Soft Computing for Database Deadlock Resolution

Saad M. Darwish, Adel A. El-Zoghabi, and Marwan H. Hassan

***Abstract*—Distributed nature of transactions arising at different sites and needing resources from diverse locations pose various operational problems, such as deadlocks, concurrency and data recovery. A deadlock may occur when a transaction enters into wait state that request resource from other blocked transactions. Deadlock detection and resolving is very difficult in a distributed database system because no controller has complete and current information about the system and data dependencies. In this paper, an enhanced technique for deadlock resolution is presented, which minimizes the abortion or waiting of the selected victim transactions. The proposed system includes the use of fuzzy logic by creating a set of fuzzy rules in order to deal with criticalness and similarity attributes of transactions. By using these rules, fuzzy logic will try to provide an easy conflict resolution method between transactions to diminish transactions wasted restart, and guaranteeing temporal consistency of data and transactions. Furthermore, the presented deadlock handling algorithm does not detect any false deadlock or exclude any really existing deadlocks. Experimental results show performance of the recommended system benefits such as increase in commit rate and decrease in re-execution or waiting of the transactions.**

***Index Terms*—Fuzzy logic, deadlock resolving, transactions conflict, concurrency control.**

I. Introduction

Deadlock is one of the most serious difficulties in multitasking concurrent programming systems. The deadlock problem becomes further complicated when the underlying system is distributed and when tasks have timing limitations. Deadlock is a system state in which every process in some group requests resources from other processes in the group, and then waits indefinitely for these requests to be satisfied [1]. Deadlock is an undesirable situation; some of the consequences of deadlock are: throughput of the system is affected; utilization of the involved resources decreases to zero; deadlock increases with deadlock persistence time; and deadlock cycles do not terminate by themselves until properly detected and resolved [2], [3]. The deadlock problem is intrinsic to distributed database system which employs locking as its concurrency control algorithm. Concurrency control and deadlock handling are the most important problems that must have a powerful attention when sharing information in distributed systems [1].

Deadlock resolution requires at least one of the transactions causing the deadlock to release locks. This involves a partial rollback, lock de-escalation, or most commonly a transaction termination [3]. The throughput of the entire database system depends on the efficiency and accuracy of the deadlock detection and resolution algorithms. The correctness of a deadlock algorithm depends on two conditions. First, every deadlock must be detected eventually. This constitutes the basic progress property in which any solution must have. Second, if a deadlock is detected, it must indeed exist (safety property) [4]. Incorrectly detected deadlocks due to message delays and out-of-date wait-for-graphs (WFGs) have been termed phantom deadlocks. The main disadvantage of deadlock detection schemes is the additional overhead incurred due to detection of cycles in the graph and abortion and restart of transactions upon detection of deadlocks. The distributed detection strategies may have additional overhead due to the inter site message transfers. Selection of the transaction to be aborted adds to the complexity of the scheme.

There are four techniques regularly engaged to deal with deadlocks in database systems: ignore the problem, deadlock detection, deadlock prevention and deadlock avoidance. Ignoring deadlocks is the easiest scheme to implement. Deadlock detection attempts to locate and resolve deadlocks. Deadlock avoidance describes techniques that attempt to determine if a deadlock will occur at the time a resource is requested and reacts to the request in a manner that avoids the deadlock. Deadlock prevention is the structuring of a system in such a manner that one of the necessary conditions for deadlock cannot occur. Each solution category is suited to a specific type of environment and has advantages and disadvantages, see [4], [5] for more details.

In general, database deadlock resolution involves the following nontrivial steps [6]-[9]: 1) Select a victim (the transaction to be aborted) for the optimal resolution of a deadlock (this step may be computationally tedious). 2) Abort the victim, release all the resources held by it, restore all the released resources to their previous states, and grant the released resources to deadlocked processes. 3) Delete all the deadlock detection information concerning the victim at all sites. Execution of the second step is complicated in environment where a process can simultaneously wait for multiple resources because the allocation of a released resource to another process can cause a deadlock. The third step is even more critical because if the information about the victim is not deleted quickly and properly, it may be counted in several other (false) cycles, causing detection of false deadlocks. To be safe, during the execution of the second and third steps, the deadlock detection process (at least in potential deadlocks that include the victim) must be halted to avoid detection of false deadlocks.

Efficient resolving of a deadlock requires knowledge of all

Manuscript received July 25, 2014; revised October 10, 2014.

Saad M. Darwish and Adel A. El-Zoghabi are with the Department of Information Technology, Institute of Graduate Studies and Research, Alexandria University, 163 Horreya Avenue, El-Shatby 21526, P.O. Box 832, Alexandria, Egypt (e-mail: saad.saad@alexu.edu.eg, zoghabi@gmail.com).

Marwan H. Hassan is with the College of Imam Aladham, Branch Anbar province, The Sunni Endowment Diwan, Iraq (e-mail: marwanhh80@yahoo.com).

the process involved in the deadlock and all resources held by these processes [3]. When a deadlock is detected, the speed of its resolution depends on how much information about it is available, which in turn depends on how much information is passed around during the deadlock detection phase. In existing distributed database, deadlock resolution is complicated by at least one of the following problems [1]: 1) A process that detects a deadlock does not know all the processes (and resources held by them) involved in the deadlock. 2) Two or more processes may independently detect the same deadlock.

One of the most commonly used technique for deadlock resolution is timestamp based approach for selecting the victim [4]. In this approach, a timestamp is allocated to each process as soon as it enters the system. The timestamp of the younger process is greater than the timestamp of older process. According to this approach, the victim is selected on this timestamps, the process with the higher timestamp is aborted, that is the youngest process is selected as the victim and is aborted in order to break the deadlock cycle. The goal behind choosing the youngest process as victim is that the youngest process would have used less resources and less CPU time as compared to older process. One problem with this technique is that it can cause starvation problem because every time a younger process is aborted which can starve the younger process from completion.

In recent years, many algorithms are proposed that support dynamic adjustment of serialization order found from transaction object attributes to deal with database' transactions conflicting and to resolve deadlocks [9]-[11]. These methods use importance of transaction and operation similarity and can ensure a very well real-time performance by minimizing transactions wasted restart, under circumstances of guaranteeing logical and temporal consistency of data. In general, using importance or criticalness of the transaction in place of the priority in the conflict resolution of existing methods avoids the dilemma of priority based conflict resolution, because transactions with very short deadline (i.e. very high priority) are not necessarily more critical than transactions with high importance [12]. However, these approaches do not take the fuzziness of transaction object attributes into consideration, and meet trouble with false positives and errors in deadlock detection. Another problem is that these approaches do not discuss the priority of all potential deadlocks, and it is hard to determine that deadlock is more baleful and need to be solved first.

Fuzzy logic systems are widely recognized to be successful at modeling uncertainty in a large variety of applications. Basically, it provides an effective means of capturing the approximate, inexact nature of the real world [13]. The use of fuzzy logic is essential at decision making processes where the description by algorithms is very difficult and criteria are multiplied. The fuzzy logic measures the certainty or uncertainty of membership of element of the set. The solution of certain case is found on the principle of rules that were defined by fuzzy logics for similar cases.

The main contribution of this paper is to propose a new priority based scheme for deadlock resolution that depends on fuzzy logic to deal with uncertainty of transaction's attributes. This scheme uses dynamic adjustment of transactions operation similarity and importance for resolving deadlock between transactions whenever conflict arises, thereby decreasing transaction re-executions or waiting and current load of the database server. To the best of our knowledge, this is the first research effort to explore the adaptation of the fuzzy logic for deadlock resolution technique to database systems.

The remainder of this paper is organized as follows: in Section II we review the most important existing deadlock control methods proposed in the literature. Afterward, Section III introduces the proposed method by presenting and explaining in detail the task of each step of this method. In Section IV, we present performance evaluation of proposed method. Finally, Section V includes the conclusion of our works and our future perspectives.

II. Related Works

As mentioned earlier deadlock occurs whenever waiting transactions want access to the resources held by each other and none of them is able to complete their execution [2]. Most of the reviewed algorithms imply rollback/abort as the solution to deadlocks. The only ways in which they differ is how they select the victim. Most of the strategies of victim selection have been reviewed in the literature, the only drawback of such strategies is that it leads to abort of the victim, or they restart the victim which leads to wastage of resources, wastage of the work done by the aborted process, low throughput of system and it makes execution time of processes unpredictable.

Some authors introduced an optimization technique for deadlock detection that minimizes the abortion of the selected victim transactions. Their technique used TQ (Transaction queue) to store the priority for all transactions which are in local or global deadlock cycles. Based on the priority, the youngest transactions are aborted to free the system from deadlock cycles. The presented technique aborts the transaction's requests which are really to blame for the formation of many deadlock cycles. Also the algorithm does not detect any false deadlock or exclude any really existing deadlocks. In this technique global deadlock is not dependent on the local deadlock system [14].

P. Sapra *et al.*, in [4] introduced a deadlock detection and resolution technique using the concept of priorities. This algorithm maintains a list of all the transactions, and whenever a deadlock cycle is detected, the priorities of the transactions constituting the deadlock are checked. The transaction with the least priority is aborted so that the resources held by it can be set free and can be granted over to the waiting transactions. But it has been found that if the order of the priorities is changed this algorithm fails to detect deadlocks. In the same direction, the authors in [15] presented a new deadlock resolution algorithm which doesn't cause any aborts /roll backs. It is based on the mutual cooperation of transactions and a random number representing time duration for which the process holding the resource will be suspended. In this algorithm the distributed system's site coordinator manages its own transactions and resolves any deadlock when detected.

Differing with above researchers, M. Ghodrati *et al.*, [16] used neighbor replication on grid to resolve deadlock. The solution provided for selecting a victim to break the deadlock cycle in addition to the ID of priority importance for transaction systems is also considered. In this model, the new rules for mapping transaction WFG to colored Petri nets modeling for the detection and resolve deadlock are elaborated. In [12] the authors proposed an optimistic concurrency control with the capability of predicting the correctness of the transactions history in case it is rescheduled. Furthermore, the authors used the concept of similarity between conflicting operations to obtain a better real-time performance, and the transaction importance criterion in order to favor transactions with higher importance in data conflict resolution. In addition, they relaxed serializability criterion by introducing data similarity and operation similarity, by allowing two conflicting operation to commit if they meet operation similarity that means when they are slightly different we consider them as acceptable. The method resolved conflicts using time intervals of the transactions. Every transaction must be executed within a specific time slot.

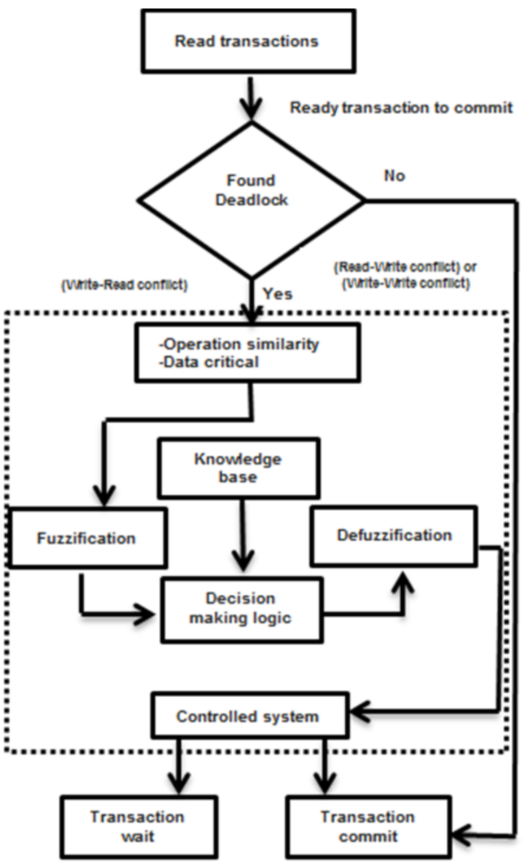


Fig. 1. Proposed fuzzy based deadlock resolution approach.

.

Built on top of the work suggested in [12], a similar type of approach is adopted here to resolve deadlocks based on fuzzification of the transaction's attributes to build a new rules-based priority for conflict resolution between transactions. Design of fuzzy logic or rule based non-linear controller is easier since its control function is described by using fuzzy sets and if-then predefined rules rather than cumbersome mathematical equations or larger look-up tables; it will greatly reduce the development cost and time and needs less data storage in the form of membership functions and rules. The properties of this solution are locality of transactions, and asynchronous operation. We elaborate our simulation results and justify performance gain of the proposed scheme for achieving deadlock management in database environments by eliminating limitations of the existing schemes, increasing commit rate and decrease in re-execution rate of the transactions.

III. Deadlock Resolution with Fuzzy Logic

In order to overcome shortcomings of the deadlock resolution methods discussed above to certain contain, by using transaction's features, a robust resolving scheme using both transaction's features-based and fuzzy logic is proposed as shown in Fig. 1. The suggested system utilizes fuzzy logic technique by creating a set of fuzzy rules that will form the fuzzy logic engine in order to deal with the criticalness and the similarity attributes of transactions. By using these rules, fuzzy logic will try to provide an easy conflict resolution method between transactions. The algorithm attempts to outperform the previous methods by reducing the number of transaction waiting and increasing the concurrency level while maintaining the data valid as much as possible. Table I shows the different terms and parameters applied in the proposed system.

TABLE I: List of Parameters Used in the Proposed Approach

Parameter	Definition
D	Transaction's data object
T	Transaction
t	Current time
T_a	Active conflicting transaction
T_v	Validating conflicting transaction
RS	Data read set (The adjustment of timestamp of an active transaction iterates through the read set)
WS	Data write set (The adjustment of timestamp of an active transaction iterates through the read set)
$S(T)$	Start time of transaction T
$A(T)$	Arrival time of T
$C_t(T)$	The estimated completion time of T at time t
$E_t(T)$	The estimated remaining execution time of T at time t
$D(T)$	Deadline of transaction T
$T(Op_i, D)$	Transaction 's operation sharing with D
$f(x;a,b,c,d)$	Trapezoidal membership function

Step 1. Input transaction (read phase): at the stage of entering the transaction, the initial timestamp will be determined. The timestamp of all other concurrently running and conflicting transactions must be adjusted to reflect the serialization order. At the start of the execution, the timestamp interval of a transaction T is initialized as $[S(T), d(T)]$. whenever the serialization order of the transaction is induced by its data operation or the validation of other transactions, its timestamp interval is adjusted to represent the dependencies [12].

In the suggested system, each transaction has a unique identifier. Many deadlock detection algorithms require a total ordering on transactions for deadlock resolution and assume that the transactions' identifiers can be used for this purpose, e.g., to determine the youngest transaction. However, after a transaction is aborted, it has to be restarted with a new identifier, otherwise information regarding the aborted and the restarted execution of the transaction could not be distinguished, possibly leading to inconsistencies. Still, changing the identifier could alter the ordering of transactions, e.g., an old transaction might become the youngest. To avoid this, in the introduced system, a transaction is associated with a timestamp (additionally to the identifier), indicating the time it has entered the system, which is not modified after an abort, and can therefore be used for transaction ordering. For simplicity reasons, in the rest of the paper, we use identifiers for the ordering of transactions and assume that they contain such a timestamp [17].

In general, there are three phases for database's transaction: (1) Read phase: The transaction reads the values of all data items it needs from the database and stores them in local variables. In some methods updates are applied to a local copy of the data and announced to the database system by an operation named pre-write. (2) Validation phase: The validation phase ensures that all the committed transactions have executed in a serializable fashion. For a read-only transaction, this consists of checking that the data values read are still the current values for the corresponding data items. For a transaction that has updates, the validation consists of determining whether the current transaction leaves the database in a consistent state, with serializability maintained. (3) Write phase: This follows the successful validation phase for update transactions. During the write phase, all changes made by the transaction are permanently stored into the database [14].

Step 2. Deadlock detection: When access has been made the same objects D_i both T_v invalidating and active T_a transactions and at least one of the operations is a write operation, then we have a conflict (deadlock is detected). In practice, deadlock detection often assumes a simplified resource model; the system contains only reusable resources and there is only a single unit of every resource. This model makes deadlock detection simple to implement, but at the cost of detecting fewer types of deadlock.

The proposed system follows Single Request Model for static deadlock detection in which a process can have at most one outstanding request for only one unit of a resource. Since the maximum out-degree of a node in a WFG for the single resource model can be 1, the presence of a cycle in the WFG shall indicate that there is a deadlock. The rationale of choosing this request model is that it simplifies the problem of detecting the deadlock and easy to implement [18]. Formally, conflict can occur when [12]:

$$D_i \in (RS(T_a) \cap WS(T_v)) \neq \phi \quad \text{(read-write conflict)} \quad (1)$$

$$D_i \in (WS(T_a) \cap RS(T_v)) \neq \phi \quad \text{(write-read conflict)} \quad (2)$$

$$D_i \in (WS(T_a) \cap WS(T_v)) \neq \phi \quad \text{(write-write conflict)} \quad (3)$$

Here, to reflect the new developments, the attempt is to use transactions' features to solve the conflict between them through employing fuzzy controller to handle uncertainty associated with these features that affecting to the transactions' priority .

Step 3. Transaction' attributes extraction: The proposed method employs the concept of similarity for non-critical temporal data items that takes into account transaction's operations such as read, write, and shared resources and criticalness that takes into account the estimated completion time of T as transaction's attribute which uses information about the importance of the transactions that will be fed into fuzzy logic engine for conflict handing. These two features were selected for ease of application and ease of calculations inside fuzzy logic engine. Suppose t_m and t_n are a pair of concurrent transactions, $Op_i \in t_m$, $Op_i \in t_n$, Op_i and Op_j operate on the same non-critical data object D (conflicting operations). If the following condition is satisfied [12]:

$$\left|T(Op_i, D) - T(Op_j, D)\right| \leq \propto \quad (4)$$

$\propto$ is the threshold value whose value depends on the application semantics, then Op_i and Op_j are said to be operation similarity, notated by $Op_i \approx Op_j$. Furthermore D is critical $C_t(T,D) = true$ if:

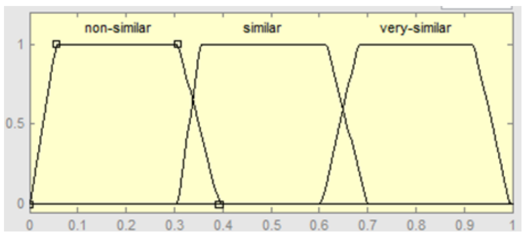
$$C_t(T,D) \leq d(T),$$
$$C_t(T,D) = t + E_t(T) \quad (5)$$

Criticalness measures how critical it is that a transaction meets its timing constraints. Different transactions have different criticalness. Furthermore, criticalness is a different concept from deadline because a transaction may have a very tight deadline but missing it may not cause great harm to the system. Here, expected execution time is very hard to predict but can be based on estimate or experimentally measured value of worst case execution time.

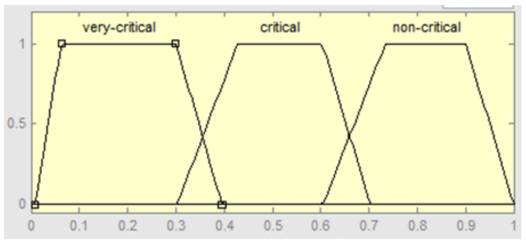
Step 4. Fuzzy logic controller: Fuzzy logic controller is adaptive in nature and can also exhibit increased reliability, robustness in the face of changing transaction's features. The first step in the design of a fuzzy logic controller is to define membership functions for the inputs; three fuzzy levels or sets are chosen and defined by the following library of fuzzy-set values for the similarity (non-similar, similar, very similar) and critical attributes (very critical, critical, non-critical) of transaction as shown in Fig. 2a,2b. For a given crisp input, fuzzifier finds the degree of membership in every linguistic variable. The number of fuzzy levels is not fixed and depends on the input resolution needed in an application. The larger the number of fuzzy levels, the higher is the input resolution. The fuzzy controller utilizes trapezoidal membership functions on the controller input [13].

Membership functions allow us to graphically represent a fuzzy set. The x axis represents the universe of discourse, whereas the y axis represents the degrees of membership in the [0,1] interval [19]. Simple functions are used to build membership functions. Because we are defining fuzzy concepts, using more complex functions does not add more precision. The trapezoidal membership function is chosen due to its simplicity. All of membership' parameters are numerically specified based on the experiences to handle transactions. In our case, all fuzzy levels have the same space on the number line. The trapezoidal curve is a function of a vector, x, and depends on four scalar parameters a, b, c, and d, as given by [13]:

$$f(x;a,b,c,d)=\begin{cases}0 & x<a\\ \dfrac{x-a}{b-a} & a\le x\le b\\ 1 & b\le x\le c\\ \dfrac{d-x}{d-c} & c\le x\le d\\ 0 & d<x\end{cases} \tag{6}$$



(a)



(b)

Fig. 2. Membership function for (a) Similarity (b) criticalness.

Fuzzy control rules are obtained from the analysis of the system behavior. The control rules that associate the fuzzy output to the fuzzy inputs are derived from general knowledge of the system behavior. However, some of the control actions in the rule table are also developed using "trial and error" and from an "intuitive" feel of the process being controlled. The derivation of the fuzzy control rules is heuristic in nature and consists of the following rules:

- **If** (similarity is non-similar) and (criticalness is non-critical)
- **Then** (wait).
- **If** (similarity is non-similar) and (criticalness is very critical)
- **Then** (commit).
- **If** (similarity is similar) and (criticalness is non-critical)
- **Then** (wait).
- **If** (similarity is very similar) and (criticalness is very critical)
- **Then** (commit).
- **If** (similarity is very similar) **Then** (commit).
- **If** (criticalness is very critical) **Then** (commit).

The results of the inference mechanism that include both of similarity and criticalness factors of the individual rule is obtained by Mamdani's min or max fuzzy implications of $Min\{\mu_A(x),\mu_B(x)\}$ or $Max\{\mu_A(x),\mu_B(x)\}$, where $\mu_A(x),\mu_B(x)$ are membership degrees for similarity and criticalness respectively. Conservation of the fuzzy to crisp or non-fuzzy output is defined as defuzzification. In the defuzzification operation a logical sum of the inference result from each of the six rules is performed [19].

$$y' = \frac{\sum_{i=1}^{6} y_i \mu_B(y)}{\sum_{i=1}^{6} \mu_B(y)} \tag{7}$$

The system decides which transaction is waiting or committing based on the output. If $y' \geq \propto$ then transaction is committing else transaction is waiting. In our case $\propto = 0.5$ to achieve systems robustness in terms of increase in commit rate and decrease in re-execution or wait of the transactions. In a DBMS, deadlock resolution means that one or more of the participating transactions, the victim, is chosen to be aborted, thereby resolving the deadlock. But when more than one deadlock cycle is involved in any distributed site or among the sites, it is required to optimize the request of the transactions which are involved for the cause of the major deadlock cycles.

IV. Simulation Results

In this section, we conducted an extensive set of simulation experiments using the above mentioned parameters in Table I through MATLAB and PHP languages. Wait percentage (Wait %) and Commit percentage (Commit %) were used as measures for the performance metrics in our simulation results. Wait % (how many transactions wait due to violation of serializability before final commit from the total number of transactions taken for concurrent execution) is the percentage of input transactions that have non critical attribute and have less than 0.6 in the similarity scale and Commit % (how many transactions successfully committed execution from the total number of transactions taken for concurrent execution) is the percentage of input transactions that have very critical attribute and have greater than 0.6 in the similarity scale (according to fuzzy system rules). We conducted simulation under normal and heavy loads with various settings of workload parameters such as number of transactions, transaction workload (simple or complex transaction) and with other corresponding parameter values.

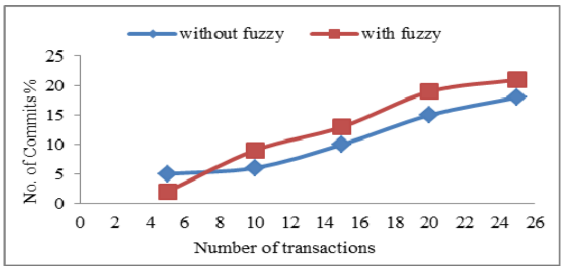


Fig. 3. Performance graph of commit rate for simple transaction.

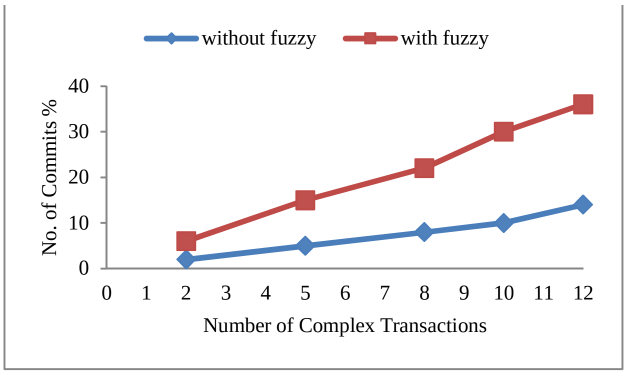
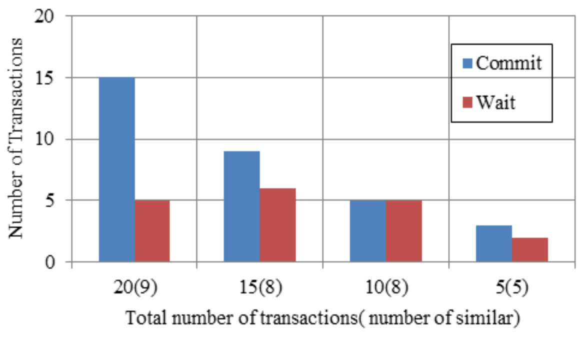


Fig. 4. Performance graph of commit rate for complex transaction.

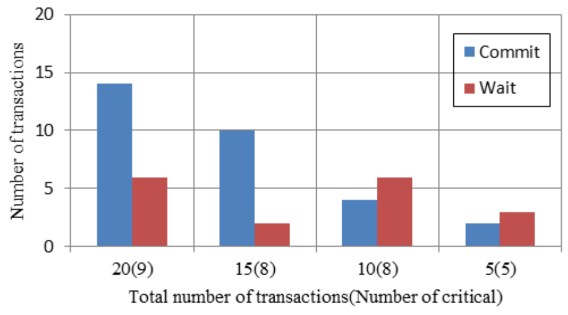
Experiment 1. Comparison of waits % and commit % for simple transactions under fuzzy and non-fuzzy deadlock resolution. In this experiment, we considered 25 transactions with one SQL statement and other parameter values were kept constant. The experimental results are shown in Fig. 3 for commit%. These results show that wait %s were low and Commit %s high for fuzzy approach compared with non-fuzzy approach. Commit %s for 25 transactions under fuzzy and non-fuzzy approaches was 80% and 72% respectively.

Experiment 2. Comparison of waits% and commit% for complex transactions under fuzzy and non-fuzzy deadlock resolution. In this experiment, we considered transactions with average 4 SQL statements and other parameter values were kept constant. The experimental results are shown in Fig. 4 for commit%. These results show that wait %s were low and Commit %s high for fuzzy approach compared with non-fuzzy approach. Commit %s for 10 transactions under fuzzy and non-fuzzy approaches was 75% and 25% respectively.

It is evident from the above graphs that the proposed scheme effectively reduces the wait rate of the transactions. This is because the proposed scheme gives more priority to the transaction's features, which requested the data item first in the execution while resolving conflict of data item access. This ensures implicit serialization order of the concurrently executing transactions, which is essential to maintain consistency of the database. Both figures show that wait %s were low and Commit %s, high.



(a)



(b)

Fig. 5. Performance graph of commit and wait rates for simple transaction attributes (a) similarity, (b) criticalness.

Experiment 3. Comparison of wait% and Commit% under fuzzy deadlock resolution for each transaction's attributes. The goal of this experiment is to compare wait % and Commit % of each simple transaction attributes. Results are shown in Fig. 5a, 5b. In Fig. 5(a) we measure the number of transactions commits and waits with related to transactions' similarity. As illustrated from the first and second column, the number of transactions is 20 in all and the number of similar transactions is 9 that actually committed and the number of non-similar transactions is 11; the proposed algorithm adds 6 transactions to be committed from the non-similar transactions, which means that that total number of committed transactions is 15 and the number of waited transactions is 5. It has been observed form the figure that when the similarity of transactions is high the number of transactions commits increases and the number of transactions waits decrease.

In the same scenario, as shown in Fig. 5(b), we measure the number of transactions commits and waits that are related to critical transactions. As depicted from the first and second column the number of critical transactions is 9 that actually committed and the total number of non-critical transactions is 11, so the proposed algorithm appends 5 transactions to be committed from the non- critical transactions thus the total number of committed transactions is 14 and the total number of waited transactions is 6. It has been observed form the figure that when the number of critical transactions increases the number of transactions commits increases and the number of transactions waits ranges from 3 to 6.

In summary, our results show that the proposed system can resolve deadlock quickly and that it introduces little performance overhead to normal applications that do not deadlock. Furthermore, our extensive experiments demonstrate that fuzzy technique substantially improves transaction throughput. Together, our resolution manager achieves up to 30x better throughput than non-fuzzy resolution algorithm for both sample and complex transaction. The complexity of the system depends on number of transactions and average query statements in each transaction besides computations of fuzzy inference engine. We measure the time that takes about to detect a deadlock, which is the duration from the time that the system enters the detection mode to the time it finishes the resolution and prints out the results. It takes 1.5 seconds in the Apache web server. For application developers, this system can be viewed as another tool to add to the deadlock resolution toolbox. When the suggested and the existing tools are used together, they can provide the best coverage of deadlocks.

V. Conclusion

Deadlock can occur in any concurrent system and is often difficult to debug. Existing deadlock resolution techniques are either impractical for large software database systems or over-simplified in their assumptions about deadlock-sensitive resources. In this paper, we propose fuzzy-based deadlock resolution, a novel database system mechanism that dynamically handles deadlock in database applications with the capability of predicting the correctness of the transactions history in case it is rescheduled. The proposed system improves the drawbacks of the existing schemes by prioritizing the transactions based on their features. The suggested system increases the overall commit rate of the system and decreases the rate of waits.

The system employs the concept of similarity between conflicting operations to obtain a better real-time performance, and the transaction criticalness criterion in order to favor transactions with higher importance in data conflict resolution. Furthermore, the system exploits fuzzy logic approach as the famous artificial intelligence technique to merge transaction's features to provide an easy conflict resolution method between transactions. The advantages of proposed scheme are 1) transactions data item access priority is maintained to ensure serializability without aborting the transactions. 2) the cost of waiting time of the transaction to execute is less than the cost of re-execution of the transaction. Hence, transaction can wait little more to acquire a data item than to access and get aborted 3) the transaction, which has done more work, is given higher priority, as it will finish early if given more privilege. Finally 4) the overall through put of the system increases by sacrificing a small amount of waiting time and overhead is conserved.

Also, a simulation implementation and a performance comparison between fuzzy and non-fuzzy real-time deadlock control methods show that our method can ensure a very well real-time performance while guaranteeing temporal consistency and can even outperform non-fuzzy method in many cases. Moreover, we can try to implement our proposed method on a real-time database test platform and on a real database management system to obtain more accurate results.

References

[1] P. Sapra, S. Kumar, and R. Rathy, "Deadlock detection and recovery in distributed databases," *International Journal of Computer Applications*, vol. 73, no. 1, pp. 32-36, July 2013.

[2] S. Singh and S. Tyagi, "A review of distributed deadlock detection techniques based on diffusion computation approach," *International Journal of Computer Applications*, vol. 48, no. 9, pp. 28-32, June 2012.

[3] S. Selvarai and R. Ramasamy, "An efficient detection and resolution of generalized deadlocks in distributed systems," *International Journal of Computer Applications*, vol. 1, no. 19, pp. 1-7, 2010.

[4] P. Sapra, S. Kumar, and R. Rathy, "Detection and Resolution of Deadlocks in Multi-Level Secure Databases," *International Journal of Engineering and Technology (IJET)*, vol. 5, no. 3, pp. 3001-3006, June-July 2013.

[5] B. Hossain and M. Grechanik, "REDACT: preventing database deadlocks from application-based transactions," in *Proc. ACM Symposium on the Foundations of Software Engineering*, 2013, Russia, pp. 591-594, 2013.

[6] A. Soleimany and Z. Giahi, "An efficient distributed deadlock detection and prevention algorithm by daemons," *International Journal of Computer Science and Network Security*, vol. 12, no. 4, pp.150-155, April 2012.

[7] T. Li, C. Ellis, A. Lebeck, and D. Sorin, " Pulse: a dynamic deadlock detection mechanism using speculative execution," in *Proc. the Annual Technical Conference*, pp. 31-44, USA, 2005.

[8] S. Srinivasan and R. Rajaram, "Message-optimal algorithm for detection and resolution of generalized deadlocks in distributed systems," *Academic Journal of Informatica*, vol. 35, no. 4, pp. 498-489, 2011.

[9] M. Goswami, K. Vaisla, and A. Singh, " VGS algorithm: an efficient deadlock prevention mechanism for distributed transactions using pipeline method," *International Journal of Computer Applications*, vol. 46, no. 22, pp. 1-9, May 2012.

[10] S. Gupta, "Deadlock detection techniques in distributed database system," *International Journal of Computer Applications*, vol. 74, no. 21, pp. 41-45, July 2013.

[11] F. Tanga, I. Youb, S. Yuc, C.-L. Wangd, M. Guoa, and W. Liue, " An efficient deadlock prevention approach for service oriented transaction processing," *International Journal of Computers and Mathematics With Applications*, vol. 63, no. 2, pp. 458-468, 2012.

[12] M. Walid, K. Nicolas, and H. Mohamad, "An optimistic concurrency control approach applied to temporal data in real-time database systems," *Journal of WSEAS Transactions on Computers*, vol. 11, no. 12, pp. 419-434, 2012.

[13] T. Govindaraj and R. Rasila, "Development of fuzzy logic controller for dc–dc buck converters," *International Journal of Engineering Technoscience*, vol. 2, no. 2, pp.192-198, 2011.

[14] Chahar and S. Dalal, "Deadlock resolution techniques: an overview," *International Journal of Scientific and Research Publications*, vol. 3, no. 7, pp. 1-5, 2013.

[15] B. Alom, F. Henskens, and M. Hannaford, "Optimization of detected deadlock views of distributed database," in *Proc. International Conference on Data Storage and Data Engineering*, pp. 44-48, Bangalore, 2010.

[16] M. Ghodrati and A. Harounabadi, "A new method for optimization of deadlock resolution of distributed database with formal model," *International Journal of Electronics Communication and Computer Engineering*, vol. 5, no. 1, pp. 220-228, 2014.

[17] H. Grover, "A distributed algorithm for resource deadlock detection using time stamping," *International Journal of Engineering Research and Technology*, vol. 2, no. 11, pp. 4124-4132, Nov. 2013.

[18] V. Geetha, N. Sreenath, and A. Kumar, "Deadlock elimination of AND model requests in distributed systems," in *Proc. IEEE International Conference on Recent Trends in Information Technology*, pp. 1154-1157, China, 2011.

[19] C. Wagner and H. Hagras, "Novel methods for the design of general type-2 fuzzy sets based on device characteristics and linguistic labels: surveys," in *Proc. the* 9th *Conference of the European Society for Fuzzy Logic and Technology*, pp. 537-543, Portugal, 2009.

Adel A. El-Zoghabi received his B.Sc. in computer engineering from Alexandria University in 1987, he received his M.Sc. and Ph.D. in information technology from Alexandria University and Old Dominion University in 1991 & 1994 respectively. His research and professional interests include intelligent systems and machine learning, inter networking and routing protocols, and distributed systems. He has published many papers in international journals and conferences worldwide during the past three decades. Currently he is a professor of Computer Science and IT and the head of Dept. of Information Technology since August 2012.

Saad M. Darwish received his Ph.D. degree from the Alexandria University, Egypt. His research and professional interests include image processing, optimization techniques, security technologies, and machine learning. He has published in journals and conferences and severed as TPC of many international conferences. Since Feb. 2012, he has been an associate professor in the Department of Information Technology, Institute of Graduate Studies and Research, Egypt.

Marwan H. Hassan received the B.Sc. degree in computer science from Department of Computer Science, College of computer, University of Alanbar, Iraq in 2004. Currently he is a M.Sc. student in the Department of Information Technology, Institute of Graduate Studies and Research, Alexandria University, Egypt. His research and professional interests include databases processing, concurrency control, and distributed systems .