An Analysis and Improvement of Probe-Based Algorithm for Distributed Deadlock Detection

Kunal Chakma, Anupam Jamatia, and Tribid Debbarma

Abstract—In this paper we have performed an analysis of existing deadlock detection algorithm for distributed systems and did some improvement on them. The algorithm proposed in this paper is an extension of previous works with an introduction of an identity set S in the probe initiation. The rate of dependency table clearance is determined by our algorithm, and how over-killing of processes can be avoided by using one additional data-structure is also shown by our algorithm. The algorithm shows how this data-structure should be updated with probe-messages with the identity set, and how it helps in determining which process should be selected as victim for resolving the deadlock, and which entries should be cleared in the dependency table after successful detection of a deadlock. The study indicates that in this algorithm, rate of probe initiation is a dominant factor in determining system performance and rules need to be framed for determining best value of it.

Index Terms—Probe, process, site, wait-for graph, controller.

I. INTRODUCTION

A deadlock is a condition in a system where a process cannot proceed because it needs to obtain a resource held by another process but it itself is holding a resource that the other process needs. The same conditions for deadlocks in centralized systems apply to distributed systems. Unfortunately, as in many other aspects of distributed systems, they are harder to detect, avoid, and prevent. Dead locks are a fundamental problem in distributed systems. Deadlock detection is more difficult in systems where there is no such central agent and processes may communicate directly with one another. Deadlock detection and resolution is one among the major challenges faced by a Distributed System. A Distributed database system (DDBS) is a collection of databases distributed over several sites interconnected by a communication network. It provides a resource-sharing environment where database activities can be performed optimally both in global and local framework. The distributed nature of database demand full proof control structure for its proper and effective functioning.

Therefore, the allocation of the resources should be properly controlled otherwise it may lead to several anomalies such as concurrency of transaction, synchronizing of events and deadlocks called local, on the other hand a distributed transaction involve resources located at several sites. The objective of the concurrency control is to allow concurrent

executions without violating the consistency of the database. Various concurrency control protocols have been proposed [1]-[4]. Among them two phase locking [1] is the most commonly adopted in the design of the commercial products. In two phase locking, the conflicts resulting from sharing of data objects are resolved by setting locks on the data objects. One of the major problems of two phase locking is that the possibility of deadlock resulting from cyclic-wait for locks among different transactions. Deadlock is undesirable because the transactions involved in deadlock cycle are blocked permanently. The resulting system performance is thus dramatically degraded.

In distributed database system deadlock detection is very complex as a result of lacking global system state. Although different deadlock detection algorithms in distributed system have been proposed [4], [5], it has been found that some algorithms fail to detect deadlock under certain cases and some detect false deadlock [6], [7]. Different approaches have been adopted in distributed deadlock detection. Some of them try to construct a global system state [8] and the other try to pass a special message through blocked transactions in order to find out a deadlock cycle [9], [10]. Such approach is called probe-based distributed deadlock detection as proposed by [5], [11]. The main feature of this approach is that no global system state is needed.

Although many literatures refer and modify [5], [11], most of them have neglected its performance issues and not its correctness. Ref. [12] presented an improvement over the algorithm proposed by [11] in which deadlock is found by passing special messages called probe messages along the edges of a wait-for graph. As per [12] it has been found that in cases where deadlock cycles are repeatedly formed among different transactions, algorithm in [11] fails completely. [13], [14] is a study of various techniques. In this paper, an analysis and extension of [12] was done to address the weakness of algorithm in [11]. The remainder of this paper is organized as follows. Section II presents deadlock models. Section III describes existing algorithms. Section IV describes algorithm in [11] and its deficiency. Section V is the modified deadlock detection algorithm. Lastly, the conclusions and future research work are presented in Section VI.

II. DEADLOCK MODELS

In this section we give a short overview of the existing deadlock models.

A. Resources / Communication Deadlock Models

In this model, deadlocks are divided into two different

Manuscript received July 2, 2014; revised September 15, 2014.

K. Chakma is with the National Institute of Technology Agartala, 799046, India (e-mail: kchakma@nita.ac.in, kchax4377@gmail.com).

types: resource deadlocks and communication deadlocks.

1) Resource deadlocks

In resource deadlocks, tasks access resources (e.g. threads access shared memory), and the resources have to be acquired before they are used. If a task cannot lock the desired resource, then it has to wait until the resource is freed by the task which currently has the lock on the resource. Trying to match this scenario to a graph type, we realize that we have two kinds of nodes resources and tasks. The locks are represented by directed edges. Therefore, resource deadlocks can be detected using a Task-Resource Graph (TRG). If tasks hold the resources and therefore the tasks themselves have to be locked, we end up with a graph which has only nodes of type task, which corresponds to a Task Wait-For Graph (TWFG). Deadlock detection with resource deadlocks corresponds to cycle detection in The TWFG (transition wait for graph).

2) Communication deadlocks

In communication deadlocks, the resources that tasks are waiting for are messages (or synchronization between tasks), and not shared memory. Also a cycle detection algorithm is not suitable for detecting deadlocks in this scenario; more advanced algorithms are necessary.

B. General Resource System Model

The general resource system model was introduced by [11]. It merges resource deadlocks with communication deadlocks. This model uses a General Resource Graph (GRG).

C. Hierarchical Deadlock Models

Edgar Knapp in [7] proposed a hierarchical set of deadlock models to describe characteristics of deadlocks in a system. The models proposed by him range from very restricted models of request forms to completely unrestricted models.

In the following sections, the most important models will be discussed.

Single-resource model: The simplest possible model is the single-resource model. Each task can only have one outstanding request and therefore, in a directed graph, only one out going edge. In this scenario, a Task-Resource Graph (TRG): is sufficient and deadlocks can be detected if there is a cycle in that graph. All deadlocked tasks can be obtained by getting all tasks in a cycle and by adding all tasks that can only reach deadlocked tasks.

AND model: The AND model allows tasks to request a set of resources and the tasks are blocked until they have acquired all requested resources. So each task can have multiple outgoing edges. As in the single-resource model, it is still sufficient to build up a task-resource graph (TRG): and detect cycles in that graph.

OR model: In the OR model, tasks are blocked only until one of the requested resources has been acquired in contrast to the AND model. This model is useful for example in Databases which have data replication and a task only needs read access to one copy of the desired resource. In contrast to the AND model, detecting a cycle in a General Resource Graph (GRG): is not a sufficient condition for detecting deadlocks anymore. Instead, detecting deadlocks in the OR model can be reduced to finding a knot² in the GRG. Therefore, a task T is deadlocked if it is in a knot or T can only reach deadlocked tasks.

III. EXISTING ALGORITHMS

Goldman's algorithm in [1] exchanges deadlock related information in the form of an ordered blocked process list (OBPL), in which a process (except the last) is blocked by a successor. The last process in an OBPL may either be waited to access the resource or be running. The algorithm detects deadlock by repeatedly expanding the OBPL, appending it to the one that holds the resource needed by the last process in the list until either a deadlock discovered (that is last process is blocked by the process in the list) or OBPL is discarded (the last process is running). An advantage of [1] is that it does not require continuous maintenance of Task Wait-for graphs (TWFG). It constructs an OBPL whenever deadlock detection is to be carried out. However, it requires that every process have at most one outstanding resource request.

Isloor Marsland algorithm in [15] detects deadlock at the earliest possible deadline based on the concept of reachable set. Reachable set is the set of all nodes that can be reached from state graph. A process is deadlocked if the reachable set of the corresponding node contains the node itself. This algorithm detects deadlock by constructing reachable set and checking whether any node belong to its own reachable set. Here every site maintains the system state graph and reachable sets for each node. Whenever a process is made to wait for a long resource or wherever the process releases a resource corresponding info is broadcasted to all other sites. If r changes occur in the state graph then the algorithm requires $r(N-1)$ messages per sec for deadlock detection.

Menasce and Muntz algorithm in [6] propagates only the two end points of a directed path, rather than the whole path to detect deadlocks. Blocking (T) of a transaction T is the set of all non-blocked transactions that can be reached from T by following TWF path. Reachable set is set of all nodes that can be reached from state graph. A process is deadlocked if the reachable set of the corresponding node contains the node itself. When a transaction T gets blocked then for each transaction in i^{th} T in the blocking set (T), algorithm sends the blocking set i^{th} of T and j^{th} of T to home sites respectively.

In Obermarck's algorithm in [8], the nonlocal portion of the global TWF graph at a site is abstracted by distinct node called external or Ex, which helps determine potential multisite deadlock without requiring a huge global TWF graph to be stored at each site. A site waits for deadlock related information from other sites. Site combines the received information with the local TWF graph detects all the cycle and breaks those which do not contain the node Ex-Algorithm reduces the message traffic by ordering nodes in lexicographic ordering.

Chandy- Misra-Hass algorithm in [11] uses:

- 1) probe message (i, j, k). Deadlock detection is initiated for process P_i and is sent by the site of P_j to the site of P_k .
- 2) boolean vector dependent(i, j) is kept at the site of P_i , which is true only if P_i knows that P_j is dependent on P_i .

Algorithm:

Deadlock initiation at P_i

if P_i is locally dependent on itself, then deadlock
else
for all P_j and P_k if

P_i is locally dependent on P_j
 P_j is waiting for P_k
 P_k runs on another site
 send a probe (i, j, k) to the site of P_k
 Upon receipt of a probe (i, j, k) at the site of P_k
 if P_k is blocked, and
 dependent(k, i) is false, and
 P_k has not replied to all the requests of P_j ,
 then
 set dependent(k, i) = true
 if $k=i$ then deadlock
 else send a probe (i, m, n) to the site of P_n , if
 P_k is dependent on P_m , and
 P_m waits for P_n , and
 P_m and P_n are on different sites

IV. LIMITATIONS OF CHANDY'S ALGORITHM

A. Example of Chandy's Algorithm and It's Deficiency

The algorithm in [11] is based on passing probe through different sites. Each site has a controller which maintains the Boolean array $Depend_k$, known as dependency table k , for each constituent process p_k , where $Depend_k(i)$ is true only if p_k 's controller knows that p_i is dependent on p_k . If $Depend_i(i)$ is true, then P_i is dependent on itself and hence is deadlocked. Whenever the constituent process is idle, the probe computation will be initiated periodically by the controller. It has been found that some deadlock may not be detected by their algorithm for some cases when the degree of data contention is high. The following example illustrates the problem.

1) System state at time T_0

We assume the initial system state to be at T_0 . This is the same example stated in [12]. There is a deadlock cycle initially involving processes $P_1, P_2, P_3, P_4, P_5, P_6$ and P_7 in the system. When P_6 receives the probe (6, 4, 6) a deadlock is detected. P_6 is selected as victim and is aborted. All the data objects being held by P_6 will be released. This is illustrated in Fig.1.

Fig. 1. Original chandy-misra algorithm.

2) System state at time T_1

Let us assume that at T_1 , after P_6 leaves, P_7 is granted the resource on which P_4 is waiting and later it (P_7) depends on P_5 . Although there is a deadlock (with the process P_1, P_2, P_3, P_4 and P_5) in the system, this deadlock will not be detected by the algorithm.

According to [11], only the process that is located at

boundary of the site and receiving probes will be declared deadlocked. Therefore in this example, only processes P_1, P_3 and P_7 are the possible victims. However, all the probes from these three processes will not be able to detect the deadlock. Probe (1, 2, 3) will be initiated by the controller C_1 on behalf of P_1 . Controller C_2 will eventually discard this probe when P_3 is receiving it. P_1 is Positive entry in the dependency table of P_3 ($Depend_3(1)=true$). Probe (3, 4, 7) and Probe (7, 5, 1) will also be discarded by controller C_1 with the same reason. No Probe can be propagated along the wait-for path, and so this deadlock cycle exists permanently and will not be detected. This is illustrated in Fig. 2 - Fig. 5.

V. IMPROVED ALGORITHM

Here we present the improvement of algorithm in [11] by enhancing [12]. Problem in [11] is that when a deadlock is resolved, the information in the dependency table is not cleared and such information is used for detecting deadlock. Therefore, the problem found in [11] can be rectified by

clearing the dependency table periodically so that the probes have to be re-transmitted and will not be discarded.

From the last example, suppose the dependency table of P1 and P3 are cleared at time T1. Subsequently the deadlock cycle can be detected. Now suppose P1 initiates Probes (1, 2, 3). Then P3 receives the probes and propagates to P7 by Probe (1, 4, 7). P7 will propagate Probe (1, 5, 1) to P1 which will eventually be declared deadlocked.

As stated in [12] if the dependency table is being cleared frequently, the number of probe messages will be high as many probes needed to be re-transmitted. The system may be frozen at deadlock state for a long time before the deadlock is detected if the dependency table is being cleared sparsely. The number of probe initiations can then use to estimate the blocking period of the process when the controller initiates probes periodically. Therefore, the number of probe initiations can be used to determine when to clear the dependency table.

There is one more problem about deadlock in the old algorithm:

In old algorithm, the deadlock is resolved by letting the process that initiated the probe commit suicide. However, this method has problems, if several processes invoke the algorithm simultaneously. In our example, imagine that both process 1 and 6 block at the same moment. In such a situation eventually both will detect deadlock, and each would kill itself.

In our alternative algorithm, a set S , called the identity set, will be added to each probe to the end of probe message. So, when probe will return to initial sender, the complete cycle would be listed.

DEADLOCK DETECTION IN DISTRIBUTED SYSTEMS

Fig. 6. Probe (1, 2, 3, S).

The sender now can see, which process has the highest number, and kill that one or send a message to kill itself. In either case, if multiple processes discover the same cycle at the same time, they will choose the same victim. This feature of adding identity to the end of probe message will also help in deciding rate of dependency table clearance. Thus, whenever a deadlock is detected then only the dependency tables will be cleared, and which dependency tables to clear, this information can also be obtained from the final probe message received. All controllers (of different-different sites) present in the final probe message clear their dependency table and hence will be able to detect all further deadlocks. So, rate of dependency clearance has been identified and is very low (i.e. equal to rate of deadlock occurrence). For illustration, let us suppose that P1 initiates Probes (1, 2, 3, S). This will be accepted as *depend* [1], [3] = false (Fig. 6). P3 receives the probes and propagates to P7 by Probe (3, 4, 7, S). Probe (3, 4, 7, S) is also accepted as *depend* [3], [7] = false (Fig. 7). P7

will propagate Probe (7, 5, 1, S) to P1 which is also accepted as *depend* [1], [7] = false (Fig. 8). Eventually the system is declared deadlocked as shown in Fig. 9.

Performance:

- No. of messages is $m \times (n-1)/2$, where m is no. of processes and n is no. of sites involved in the deadlock.
- Message size is very small (slightly more than size of the original algorithm).
- Detection delay is $O(n)$.

DEADLOCK DETECTION IN DISTRIBUTED SYSTEMS

Fig. 7. Probe (3, 4, 7, S).

DEADLOCK DETECTION IN DISTRIBUTED SYSTEMS

Fig. 8. Probe (7, 5, 1, S).

DEADLOCK DETECTION IN DISTRIBUTED SYSTEMS

Fig. 9. Deadlock detection.

VI. CONCLUSION AND FUTURE WORK

As we have seen above in section 3, a lot of deadlock detection techniques have proved to be successful but there are a number of flaws too related to these techniques. Like in case of [11], it may detect false deadlock. Even in this algorithm a lot of performance degradation is seen whereas if we compare it to the algorithm proposed in [12], we see that the system suffers very little performance degradation from the additional overhead for the clearance of dependency table. Dependency tables should be cleared whenever any deadlock is detected and resolved. We have enhanced the implementations by introducing the *identity set* S in our proposal. The identity set helps in deciding the rate of dependency table clearance. Thus, whenever a deadlock is detected then only the dependency tables will be cleared, and

which dependency tables to clear, this information can also be obtained from the final probe message received. All controllers (of different-different sites) present in the final probe message clear their dependency table and hence will be able to detect all further deadlocks. So, rate of dependency clearance has been identified which is comparatively low.

More study can be done on the effect of maintaining the new data-structure on the system performance and how this data-structure can be modified to use least system resources. More extensive study on enhancing deadlock detection efficiency is required.

REFERENCES

- [1] A. Bernstein and N. Goodman, "Concurrency control in distributed database system," *ACM Comp. Sur.*, vol. 13, no. 2, pp. 185-221, June 1981.
- [2] A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Reading, Massachusetts: Addison-Wesley, 1987, ch. 3, pp. 47-87.
- [3] S. Ceri and G. Pelagatti, *Distributed Databases Principles and Systems*, New York: McGraw-Hill Book Company, 1984, ch. 8, pp. 209-226.
- [4] H. T. Kung and J. T. Robinson, "Optimistic methods for concurrency control," *ACM Trans. Database Syst.*, vol.6, pp. 212-226, June 1982.
- [5] K. M. Chandy and J. Mishra, "A distributed algorithm for detecting resources deadlock in distributed systems," in *Proc. the ACM Symposium on Principles of Distributed Computing*, Ottawa, Canada, Aug. 1982, pp. 157-164.
- [6] D. E. Menasce and R. R. Muntz, "Locking and deadlock detection in distributed databases," *IEEE Trans. Software Engineering*, pp. 195-202, May 1979.
- [7] E. Knapp, "Deadlock detection in distributed databases," *ACM Computing Surveys*, vol.19, no. 4, pp. 303-328, December 1987.
- [8] R. Obermarck, "Distributed deadlock detection algorithm," *ACM Trans. Database Syst.*, vol. 7, no. 2, pp. 202-223, June 1982.
- [9] S. Ceri and G. Pelagatti, *Distributed Databases Principles and Systems*, New York: McGraw-Hill Book Company, 1984.
- [10] R. C. Holt, "Some deadlock properties on computer systems," *ACM Computing Surveys*, vol. 4, no. 3, pp. 179-196, September 1972.
- [11] K. M. Chandy, J. Mishra, and L. M. Hass, "Distributed deadlock detection," *ACM Trans. Comput. Syst.*, vol. 1, pp. 143-156, May, 1983.
- [12] C. F. Yeung, S. L. Hung, K. Y. Lam, and C. K. Law, "A new distributed deadlock detection algorithm for distributed databases systems," in *Proc. TENCON '94. IEEE Region 10's Ninth Annual International Conference*, 1994, pp. 506-510.
- [13] H. Grover and S. Kumar "Analysis of deadlock detection and resolution techniques in distributed database," *International Journal of Computer Engineering & Science*, pp. 17-25, 2012.
- [14] D. Gupta and V. K. Gupta "Approaches for deadlock detection and deadlock prevention for distributed systems," *Research Journal of Recent Sciences*, pp. 422 -25, 2012.
- [15] S. S. Isloor and T. A. Marsland, "An effective on-line deadlock detection technique for distributed database management systems," in *Proc. Compsac 78*, Nov. 1978, pp. 283-288.

Kunal Chakma is an assistant professor in Computer Science and Engineering Department at National Institute of Technology Agartala, Tripura, India. He obtained his bachelor of technology in computer science & engineering from National Institute of Technology Jalandhar, India and master from National Institute of Technology Agartala, India. His fields of interests are system software and distributed systems.

Mr. Chakma is a professional member of Association for Computing Machinery.