An improved algorithm for database concurrency control

Article *in* International Journal of Information Technology · March 2019

DOI: 10.1007/s41870-018-0240-y

3 authors:

Marwa mohamed Sharafeldin
faculty of electronic engineering ,egypt

Mohammed Badawy
Menoufia University

Ayman El-Sayed
Menoufia University

Int. j. inf. tecnol.
https://doi.org/10.1007/s41870-018-0240-y

ORIGINAL RESEARCH

An improved algorithm for database concurrency control

Marwa Mohamed[1] · **Mohammed Badawy**[1] · **Ayman EL-Sayed**[1]

Received: 11 February 2018 / Accepted: 18 August 2018

Abstract Concurrency is an effective solution for some of the database problems. As, in database systems, transactions' conflict is a negative factor that effects on the system performance. Concurrency can be considered a positive solution for this problem, if it is applied under some constraints. This paper proposes an enhancement algorithm of two-phase locking to reduce the transactions' conflict and achieve deadlock free locking namely deadlock-free cell lock algorithm. Our proposal is based on the reduction of locking level of the data to the smallest restricted point. Also, it is proposed to eliminate deadlock problem of the locking algorithms via forcing the waiting transaction to pass into the rollback or the commit phase. Our proposed algorithm improves the performance of the database and the transactions.

Keywords Concurrency · Locking · Cell lock · Deadlock · Database

✉ Marwa Mohamed
eng.marwa.sharaf@el-eng.menofia.edu.eg

Mohammed Badawy
mohamed.badawi@el-eng.menofia.edu.eg

Ayman EL-Sayed
ayman.elsayed@el-eng.menofia.edu.eg

[1] Computer Science and Engineering Department, Faculty of Electronic Engineering, Menoufia University, Menouf 32952, Egypt

1 Introduction

Concurrency is defined as concurrent execution of multiple concurrent transactions [1, 2]. For this property of concurrency, it is considered a good way to improve the performance of the database. But, there are some problems produced by applying concurrency [3]. Some of the concurrency problems are transactions conflict and deadlock. Transactions conflict produced due to conflicting multiple transactions on the same data. This leads to rollbacking some of the conflicting transactions to enable the others from executing their operations. Also, deadlock problem produced from an infinite wait of transactions for data lock. To apply concurrency under control, there are some of traditional concurrency control algorithms designed such as *two-phase locking (2PL), Timestamp based Concurrency Control (TCC), Optimistic Concurrency Control (OCC), Multi-Version Concurrency (MVCC), and Partition Locking (PARL).*

The two-phase locking (2PL) is based on locking data requested from concurrent transactions [4, 5]. The complexity of lock depends on the level of locking, as complexity increases with increasing blocked area of the database. As lock level has a strong effect on concurrency and complexity, enhancement on this level can produce an improvement of concurrency and complexity [6]. Deadlock is one of the problems of 2PL. Timestamp based concurrency control (TCC) is based on providing execution priority to older concurrent transaction than the others [6]. In this algorithm, the transaction may enter into a loop of rollbacking and restarting, which has a negative effect on database performance [7, 8]. Optimistic concurrency control (OCC) is based on checking the timestamp of the transaction at the validation phase [9, 10]. OCC is

Published online: 01 September 2018

considered a good solution of concurrency problems, but in rare situations [11–13].

In multi-version concurrency control (MVCC), there are multiple versions of data acquired by concurrent transactions for write request [14, 15]. MVCC improves read operations but has some other problems produced from write operations such as lost updates and huge memory requirements [16]. Partition locking (PARL) algorithm is an enhancement of 2PL in order to reduce locking overhead [17]. This algorithm improves concurrency degree and decreases locking overhead, but cannot overcome deadlock problem.

In this paper, our proposal provides an enhancement of 2PL in order to improve the database performance. As the degree of concurrency in 2PL can be improved via reduction of locking level. Locking level expands from table locking to the proposed level. The degree of concurrency and system complexity varies from one level to the another. Firstly, in table locking, the transaction acquires lock overall the table, this may increase restrictions on the table and reduce concurrency degree. Secondly, in page locking forces transactions to lock the only page not the table. In concurrency algorithm which based on the record, locking tries to decrease a degree of locking. It maintains a level of concurrency between transactions, as it holds locking only on the requested record. Finally, in attribute level, all data stored in this attribute locked by one transaction, while other transactions blocked from accessing any data of the same attribute.

Our proposed level reduces locking level to a restricted area “cell level”. This reduction of locking level tries to reduce blocking area and blocking time. So, our proposed algorithm aims to increase concurrency degree of concurrent transactions.

The rest of this paper organized as follows: Sect. 2 describes the previous related work. Section 3 introduces the proposed algorithm and its analysis against different concurrency control algorithms. Whereas, Sect. 4 describes the performance evaluation and gives the results and discussion. Finally, Sect. 5 concludes the paper.

2 Related work

There are some recent enhancements to these concurrency control algorithms in order to solve some problems of concurrency and reach an improved performance of database system. This present study can be briefed in some researches such as:

In 1999, the researchers proposed an algorithm to improve the response time of 2PL [5]. Also, a useful discussion for some of concurrency control algorithms widely presented as shown in [18]. In 2002, the researchers proposed an algorithm as depicted in [19] in order to minimize the number of interactions in lock manager and number of page fixes. In 2007, the researchers proposed a new protocol [9] to reduce space for coarse detection of conflicts of optimistic concurrency. In 2009, they proposed an algorithm [17], that aims to reduce locking overhead and increase concurrency, based on the partitioning of resources.

In 2010, they proposed an algorithm [20] to improve throughput relative to the other multi partitioning schemes, based on partitioning the database into many partitions. In 2012, the proposed algorithm in [21] has two models, The first model is to avoid the overhead associated with the traditional lock. The second model is to increment throughput of transactions. In 2013, they proposed an algorithm [10] to improve transactions time by increasing number of committed transactions within the deadline. Also, the proposed algorithm [13] is to increase response time and reduce waiting time for concurrent transactions.

In 2014, they proposed a novel KV-Indirection algorithm [15] to achieve a high degree of concurrency. In 2015, they proposed BOHM algorithm [22] to ensure serializable execution of transactions while ensuring read transactions never block write ones. In 2016, to solve the problem of both the scalability and the concurrency, an algorithm [12] is proposed. Also, another proposed algorithm [23] is to improve the performance of a database. In 2017, the improvement of transaction processing throughput is done as shown in [24]. It is based on quick detection of transaction conflict and re-execution of these conflicting transactions. In 2017, they proposed SSN [25] protocol to ensure serializability in the execution of concurrent transactions.

3 Our proposed algorithm: deadlock-free cell lock

3.1 Objectives of our proposed algorithm

As deadlock occurrence increases in situations related to compound transactions, whereas, each transaction executes more than one statement on different data items. This leads to executing some statements and waiting to execute the others. Because of waiting, it can be successful execution or abortion of the transaction. The deadlock has then a negative effect on the database system and transaction performance. The elimination of this problem provides more chances for the waited transaction to complete its operations. So deadlock Free Cell lock (DFCL) is proposed to eliminate the deadlock of the locking algorithms. Also, DFCL **improves** database performance via increasing the number of commits.

3.2 Basic of DFCL

Deadlock-Free Cell lock (DFCL) algorithm is based on locking system with applying a low degree of locking. This algorithm ensures a high degree of concurrency, as it enables many transactions to access the same data concurrently with a low percentage of conflict. DFCL enables each transaction to proceed its operations on its required cells and leaving the remainder data free for the others.

This algorithm follows partitioning of the database into small parts that leads to partitioning lock of them to different locks with different transactions instead of one. Partitioning database record into many cells leads to partitioning record lock to many locks that can be distributed to many transactions. With DFCL trigger, execution priority is to the waiting transaction with many executed statements.

This allows many concurrent transactions to access the same record without conflict. DFCL algorithm reduces the number of transactions that rollback their operations via conflicting. This algorithm improves the performance of concurrent transactions and database performance. Increasing the number of transactions that can be executed per unit of time, leading to an increase in the efficiency of databases and system throughput.

3.3 Execution of reading and write operations in DFCL

In DFCL, read transactions never reject each other's, but write transactions always reject each other. Write transaction blocks any else such either read or write transaction from accessing the same cell. As, in DFCL cell lock should be acquired by only one transaction, and maintained until this transaction completes its operations on this cell. Read transaction locks its requested cell in share lock mode (*read_lock*), where the other transactions can access this cell for either reading or write operation simultaneously. But, write transaction locks cell in an exclusive mode (*write_lock*), where no one can access this cell for any read or write operation until the current write transaction commits. Operation of DFCL for execution of insert operations is shown in the DFCL Pseudo Code in Table 1, Operation of DFCL for execution of update operations is shown in the DFCL Pseudo Code in Table 2, Operation of DFCL for execution of delete operations is shown in the DFCL Pseudo

Table 1 Insert operation under DFCL control

Insert operation in DFCL

```
Call Begin _Transaction ();

// this code is repeated for the same transaction equal to number of requested cell for it
 If Operation=INSERT, then
     Set transaction _statement= "insert";

     L1: Boolean x=record _Lock (record#);
     If (x==0) then
            Call Write_lock(record#);
             Execute Insert statement;
             Print <<" Record is successfully inserted;
             Call Release _Write_lock(record#);
              Set transaction _status= "committed";
           Set no _of _executable _statements=old (Set no _of _executable _statements) +1;
            Set end _time=Current _timestamp ();

       Else
              Sleep (0.1);
             Set no _of _trans _waits=old (no _of _trans _waits) +1;
              Set transaction _status= "waiting for" ||cell#;
                     If (no _of _trans _waits>selected _period) then
                        Call Abort_transaction (transaction_id);
                       Print <<" error, Record is currently inserted by another one";
                         Set transaction _status= "rolledback";
                         Set end _time=Current _timestamp ();
                     Else
                                  Goto L1;
                     End If;
        End if;
```

Table 2 Update operation under DFCL control

```
Update operation in DFCL

Call Begin _Transaction ();

// this code is repeated for the same transaction equal to number of requested cell for it
 If Operation=UPDATE, then
        Set transaction _statement= "update";
         L2: Boolean x=Cell_Lock (cell#);
        If (x==0) then
             Call update _lock(cell#);
             Execute update statement;
             Print <<" Cell is successfully updated";
             Call Release _update _lock(cell#);
              Set transaction _status= "committed";
               Set end _time=Current _timestamp ();
            Else
               Sleep (0.1);
                Set no _of _trans _waits=old (no _of _trans _waits) +1
               Set transaction _status= "waiting for" ||cell#;
               If (no _of _trans _waits>selected _period) then
                       Call Abort_transaction (transaction_id);
                       Print <<" error, cell is currently updated by another one";
                       Set transaction _status= "rolledback";
                   Set end _time=Current _timestamp ();
                 Else
                    Goto L2;

            End If;
        End if;
```

Code in Table 3, Operation of DFCL for execution of read operations is shown in the DFCL Pseudo Code in Table 4.

3.4 DFCL trigger to eliminate deadlock problem

The presence of conflict between the number of transactions on the same cell produced in a long or an infinite wait of transactions for the requested cell. An infinite wait for data is known as deadlock, that spends processing time of transactions in doing nothing instead of the real execution of their operations. DFCL solves this problem by firing trigger to check the number of waiting transactions with a long wait and for the same data. Then it compares between these infinite waiting transactions to determine which transactions should be forced to abort to allow the others to complete. In DFCL trigger, execution priority is assigned to the waiting transaction with the largest number of executable statements. So DFCL trigger forces transaction with the smallest number of executable statements to be aborted. DFCL Trigger is shown in Fig. 1.

DFCL increases the number of successful transactions and decreases the number of aborted ones. Also, it eliminates the deadlock. Then, the read operations never block the write ones, but the write operations will block the read ones.

4 Performance evaluation

4.1 Simulation setup

The first step of an experiment is creating a dataset for Aqua market system with many tables such as *Udata*, *Users*, Offices table, *City* etc., using APACHE MySQL program and create some tables related to DFCL control such process table. Then, experiment inserts some data into many records using *SQL* statements (i.e. insert). Figure 2 shows the structure of used database with inserted data.

The second step of an experiment is sending SQL statements (select, update, insert and delete) to APACHE JMeter program with determining the setting of concurrent execution and number of parallel transactions.

APACHE JMeter is an independent-platform program, supported by APACHE. It has the ability to create a high number of transactions that work concurrently with each other. It receives transactions query from the user and the number of concurrent execution for this transaction, then it

Table 3 Delete operation under DFCL control

```
Delete operation in DFCL

Call Begin _Transaction ();

// this code is repeated for the same transaction equal to number of requested cell for it
If Operation=DELETE, then
        Set transaction _statement= "delete";
        L3: Boolean x=Cell_Lock (cell#);
        If (x==0) then
                Call update _lock(cell#);
                Execute Delete statement;
                Print <<" Cell is successfully deleted";
                Call Release _update _lock(cell#);
                Set transaction _status= "committed";
                Set end _time=Current _timestamp ();
        Else
                Sleep (0.1);
                Set no _of _trans _waits=old (no _of _trans _waits) +1;
                Set transaction _status= "waiting for" ||cell#;
                 If (no _of _trans _waits>selected _period) then
                        Call Abort_transaction (transaction_id);
                        Print <<" error, cell is currently accessed by another one";
                        Set transaction _status= "rolledback";
                        Set no _of _trans _waits=old (no _of _trans _waits) +1;
                Else
                        Goto L3;

                End If;
          End if;
```

creates the number of transactions with supported statements and runs them concurrently on the selected database.

In our system, there is a middle layer between the application layer (created by APACHE JMeter) and source layer.

In this test, 1500 transactions are generated in each run, 20% of them for read operations (i.e. 300) and the other 80% for write operations (i.e. 1200). The evaluation metrics values of the average of ten runs. Simulated the concurrency control algorithms and our proposal uses the same environment. This environment contains a machine with processor 2.3 GHz Intel Core i5, Ram 8 *GByte*, MacBook pro OSX10.11 operating system, and using APACHE JMeter and MYSQL. The various number of transactions are used such as 1000, 1300, 1500 to ensure the performance of DFCL.

4.2 Evaluation metrics

There are some metrics that can be measured to evaluate the performance of concurrency control algorithms such as:

(1) A number of committed transactions.
(2) A number of the rolled-back transactions.
(3) A number of the waiting transactions.
(4) Throughput = no_of_executed_transactions/ execution_time.
(5) Concurrency = throughput/latency.
(6) System performance is based on concurrency and throughput. As, increased throughput and concurrency improves the response time of the transaction and consequently, improves the performance of database system.
(7) Time complexity is based on the whole execution time of the transaction, if the algorithm reduces the waiting time of the transaction, then it reduces time complexity of the system.
(8) Space complexity is based on the storage requirements of the concurrency algorithm.

4.3 Results and discussion

Practical results captured from the previous measurements of evaluation metrics described in Sect. 4.2, among the different concurrency control algorithms are shown in

Table 4 Read operation under DFCL control

Read operation in DFCL

```
Call Begin _Transaction ();

// this code is repeated for the same transaction equal to number of requested cell for it
        If Operation=Select, then
        Set transaction _statement= "read";
        L4: Boolean x=Cell_Lock (cell#);
        If (x==0) then
                Call Read_ lock(cell#);
                Execute Read operation;
                Print <<" Read operation is successfully done";
                 Call   Release _Read_ lock(cell#);
                Set transaction _status= "committed";
                Set end _time=Current _timestamp ();

        Else
               Sleep (0.1);
               Set no _of _trans _waits=old (no _of _trans _waits) +1;
               Set transaction _status= "waiting for" ||cell#;
                If (no _of _trans _waits>selected _period) then
                     Call Abort_transaction (transaction_id);
                     Print <<" error, cell is currently written by another one";
                      Set transaction _status= "rolledback";
                       Set end _time=Current _timestamp ();
                Else
                     Goto L4;
                End If;
        End if;
```

Table 5. These results are captured by running result script (run the code to calculate the number of committed, rolled-back and waiting transactions from transaction_status column stored in process table). Then these results are stored for each run, as our experiment consists of ten runs for this simulation. At the end of this experiment, we have ten values for each metric, the final step in this experiment is the calculation of the average of these ten values for each metrics. Representation of an average number of committed, rolled-back and waiting transactions against different concurrency control algorithms is shown in Fig. 3. Representation of an average number of committed transactions among different concurrency control algorithms is shown in Fig. 4. Representation of an average number of rolled-back transactions among different concurrency control algorithms is shown in Fig. 5. Representation of an average number of committed transactions with various numbers of running transactions is shown in Fig. 6. Representation of an average number of rolled-back transactions with various numbers of running transactions is shown in Fig. 7.

Discussion: From the results shown in Figs. 3, 4, 5, we can identify that the average number of committed transactions in DFCL (with applying cell locking) is greater than that in the other concurrency control algorithms. This improvement is because of reducing blocking area to a small point called the cell. This provides more chances for the transaction to complete its operations without conflict with the others. Transaction conflict can be considered an important factor that effects on the performance of transaction and database. If this conflict presented with high percentage, this forces a large number of transactions to abort, which affects negatively on database performance. Otherwise, conflict allows a large number of concurrent transactions to succeed, which affects positively on the database performance.

As DFCL provides less percentage of conflict, this reduces the need to transaction abortion in addition to reducing the average number of rollbacks. So; the average number of rollbacks—in DFCL—is fewer than that in the other concurrency algorithms. Elimination of deadlock in DFCL prevents an infinite wait problem of transactions. So this advantage of DFCL effects positively on the number of committed transactions.

From calculating throughput of the system, we can observe that execution time for 1500 transaction is constant for all algorithms. But DFCL provides the highest number of executed transactions for the same execution time. This provides higher throughput with DFCL rather than the others. This property of DFCL improves transaction

Fig. 1 DFCL trigger for deadlock elimination

DFCL TRIGGER for Deadlock Elimination

```
// this trigger is fired until the all the transaction is either committed or rolled-back
     X=Select count (process _id) from process table where transaction_status=waiting for
// then compare using transaction id with process _id =i, j, k, ……….
       For (k=0; k<x; k++)
        {
          If (no _of _executable _statements (i)> no _of _executable _statements (j))
            then Small _id= j;
            Else Small _id= I;
         }
       Call Abort_transaction (Small _id);
         Print <<" error, cell is currently written by another one";
         Set transaction _status= "rolledback";
         Set end _time=Current _timestamp ();
```

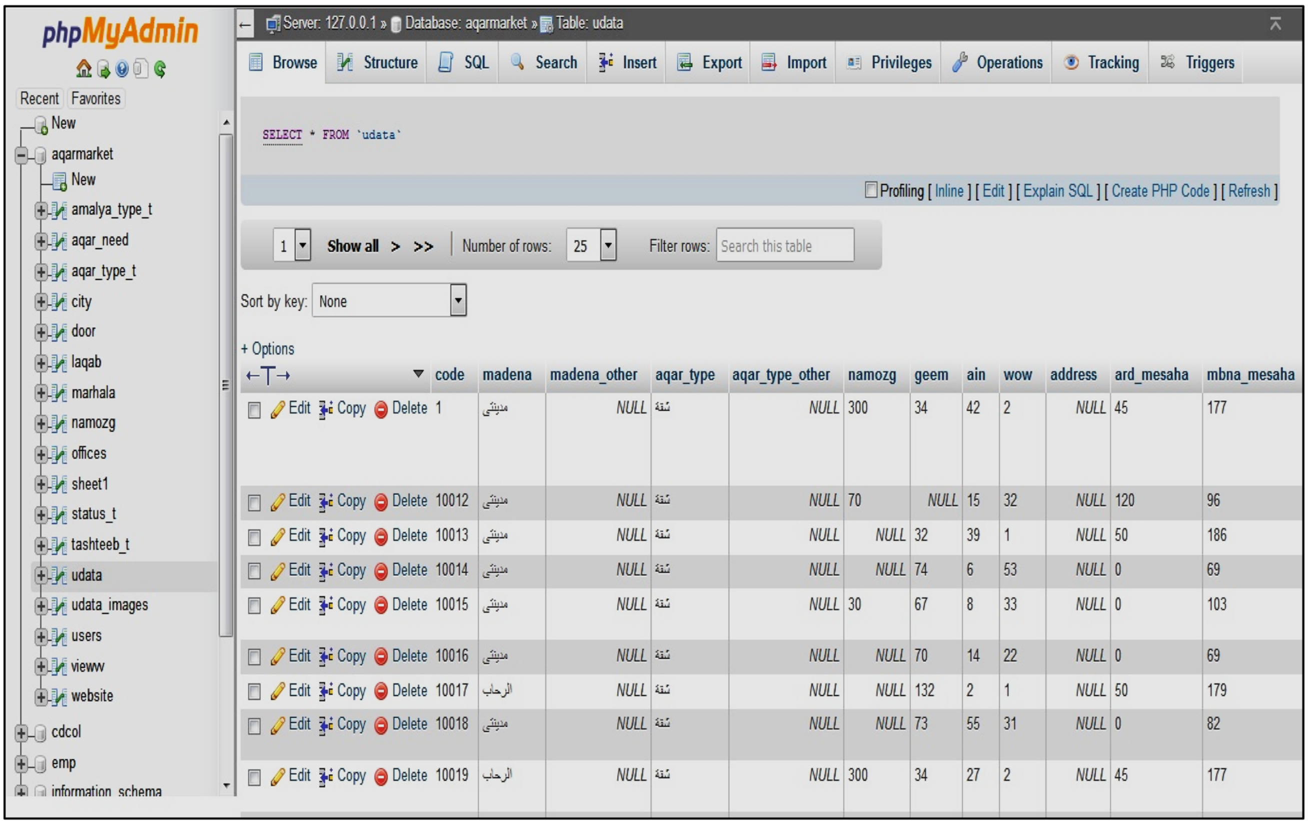

Fig. 2 Example of inserted records in our experiment

Table 5 Average number of committed, rolled-back, waiting transactions, and deadlock in different concurrency algorithms

Concurrency control algorithm	Transactions in each run	Number of runs	Average no. of committed transactions	Average no. of rolled-back transactions	Average no. of waiting transactions	Deadlock-free	Execution time (s)	Throughput (transaction/s)
2PL (file level)	1500	10	290	410	800	No	3	100
Timestamp-based concurrency	1500	10	432	1068	0	Yes	3	144
OCC	1500	10	499	1001	0	Yes	3	166
MVCC	1500	10	999	501	0	Yes	3	333
Partition locking	1500	10	1009	341	150	No	3	336
DFCL	1500	10	1234	266	0	Yes	3	411

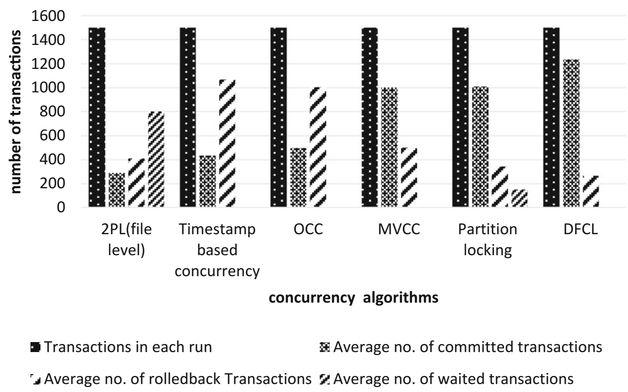

Fig. 3 Comparison among concurrency algorithms and DFCL

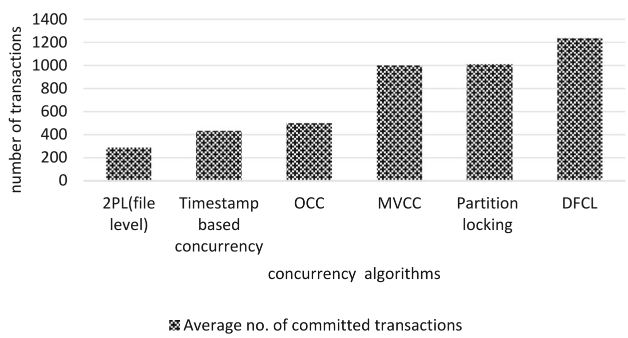

Fig. 4 an average number of committed transactions among different concurrency algorithms

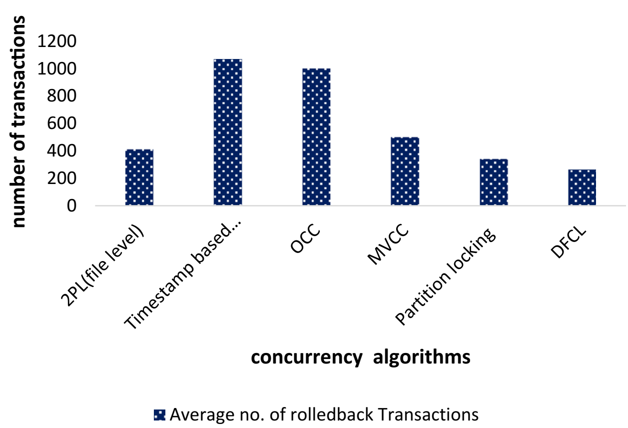

Fig. 5 an average number of rolled back transactions among different concurrency algorithms

response time that saves processing time in real execution instead of waiting without doing anything. Also DFCL improves concurrency degree, that increased with increasing throughput. Improvement of throughput, concurrency, transaction response time leads to an improvement of the database system.

As DFCL is deadlock-free, it can be better algorithm than other locking algorithms. Increasing number of

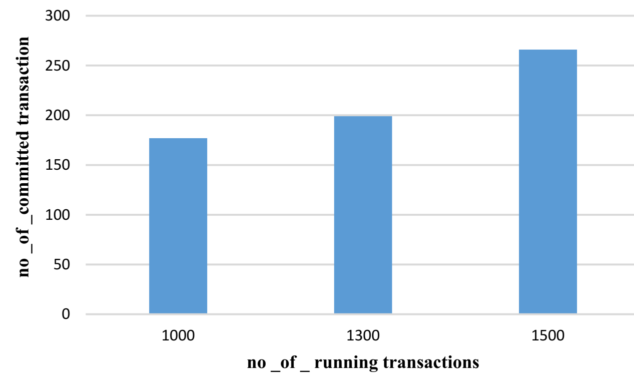

Fig. 6 average number of committed transactions for DFCL with a various number of concurrent transactions

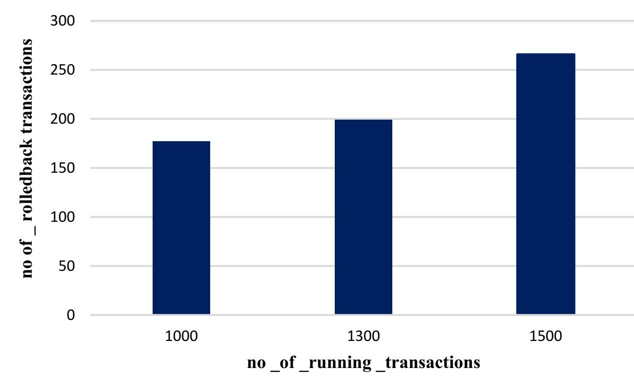

Fig. 7 average number of rolled-back transactions for DFCL with a various number of concurrent transactions

successful transactions of DFCL, make it better than other concurrency control algorithms such as timestamp, Optimistic, and multi-version concurrency control.

As DFCL has the advantages of a deadlock-free and an increasing number of committed transactions, which allows it to be the best algorithm to apply concurrency with an improvement of database performance. DFCL can be considered a useful algorithm in many fields such as banking systems because it decreases the response time of the system. As DFCL reduces wait time of the transaction, this allows the system to process a high number of operations concurrently without delay of customers. DFCL enables the system to save processor time in processing successful operations instead of waiting without doing any operation. This allows the system to process a high number of operations without delay. So DFCL reduces the average waiting time of transactions that improves time complexity of it.

DFCL compared with concurrency algorithms in this paper. 2PL achieves a high degree of locking overall large area of the database in addition to many rolled-back transactions. Timestamp eliminates the problem of

Table 6 Comparison between DFCL and other concurrency algorithms

Concurrency algorithm	Reduce blocking area	Increase committed	Reduce rolled-back	Deadlock-free	Enhance data accuracy	Enhance performance	Reduce time complexity	Reduce space complexity
2PL	No	No	No	No	Yes	No	No	Yes
Timestamp	Yes	No	No	Yes	Yes	No	No	No
OCC	Yes	No	No	Yes	Yes	No	Yes	Yes
MVCC	Yes	No	No	Yes	No	No	Yes	No
Partition locking	Yes	No	No	No	Yes	No	No	Yes
DFCL	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No

deadlock but provides a low number of committed transactions and a high number of rolled-back ones.

OCC also provides many rolled-back transactions and cannot be applied in the situations with a high degree of concurrency. Multi-version concurrency provides the highest number of committed transactions than 2PL, timestamp, and optimistic concurrency, but it has lost updates problem that effects on the correctness of data. Partition Locking Algorithm provides a satisfied number of committed, but cannot overcome deadlock problem. Finally, DFCL achieves the highest rate of transaction processing that provides the highest number of committed transactions. In addition to the elimination of deadlock problem. Because it increases the possibility of the transaction to succeed.

Decreasing wait time in DFCL enables the system to save time for processing successful-transactions. This enables the system to process many transactions concurrently, that improves system performance and response time. For these benefits of DFCL, it ensures the best performance of database and transactions. From results shown in Figs. 6, 7, we can observe that DFCL saves its improvement with different scenarios from a various number of concurrent transactions. DFCL with 1000 running transactions provides a high number of committed transactions and low number of rolled-back ones. Also, DFCL with 1300 running transactions keeps its high number of committed transactions and a low number of rollback ones. Finally, DFCL with 1500 running transactions provides a high number of committed transactions and a low number of rolled-back ones. Table 6 gives a comparison of DFCL and other concurrency algorithms; into proof the witness of DFCL.

DFCL is considered the best algorithm to improve the performance of a system and the performance of concurrent transactions. It can be used in situations that require a high number of committed transactions and a low number of failed ones. As DFCL reduces blocking area to a restricted point called "cell", it is a good method to be applied for concurrency with a low percentage of conflicts. But DFCL requires high storage requirements to store each cell with its status individually and store each transaction with its complete information. So DFCL achieves reduced time complexity and increased space complexity.

5 Conclusion

This paper proposed an improvement of 2PL to achieve deadlock free cell locking (DFCL), that improves processing of concurrent transactions. DFCL improves committed transactions, response time, throughput, concurrency and consequently database performance and reduces rolled-back ones. In addition, it eliminates deadlock problem of locking algorithms such (2PL). So DFCL is a good algorithm to be applied in the situations with a high degree of concurrency. The future work will address solutions of the high overhead of some concurrency algorithms to overcome this issue. Also, it may enhance the performance metrics of timestamp, multi-version, and/or optimistic.

References

1. Kanungo S, Morena R (2015) Comparison of concurrency control and deadlock handing in different OODBMS. Int J Adv Res Comput Commun Eng 4(3):245–251
2. Kanungo S, Morena R (2017) Issues with concurrency control techniques. Int J Electr Electron Comput Sci Eng 1–6
3. Gohil J, Dolia P (2016) Study and comparative analysis of basic pessimistic and optimistic concurrency control methods for database management system. Int J Adv Res Comput Commun Eng 5(1):178–186
4. Kaspi S, Venkatraman S (2014) Performance analysis of concurrency control mechanisms for OLTP databases. Int J Inf Educ Technol 4(4):313
5. Al-Jumah N, Hassanein H, El-Sharkawi V (2000) Implementation and modeling of two-phase locking concurrency control—a performance study. Inf Softw Technol 42(4):257–273

6. Mohamed M, Badawy M, El-Sayed A (2016) Survey on concurrency control techniques. Commun Appl Electron 5(1):28–31
7. Barghouti N, Kaiser GE (1991) Concurrency control in advanced database applications. ACM Comput Surv 23(3):269–317
8. Sippu S, Soininen E (2014) Transaction processing, data-centric systems and applications. Springer International Publishing Switzerland, Basel
9. Mamun Q, Nakazato H (2007) Timestamp based optimistic concurrency control. IEEE Region 10 Annual International Conference, Proceedings/TENCON
10. Rambol R, Imam Z, Ahmad N (2013) An efficient approach concurrency control in database management system: a performance analysis. IJCSNS Int J Comput Sci Netw Secur 13(7):29–33
11. Kung H, Robinson T (1981) On optimistic methods for concurrency control. ACM Trans Database Syst 6(2):213–226
12. Sanchez D et al (2016) TicToc: time traveling optimistic concurrency control. Proceeding SIGMOD '16 Proceedings of the 2016 International Conference on Management of Data:1629-1642
13. Rahman MD et al (2013) An efficient concurrency control technique for mobile database environment. Glob J Comput Sci Technol 13(2):17–21
14. Silberschartz A et al (2010) Database system concepts. McGraw-Hill Education, New York
15. Sadoghi M et al (2014) Reducing database locking contention sthrough multi-version concurrency. Proc VLDB Endow 7(13):1331–1342
16. Larson P et al (1986) The performance of multi-version concurrency control algorithms. ACM Trans Comput Syst 4(4):338–378
17. Lomet D, Mokbe M (2009) Locking key ranges with unbundled transaction services. Proc VLDB Endow 2(1):265–276
18. Bhargava B et al (1999) Concurrency control in database systems. IEEE Trans Knowl Data Eng 11(1):3–16
19. Mohan C et al (2002) An efficient method for performing record deletions and updates using index scans. Proceeding VLDB '02 Proceedings of the 28th international conference on Very Large Data Bases: 940–949
20. Josnes E et al (2010) Low overhead concurrency control for partitioned main memory databases. Proceeding SIGMOD '10 Proceedings of the 2010 ACM SIGMOD International Conference on Management of data 603–613
21. Thomson A et al (2012) Lightweight locking for main memory database systems. Proc VLDB Endow 6(2):145–156
22. Faleiro J, Abadi D (2015) Rethinking serializable multi-version concurrency control. J Proc VLDB Endow 8(11):1190–1201
23. Lomet D et al (2015) Multi-version range concurrency control in deuteronomy. Proc VLDB Endow—Proceedings of the 41st International Conference on Very Large Data Bases, Kohala Coast, Hawaiis 8(13): 2146–2157
24. Dashti M et al (2017) Transaction repair for multi-version concurrency control. SIGMOD '17 Proceedings of the 2017 ACM International Conference on Management of Data: 235-250
25. Wang T et al (2017) Transaction repair for multi-version concurrency control. VLDB J 26(4):537–562