Probe Algorithm for Distributed Deadlock Detection and Resolution," IEEE Trans. on Software Engineering, 2008.
[13]. Elmagarmid, A.K., N. Soundararajan, and M.T. Liu. "A Distributed Deadlock Detection and Resolution Algorithm and Its Correctness," IEEE Trans. on Software Engineering, 2007.
[14]. 14. Gligor, V. D. and S. H. Shattuck. "On Deadlock Detection in Distributed System," IEEE Trans. on Software Engineering, 2010.
[15]. Goldman, B., "Deadlock Detection in Computer Networks," Technical Report MIT/LCS/TR185, MIT, 2006.