Performance Evaluation of A New Distributed Deadlock Detection Algorithm

Chim-fu Yeung, Sheung-lun Hung and Kam-yiu Lam

Department of Computer Science
City Polytechnic of Hong Kong
83 Tat Chee Avenue, Kowloon Tong, Hong Kong
email: {cscfjims|cshung|cskylam}@cphkvx.cphk.hk

Abstract

In this paper, a new probe-based distributed deadlock detection algorithm is proposed. It is an enhanced version of the algorithm originally proposed by Chandy's et al. [5,6]. The new algorithm has proven to be error free and suffers very little performance degradation from the additional deadlock detection overhead. The algorithm has been compared with the modified probe-based and timeout methods. It is found that under high data contention, it has the best performance. Results also indicate that the rate of probe initiation is significantly reduced in the new algorithm.

1 Introduction

The objective of concurrency control is to allow concurrent execution of transactions without violating the consistency of the database [10]. Various concurrency control protocols have been proposed [1,2,4,13]. Among them two phase locking [1] is the one most commonly adopted in the design of 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 problem 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 systems, deadlock detection become very complex as a result of uncertainties in the global system state. Although many deadlock detection algorithms in distributed database systems have been proposed [5, 6, 7, 14], most of them are impractical because of high system overheads. Two main approaches have been adopted in distributed deadlock detection. The first one is to construct a global system state [14] and the second is to try to pass a special message through blocked transactions in order to detect a deadlock cycle [5, 6, 7]. One method of the latter approach is the so called probe-based distributed deadlock detection as proposed by Chandy, Misra and Haas [5,6]. The main feature of this method is that no global system state is needed.

Chandy's deadlock detection algorithm is based on passing probes through different sites. Only processes residing in site boundaries (process which are in direct communication with processes in another site) can initiate probe messages [5,6]. Chandy's algorithm can fail to detect deadlocks as a result of boundary process abort [3]. A modified version of Chandy's probe-based distributed deadlock detection algorithm (MPA) was proposed in [3]. As a result of the high system overheads incurred in maintaining the dependency table for MPA, system performance is expected to be a major problem. An enhanced version of MPA (EPA) is introduced by replacing the dependency table with a local wait-for graph. This paper compares the performance of timeout, MPA and EPA through extensive simulation studies. As anticipated, EPA outperforms both MPA and timeout under most conditions, especially under a high data contention environment.

The remainder of this paper is organized as followed. Section 2 presents the modified as well as the enhanced probe-based distributed deadlock detection algorithm. The simulation model and workload parameters for the set of experiments are presented in section 3. Section 4 are the results and their interpretation. Lastly, the conclusions and future research work are presented in section 5.

2 The Modified and Enhanced Probe-based Algorithms

The problem found in Chandy's algorithm concerning boundary process aborts can be rectified by eliminating the need to define boundary processes. Probes are transmitted whenever a transaction is blocked while another remote transaction is involved in the wait path. The major modifications to the original algorithm for MPA are:

(1) In the original algorithm, only boundary transactions can send probe messages, and this limits the deadlock detection capability of the system (certain type of deadlocks will escape detection). The modified algorithm includes all the involved transactions in the sending of probe messages.
(2) The modified algorithm will set all the dependency table entries for every transaction involved in the wait path instead of the first element only [3].
(3) As a transaction cannot wait for more than one transaction at a time, the modified algorithm traverses the wait chain once only starting from the newly blocked transaction. The traversal will terminate either at the last transaction in the wait path or when a deadlock cycle is detected.

MPA suffers in performance because of the high overheads in maintaining the dependency table. To overcome this difficulty, an enhanced version of probe-based system (EPA) has been introduced. The major modifications to MPA are:

(1) Local wait-for graph is used instead of dependency table for deadlock cycle detection. Probes are initiated whenever any of the local site's wait-for graph involves transactions being blocked by transactions from another site.
(2) Probe messages are sent between the source and destination transactions in the wait path rather than through all the transactions. This should significantly reduce the message overheads.

The detail of the enhanced probe-based algorithm is shown below:

```
For initiation of probe computation by a controller
for a constituent idle process $P_i$:
if    $P_i$ is locally waiting on itself
then  declare deadlock
else  for a process $P_b$ such that
        (i) there is a waiting path from $P_i$ to $P_b$ and
        (ii) $P_i$, $P_b$ are on different controllers
      send Probe(i,b).

For a controller on receiving a Probe(i,k):
if    $P_k$ is idle
then  begin
      if k=i
      then declare that $P_k$ is deadlocked
      else for a process $P_b$ such that
        (i) there is a waiting path from $P_k$ to $P_b$ and
        (ii) $P_k$, $P_b$ are on different controllers
        send Probe(i,b).
      end
```

Table 1: Enhanced probe-based algorithm

3 Performance Model

3.1 Distributed Database Model

The distributed database system (DBS) consists of a collection of N sites, S_1, S_2, ... , S_N, connected by a communication network. The network is assumed to be reliable and fully connected. Each site is a centralized database system where portion of the database is stored. There are M transactions, T_1, T_2, ..., T_M running on the distributed database system. A transaction presents its lock requests to its site controller. There is one controller C_i per each site S_i. A transaction is blocked from the time it submits a request until it is notified that its requesting lock is granted. While a transaction is being blocked, it may not send any lock requests. A lock request can be local or refer to a data object located at another site. In such case the transaction is said to be distributed. A distributed transaction T_i is implemented by processes P_{ij}, each of which is the local process for transaction T_i at site S_j. In case a process P_{ij} requests a nonlocal data object that is managed by some controller C_m, the controller C_j transmits the request to process P_{im} via controller C_m. When P_{im} acquires the requested data object from C_m, it sends a message to P_{ij} (via C_m and C_j) stating that the data object has been acquired. Hence, intersite requests are always between two processes of the same transaction.

It is assumed that messages send by any controller C_i to C_j arrive sequentially and in finite time. If a single transaction runs by itself in the distributed database system, it will terminate in finite time and release all its seized data objects.

3.2 Simulation model

Three sites are used in the model. They are fully connected by duplex communication links. The communication links are modeled as service centers and it can service only one message at any time in any one direction. Each site consists of a central subsystem with a number of terminals. The central subsystem has one CPU and one disk unit where the database is stored. The terminals submit transactions interactively after a think time. Two phase commitment [11] is employed in the model to ensure the atomicity of distributed transactions. A modified Two phase locking concurrency control protocol is adopted. Under this locking protocol, the number of messages for locking is minimized by grouping the lock requests for the same site into one message. For example, all the locks for the data objects in site i will be send together in message m_i. The order of sending messages for a transaction is m_1, m_2, ..., m_n.

Local wait-for graph (WFG) are built for each site. The site controllers keep track of the owner of each data object, i.e. the transaction which locks a given data object. By referring to the data object ownership, local dependency can be determined. The dependency of a transaction on another transaction at a different site can be detected by searching the out-going edges of the WFG.

3.3 Workload parameters

The following table is a list of workload parameters used in the set of experiments.

Parameter	Description	Values
N_s	Number of sites	3
DO	Number of data objects in the database of each site	1000
TS	Mean number of data objects to be accessed	5,20
P_l	Probability of a lock request being local	60%
MPL	Multiprogramming level	1-25
T_{cpu}	Mean CPU processing time for a data object	30 ms
T_{io}	Mean I/O time to access a data object	30 ms
T_{ch}	Mean time to check a lock	1 ms
T_{set}	Mean time to set a lock	1 ms
T_{rel}	Mean time to release a lock	2 ms
T_{wfgchk}	Mean time to check local wait-for graph	1 ms
T_{wfgupd}	Mean time to update local wait-for graph	1 ms
Time_out	Timeout period	2.5 sec

Table 2: Model Workload parameters

One of the greatest problems in performance modeling is to choose relevant parameter values which are able to demonstrate the performance characteristics of the protocols under studied. In order to be able to observe interesting performance characteristics without unduly long simulation runs, a heavy workload and high probability of lock conflicts among different transaction is highly desirable. A large number of terminals are used to model a heavy loaded system. To increase the degree of lock conflicts, a relative small database size in comparison with transaction size has been chosen.

The degree of data contention is determined by the probability of conflict among different concurrent executing transactions and is therefore dependent on the transaction sizes as well as the multiprogramming level. Two transaction sizes, 5 and 20, are selected to represent low and high data contention environments. The performance measures are system throughput and resource overheads for deadlock detection. Deadlock detection overhead is defined as the amount of CPU time used on deadlock detection.

Performance studies for system using timeouts are complicated by the need to carefully select the time-out period. If the time-out period is too small, the number of false deadlock will be very large. If the time-out period is too long, deadlock cycles detecting will be delayed and more transactions will be blocked. Figure 1 shows the system throughput as a function of the time-out period. It is observed that the system performance for the present system is best when the time-out period is 2.5 second. Therefore it is adopted as the input parameter for our simulation model.

4 Performance Results and Interpretation

Measurements taken in this paper are the throughput, deadlock detection overhead, deadlocked transaction to committed transaction ratio, number of restart per transaction and blocking time. Throughput is the number of transaction committed per second. Deadlock detection overhead measures the percentage of the CPU time spent in the detection of deadlock cycles. Factors contributing to the deadlock detection overhead will differ for different algorithms. Block time is also used as a performance indicator. It denotes the percentage of CPU time wasted waiting for the required data objects to be released by another transaction. The deadlocked to committed transaction ratio measures the number of transactions deadlocked per committed transaction. The number of restarts per transaction measures the average number of restarts by each transaction before it commits.

In the set of experiments, the performance of the three algorithms i.e. enhanced & modified probe-based algorithms, and timeout under low and high data contention environment are compared. When data contention is negligible, timeout algorithm should outperform all the probe-based approaches [3]. As can be seen in figure 2, with increased data contention, the performance of timeout and the modified probe-based algorithm are comparable. Deadlock occurrences become more frequent, the deadlocked to committed transaction ratio as a function of MPL is shown in figure 3. It is obvious from figure 2, EPA has the worst performance amongst the three algorithms. For EPA, overheads have been incurred in searching the wait-for graph. Figure 5 shows that EPA has the highest deadlock detection overhead, especially for small MPLs. From figure 4, it is apparent that Blocking delay for EPA is only marginally higher than the other two algorithms. As can be seen in figure 6, under high data contention, the enhanced probe-based algorithm is much more efficient than the other two algorithms. Figure 7 gives the throughput for the three algorithms as a function of MPL when data contention is high. The performance of the enhanced probe-based algorithm is much better than the modified probe-based algorithm because of the smaller deadlock detection overhead. Figure 8 shows that the deadlock detection overhead for the enhanced probe-based algorithm is much lower than the other two algorithms. The high deadlock overheads for timeout can be explained in terms of the high cost of aborts for false deadlocks. As can be observed in figure 9, the number of restarts per transaction for the timeout algorithm (many are false deadlocks) is much higher than the other two algorithms.

5 Conclusions

Two new probe-based distributed deadlock detection algorithms have been introduced to rectify some of the shortfalls of the algorithm proposed by Chandy, Misra, and Haas. The new algorithms have been subjected to extensive simulation experiments and have shown to be error free. From the results obtained, the following conclusions can be drawn:

(1) Under low data contention, timeout algorithm out-performs the probe-based approaches. The overheads in deadlock detection for the timeout algorithm is minimal when deadlock is infrequent. The deadlock detection overheads for the enhanced probe-based algorithm is greater than the modified probe-based algorithm because of the need to check the wait-for graph even deadlock is infrequent or even nonexistent.

(2) Under high data contention, enhanced probe-based algorithm outperforms the modified probe-based and timeout algorithms. The number of transaction restarts for the timeout algorithm is much greater than the probe-based approach. This is the result of a large number of false deadlocks.

(3) System suffers very little performance degradation from the additional overhead for the enhanced probe-based algorithm.

Another observation from the experiments is the heavy penalty paid on the arbitrary initiation of the deadlock detection mechanism. Other ways of enhancing deadlock detection efficiency and the effect of communication network bandwidth also be examined. In the probe-based approach, there is no rule to-date to determine the best time for the initiation of probes.

References

[1] A. Bernstein and N. Goodman, "Concurrency Control in Distributed Database Systems," *ACM Comp. Sur.*, Vol. 13, No. 2, pp. 185-221, Jun. 1981.

[2] A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Reading, Massachusetts, Addison-Wesley, 1987.

[3] Chim-fu Yeung, Sheung-lun Hung, Kam-yiu Lam and Chee-keung Law, "A New Distributed Deadlock Detection Algorithm for Distributed Database Systems", *IEEE TENCON '94*, 1994.

[4] S. Ceri and G. Pelagatti, *Distributed Databases Principles and Systems*, New York, McGraw-Hill Book Company, 1984.

[5] M. Chandy and J. Misra, "A distributed algorithm for detecting resource deadlocks in distributed systems", *Proceedings of the ACM Symposium on Principles of Distributed Computing (Ottawa, Canada, Aug.), ACM*, New York, pp. 157-164, 1982.

[6] M. Chandy, J. Misra, and L. M. Haas, "Distributed Deadlock Detection," *ACM Trans. Comput. Syst.*, Vol. 1, pp. 143-156, May 1983.

[7] N. Choudhary, W. H. Kohler, J. A. Stankovic, and D. Towsley, "A Modified Priority Based Probe Algorithm for Distributed Deadlock Detection and Resolution," *IEEE Trans. Software Eng.*, Vol. 15, No. 1, pp. 10-17, Jan. 1989.

[8] A. K. Elmagarmid, "A survey of distributed deadlock detection algorithms," *ACM SIGMOD Rec.* Vol. 15, No. 3, Sep. 1986.

[9] H. Enslow, "What is a "Distributed" Data Processing System?" *Computer*, Vol. 11, pp. 16-23, Jan. 1978.

[10] P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger, "The Notions of Consistency and Predicate Locks in a Database System," *ACM Comm.* Vol. 19, No. 11, pp. 623-633, Nov. 1976.

[11] N. Gray, "Notes on Database Operating System," in Operating Systems: An Advanced Course (Lecture Notes in Computer Science 60), Berlin, Germany, Springer-Verlag, pp. 398-481, 1978.

[12] E. Knapp, "Deadlock Detection in Distributed Database," *ACM, Comp. Sur.*, Vol. 19, No. 4, pp. 302-328, Dec. 1987.

[13] H. T. Kung and J. T. Robinson, "Optimistic Methods for Concurrency Control," *ACM Trans. Database Syst.*, Vol. 6, pp. 212-226, Jun. 1981.

[14] R. Obermarck, "Distributed Deadlock Detection Algorithm," *ACM Trans. Database Syst.*, Vol. 7, No. 2, pp. 202-223, Jun. 1982.

[15] Y. Parker and J.P. Verjus, *Distributed Computing Systems (Synchronization, Control and Communication),* Academic Press, 1983.

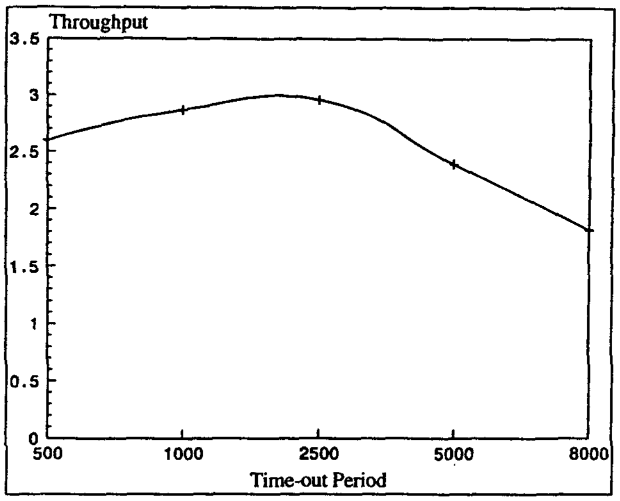

Figure 1: Throughput vs Time-out Period (ms)

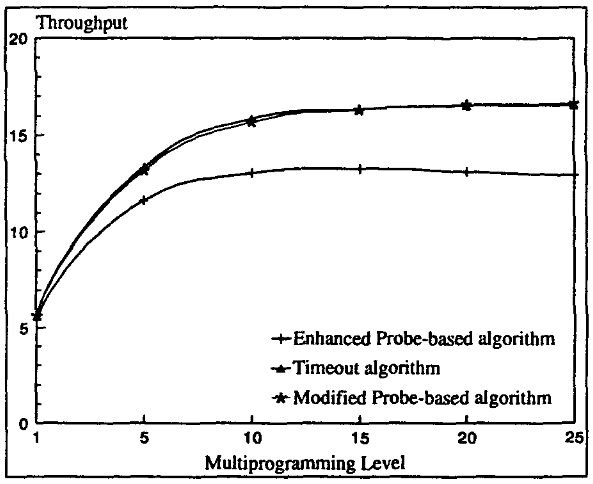

Figure 2: Throughput vs MPL

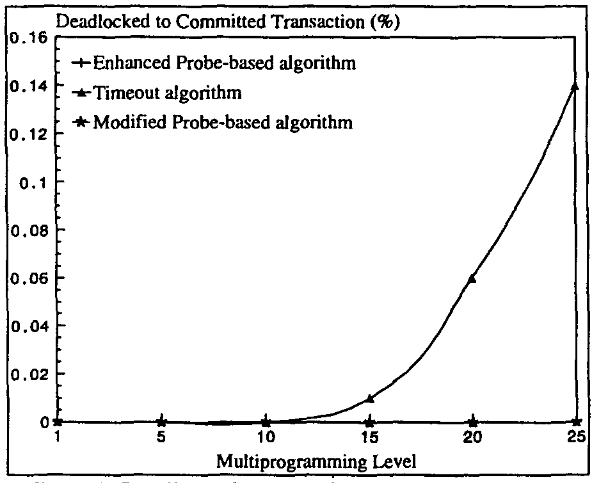

Figure 3: Deadlocked to committed transaction vs MPL

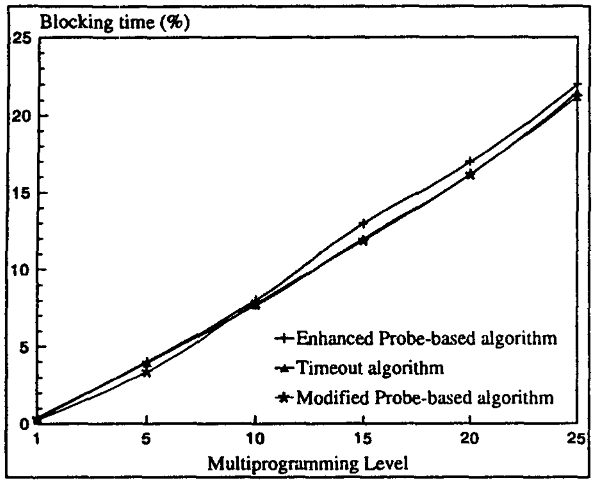

Figure 4: Blocking time vs MPL

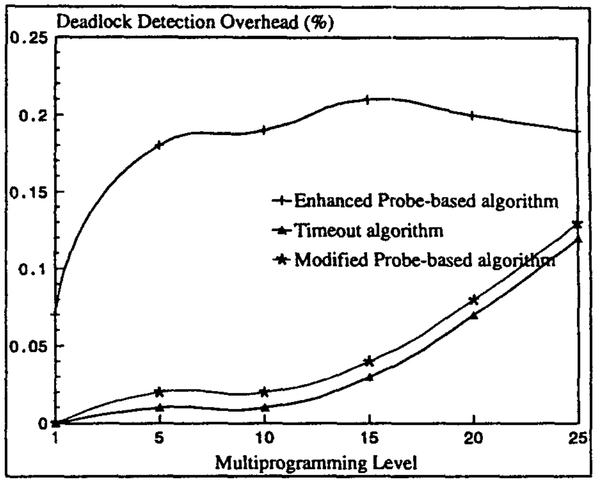

Figure 5: Deadlock detection overhead vs MPL

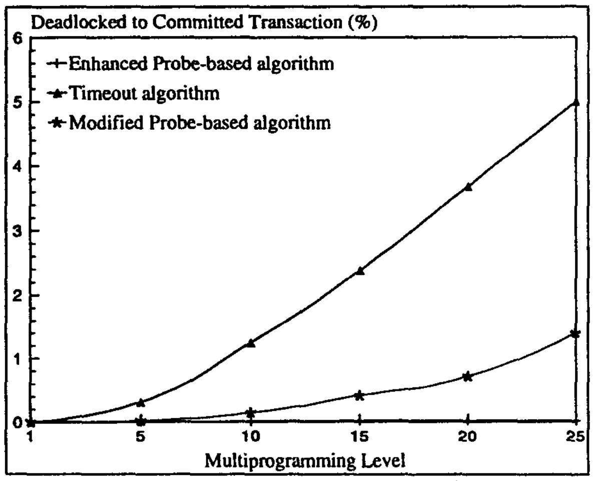

Figure 6: Deadlocked to committed transaction vs MPL

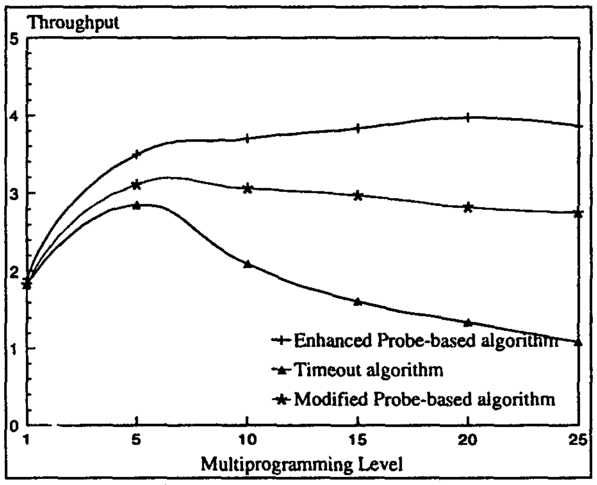

Figure 7: Throughput vs MPL

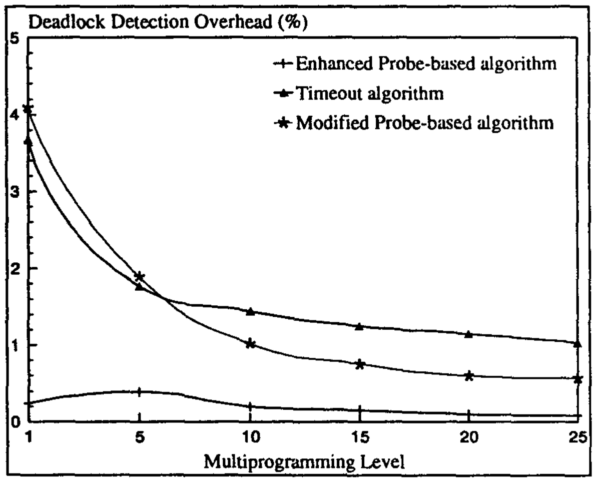

Figure 8: Deadlock detection overhead vs MPL

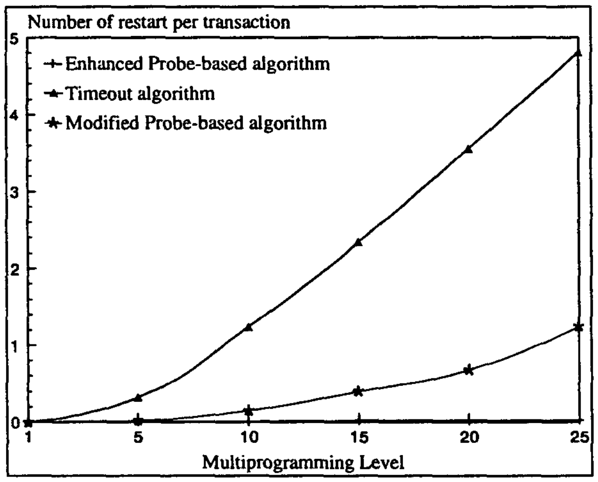

Figure 9: Number of restart per transaction vs MPL