Research and Improvement of SQLite's Concurrency Control Mechanism

Xiaolin ZHAO[1,a], Jianyang DING[1,b*], Rui MA[1,c], Qingyu LIN[1,d], Yining YANG[1,e]

[1] Beijing Key Laboratory of Software Security Engineering Technology, School of Software, Beijing Institute of Technology, Beijing 100081, China

[a]zhaoxl@bit.edu.cn, [b]1244262981@qq.com, [c]mary@bit.edu.cn, [d]18810278587@163.com, [e]linda_0008@163.com

Keywords: SQLite; concurrency control; adaptive; timestamp; multi-granularity

Abstract: On the basis of the study of SQLite and other concurrency control mechanisms in memory database, an adaptive concurrency control mechanism is proposed according to the operating environment of SQLite in intelligent mobile devices. The function and concrete flow of improved concurrency control are introduced and discussed. The data structure and the main functions of each part are analyzed. The function and performance of the improved concurrency control mechanism are tested. The results verify the correctness and validity of the adaptive concurrency control.

1. Introduction

In recent years, concurrency control, as one of the performance indicators of memory, has also gained rapid development, from the traditional lock-based pessimistic control mechanism to the current multi-granularity model, from the original optimistic concurrency control mechanism to a variety of improved optimistic concurrency control Mechanism, which all contributes to the upgrade of the memory database concurrency and efficiency. But SQLite is progressing slowly in concurrency control, still using traditional locking-based concurrency protocols, database-level locking granularity becomes a shackle of concurrency and results in lower degrees of concurrency. Considering the shortcomings of the original concurrency control mechanism and the characteristics of its operating environment, combined with the advantages of other concurrency control mechanisms, the original mechanism can be improved and a new concurrency control mechanism can be designed to improve the concurrency and efficiency of SQLite database[1,2].

2. SQLite Concurrency Control

2.1 Lock Mechanism of SQLite. SQLite uses extensive database granularity locking, using a harsh two-stage blocking protocol. If a transaction applies for a lock, the lock is released only when the transaction completes, and the transaction cannot perform any conflicts with the lock until the lock is released. At the same time, all the blocking requests in each transaction precede all unlock requests, the two phases refer to the lock growth phase and the lock shrink phase, 1) the lock growth phase: before any data is read and written, the transaction first obtains the data of the blockade; 2) lock shrink phase: after the release of a blockade, the transaction no longer get any lock. The SQLite system manages a lock table that enables the transaction to be locked at the last minute to ensure maximum concurrency and efficiency[3,4].SQLite contains 5 different lock states: unlocked state, shared state, hold status, pending state, and exclusive status, each transaction can only be in one of the states at the same time. In addition to the unlocked state, each state has a lock corresponding to it: 1) shared locks; 2) reserved locks; 3) pending locks; 4) exclusive locks,. A file can only have an exclusive lock, any other lock and exclusive lock cannot coexist.

2.2 Analysis of Lock Mechanism. At present, SQLite is widely used in intelligent mobile devices, in such an environment, the database concurrency is low, such as Android dialers, SMS transceivers, and telecommunications or financial account; these data may be read by multiple transactions, but in general there is only one transaction to write it[5,6]. According to the study, it is found that SQLite's existing concurrency control is based on the pessimistic concurrency control of the lock[7]. This

mechanism is mainly used in the environment of fierce competition, and cannot fully improve the efficiency and play a role in the low degree of concurrent application environment, also bring about additional overhead of the lock management and operation; At the same time, when the concurrency is high, SQLite database-level lock granularity will to some extent affect the degree of concurrency and reduce its efficiency. In the following chapters of this paper, we will discuss the multiple concurrency control mechanisms in the memory database and propose improvements to the SQLite concurrency control mechanism based on the characteristics of these mechanisms to improve efficiency and concurrency on the basis of excessive consumption of memory space[8].

3. Improvement - Adaptive Concurrency Control Mechanism

3.1 Basic Concept. In this paper, an Adaptive Concurrency Control Protocol is proposed. This protocol is mainly composed of time-based optimistic concurrency control and multi-granularity locking mechanism. Adaptation mainly refers that in the process of operation, the mechanism can be based on the current degree of intense conflict, automatic adjustment of concurrency control strategy to ensure efficiency and concurrency[9]. The default concurrency control protocol is based on the timestamp of the optimistic concurrency control, when the concurrency conflict is higher than a certain threshold, automatically converts into multi-granularity lock mechanism; when the concurrency conflict is below a certain threshold, it will be reduced to time-based Optimistic concurrency control. The mechanism can meet the high degree of concurrency and low concurrency in the operating environment to a certain extent. The use of timestamp-based concurrency control in the case of low concurrency can shorten the transaction run time to a certain extent, improving existing efficiency; and in the case of high degree of concurrent use of multi-granularity lock mechanism can improve the original degree of coherence and improve efficiency[10].

3.2 Strategy Description. When there is a transaction request, first of all, it is necessary to determine the strategy of current concurrency control, to design a global variable int state for the record of the current concurrency control type, then start concurrency control according to the strategy. The following describes the respective strategies of the two parts of the adaptive mechanism.

In the timestamp-based optimistic concurrency control strategy, need to set up a shared authentication area to record the current situation of all transactions. The shared authentication area mainly includes transaction node and conflict box. Each active transaction corresponds to a transaction node, which records the various types of information during the course of the transaction, including the read set of transactions, the set of collection conflicts, and the timestamps at different stages. The conflict box is a data structure containing capacity, slowdown, and existing bursts, used to record whether the overall concurrency control conflict is intense, adjust the control strategy by monitoring the state of the conflict box[11].

When executing a read transaction, first need to initialize the transaction node in the shared authentication area, and then read the data object required by the transaction in the independent workspace, record the timestamp, add the read set, maintain its own transaction node, then wait for the transaction to complete, can submit the transaction. And in the implementation of the transaction, similarly, first initialize the shared authentication area in the transaction node; then enter the read phase, read the transaction required data objects, and in their own independent work space to operate and deal with, while recording timestamp, update read and write collections, and maintain their own transaction node; after completing the operation and processing of the data object, enter the verification phase of the transaction: if passed, then enter the write phase and submit the transaction; otherwise accumulate the amount of existing conflicts, and detect, and finally restart the transaction.

In addition, at SQLite, a sub process is created by the master process to monitor and detect a burst box, and periodically to access the state of the conflicting amount to determine whether concurrency policies need to be converted. The specific steps are as follows: 0) sleep (T); 1) the application of signal, to ensure exclusive access; 2) in accordance with the affairs of the slow speed, reduce the existing conflict of the system; 3) to determine whether the existing conflict is more than 0, if it is,

then release the semaphore, and go to the 0), if the result is not, then go to 4); 4) to determine whether the current concurrency strategy is based on timestamp optimistic concurrency control, if the result is yes, then go to the 7), if the result is no, then the concurrency strategy is converted to time-based concurrency control, and go to 5); 5) the existing conflict is set to 0; 6) semaphore to ensure mutual exclusion; 7) Go to 0).

In this part of the multi-granularity lock mechanism, a granularity state is set for each database to record whether the current granularity of the lock is the library granularity or table granularity. A granularity factor is set for each database to record the current database; for each database to maintain a library lock information table, and record the database currently held lock information; for each table to maintain a table lock information table, record the table currently held lock information[12].

When there is a transaction request lock, the system will view the database on the lock granularity state, and according to the current state to see the current locking situation of the data object, then according to the lock compatibility matrix to determine whether there is a lock conflict, if so, then the lock particle size factor plus 1, at the same time, conducting the lock granularity detection, and the existing amount of conflict plus 1, and finally revoke the transaction; if not, then the corresponding part of the data object lock, and then update the data object lock information.

When the transaction conflicts, the system will carry out the lock granularity detection, according to the test results dynamically adjust the lock granularity. The specific situation is: If the lock granularity factor is less than the system-set lock granularity transition threshold, then the conflict of the concurrent transaction is not intense on the database. If the current database is a table granularity lock, then the upgrade operation is required, that is, the table granularity lock is upgraded to the library granularity lock. The specific process is: 1) Set the lock granularity status to the library; 2) Lock the lock granularity factor to prevent the system from "jittering"; 3) wait for the implementation of the transaction in the database, after all the execution is complete, release the table granularity lock and set the lock granularity factor to the capacity of the collision box; 4) when there is a new transaction request lock, use the library granularity lock.

If the current database is a library granularity lock, then there is no need to change the lock granularity. If the lock granularity factor is greater than the system-set lock granularity transition threshold, then the concurrency of the concurrent transaction is intense on the database. If the current database is a library granularity lock, you need to lock the lock operation, that is, the library granularity lock down to table granularity lock, the specific process is: 1) the lock granularity status is set to table; 2) wait for the database transaction execution, After the implementation of the release of the library size lock; 3) when there is a new transaction request lock, use the table granularity lock. If the current database is a table granularity lock, then there is no need to change the lock granularity.

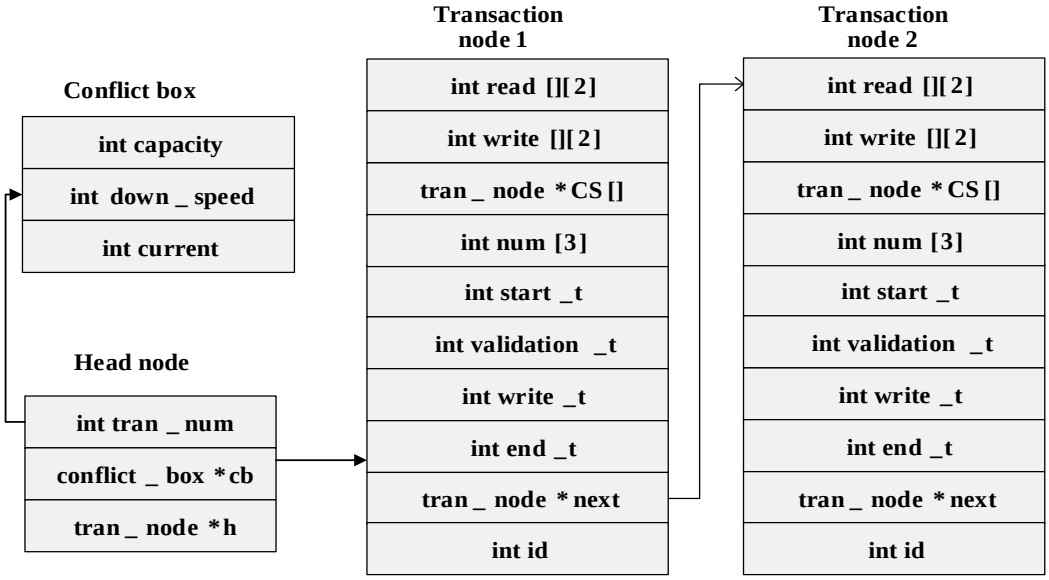

Figure 1 Shared authentication area structure

3.3 Optimistic Concurrency Control based on Timestamp. In order to implement timestamp-based optimistic concurrency control, we need to add a shared validation area. It should be noted that this must be global because it has to contain all the information about transaction, mainly includes Transaction Node and Conflict Box. The structure of the shared authentication area is shown in Figure 1.

Every active transaction corresponds to a transaction node, recording various information produced by the transaction when running, including the read, write and conflict set and timestamps of different stages. Each transaction assigns value to timestamps at different stages, record it in the transaction nodes at the same time, and carry out maintenance when processing. 'Conflict Box' is a data structure that contains capacity, down speed and current conflict. By monitoring the status of Conflict Box we can adjust concurrency strategy. The capacity of Conflict Box can be initialized by the hardware of the system when booting, it defines the max capacity that the system can tolerate. The down speed of Conflict Box means the average transaction commit speed of the current system. Current conflict means the conflict that the system adds up. Main methods of timestamp-based optimistic concurrency control are shown in Table 1.

Table1 Main methods of timestamp-based optimistic concurrency control

name	function
init_OCCP()	Initialize data of share validation area
read_OCCP()	Accomplish reading stage of OCCP
validation_OCCP( )	Validate OCCP part of transaction
des_state()	Periodically check whether it needs to change concurrency strategy
write_OCCP()	Accomplish writing stage of OCCP

If current strategy is timestamp-based optimistic concurrency control, first call read_OCCP(), includes calling sqlite3_malloc() to apply for memory and visit read and write set in shared byte section to copy needed data objects to its own working set; then starts VDBE at standalone working set to operate on byte codes; call validation_OCCP() to validate the process, if passes call write_OCCP() to write data; if not stop the transaction and call sqlite_free to free the working set; finally, call sqlite3_finalize() to accomplish the transaction. What's more, des_state() periodically monitor the status of conflict to make sure whether it needs to change the concurrency strategy.

3.4 Multi - granularity lock mechanism. In order to lock the library and table in multi-granularity lock mechanism ,we need to modify the shared byte area: Change the size of the Shared Size area to 508 bytes, and then set a lock particle factor CONFLICT_C which size is one byte after the Shared Size area, used to indicate the degree of concurrency of the transaction in the database; set a lock granularity state LOCK STATE which size is one byte, used to indicate the current database lock granularity state. The modified shared byte area is shown in Figure 2.

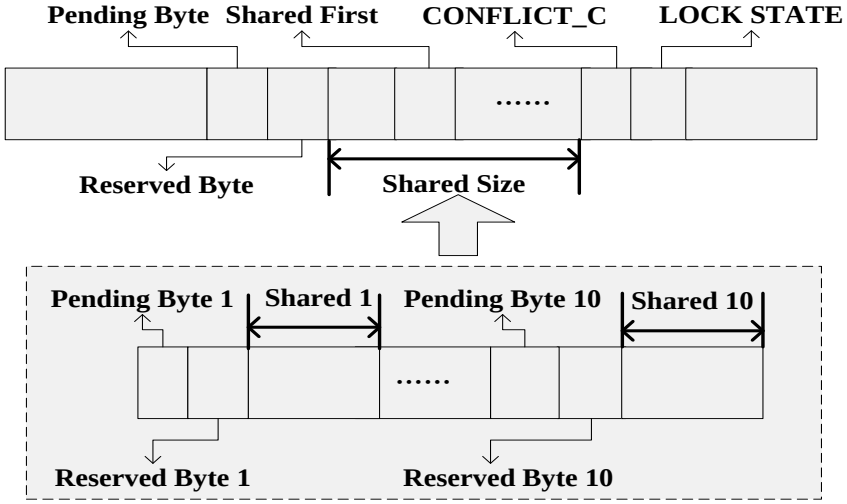

Figure 2 Modified shared byte area

When LOCK STATE is the library granularity lock state, the above modified shared area is used as the shared byte area; When LOCK STATE is the table granularity lock state, the Shared Size area in the shared area of the library is used as the table size shared byte area, and the Shared Size area is evenly distributed according to the number of tables in the database, so that each table corresponds to A shared byte field, the shared byte field of each table includes a reserved Byte area of one byte, a Pending Byte area of one byte and a Shared Size field of the size determined by the number of tables. The order corresponds to the root page number of the B tree in the main table of contents in SQLite.

When a transaction is requested, the system first checks the Lock State in the shared byte to determine the granularity of the block. If Lock State is the library granularity lock state, the operation of the shared byte area will add the entire library to unlock; If Lock State is the table granularity lock state, the operation of the shared byte area will only add the corresponding table to unlock. The purpose of this definition is to facilitate the dynamic adjustment of the lock granularity, improve the degree of concurrent affairs, but also will not increase the memory costs caused by excessive lock maintenance.

In addition, in order to record the database and database table lock information, we also need to define the lock structure, library structure and table structure. The lock structure is used to record the number of shared locks held by the current data object and other types of locks that have been obtained. The library structure and the table structure record the information of the database and the current lock.

Table 2 The main function in the multi-granularity lock

Function name	Features
Winfilelock()	Process the transaction locking process under the granularity of the library
wintalbelock()	Process the transaction locking process under table granularity
LockFile()	Achieve the lock of shared byte area under the library size specifically
LockTable()	Achieve the lock of shared byte area under the table size specifically
Dlock()	Detect lock particle size factor
winunfilelock()	Handle the process of unlocking the transaction under the granularity of library
winuntalbelock()	Handle the process of unlocking the transaction under table granularity
UnlockFile()	Achieve the unlock of shared byte area under the library size specifically
UntableFile()	Achieve the unlock of shared byte area under the table size specifically

The main functions involved in the implementation of the multi-granularity locking mechanism are shown in Table 2. Where the function LockFile () is used to lock the corresponding bytes in the library shared byte area, and to achieve the functions of the Shared lockReserved lock, the Pending lock, and the Exclusive lock to the database which are accessed. The functions Winfilelock () and Wintablelock () call the functions LockFile () and LockTable () according to the current lock granularity state to apply for various locks on the granularity or table size. In the process of application of the lock, the conflict factor will change accordingly, and then call the function Dlock () to compare the size of the conflict factor and the threshold, dynamically adjust the lock granularity between the library granularity and the table size.

4. Testing and Analysis

First download the SQLite at http://www.Sqlite.org/download.html. Improve its concurrent control mechanism, then compile in the Windows 7 system Microsoft Visual C ++ 6.0. The test database include total of nine tables named t1, t2, ..., t9, each table contains two attributes which are id and name, and a number of data has been inserted in each table.

4.1 Function test. Create five threads in the test program, while the implementation of the test database 5 concurrent read and write transactions, carry out five concurrent read and write transactions to the test database at the same time, the specific statements of each transaction are shown in Table 3. Transaction T0 to T4 come in Time1 , the transaction T5 to T9 come in Time2.

The test results are shown in Figure 3. It can be seen from the results that in the SQLite system using the adaptive concurrency control mechanism, for the 10 read and write transactions, which read transaction T0, T1, T4, T5 and write transaction T7, T9 can be successfully executed, But write transaction T2, T3, T6, T8 return error message. At Time1, the system is in optimistic concurrency control based on timestamp, allowing multiple read transactions to execute concurrently, and writing transactions are not allowed. When read transaction T0 read the database, the transaction T1, T4 can read the database, and write transaction T2, T3 will be read and write conflict when verifying, can not be verified, and return error message. At this point, the amount of conflict box is accumulating constantly. When the capacity is exceeded, the concurrency control policy becomes the library granularity lock mechanism. As the current test database have read transaction (T0, T1, T4) and shared lock, so write transaction T2, T3 can not be carried out. Need to wait for all read transactions are completed and then run separately (due to the current lock granularity for the library, so the transaction T2, T3 can not be carried out at the same time).

Table 3 Transaction test and main statement

Transaction name	Main statement
T0	select id, name from t1 where id=’12’;
T1	select id, name from t1 where id=’12’;
T2	update t1 set name=’aa1’ where id=’11’;
T3	update t3 set name=’cc1’ where id=’31’;
T4	select id, name from t4 where id=’41’;
T5	select id, name from t5 where id=’51’;
T6	update t6 set name=’hh1’ where id=’61’;
T7	update t7 set name=’gg1’ where id=’71’;
T8	update t7 set name=’gg1’ where id=’72’;
T9	update t9 set name=’ii1’ where id=’91’;

Transaction T5 in Time2 add a shared lock to the database, transaction T6 can not be carried out. At this point, the lock particle size factor CONFLICT_C constantly increase, When CONFLICT_A as the threshold for lock granularity is reached, the system will do the lock operation, lock granularity reduced to table size. So write transaction T7, T9 can be successfully executed. However, in the table granularity lock state, multiple write transactions operate on the same table at the same time is not allowed. So when transaction T7 do write operation to table t7, transaction T8 can not do write operation to table t7,and transaction T8 return error message.

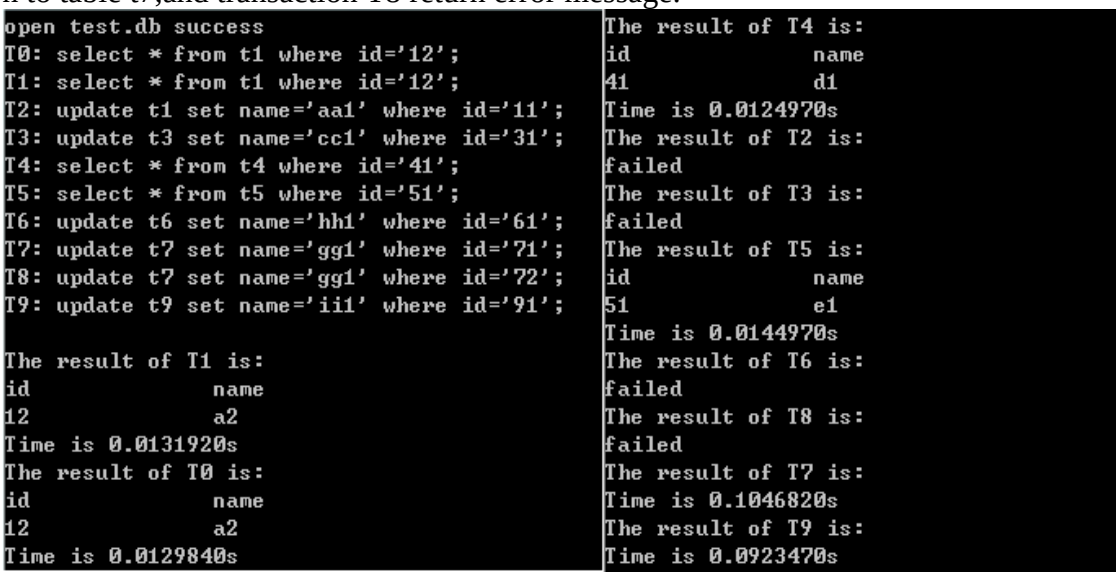

Figure 3 Results of the functional test

4.2 Performance Testing. We simulate the operating environment with less business applications and low concurrency in smart mobile devices, Run 100 transactions respectively that the concurrency is 0%,10%,20%,30%,40%,50%. Get the entire transaction set from the beginning to the end of the

time By calling the time function provided by SQLite. The result of its implementation is shown in Figure 4. It should be noted that, because the test simulate the embedded system environment in the Windows system with more adequate hardware resources, so the efficiency is not obvious. If you are testing in an embedded environment, you can expect a significant improvement in efficiency. And test in the embedded real case is the next step of the work focus.

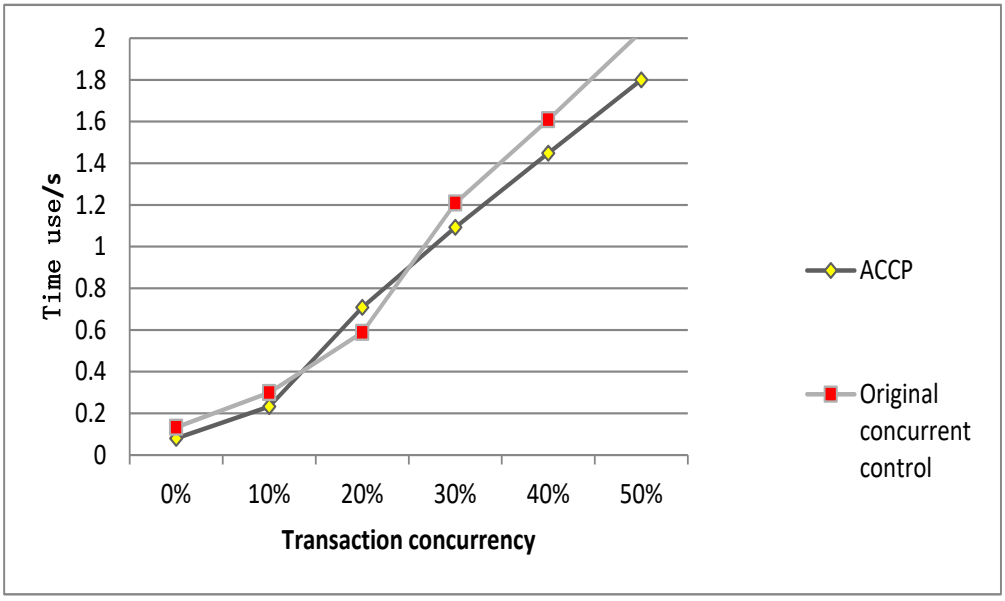

Figure 4 Results of the performance test

Analyze the single data node of the comparison graph, from the single data node analysis of the comparison graph, in the running environment of a small amount of transaction, the concurrency control mechanism has a certain improvement in the case of low concurrency degree (0%,10%)and high degree(30%,40%,50%) of concurrency. From the overall analysis of each comparison chart, the slope of the original parallel control mechanism has become more and more in the case where the running environment concurrency of a small amount of transaction (60,80,100) is increasing continuously. And the slope of the line graph of the adaptive concurrency control mechanism shows the first smooth, then jitter, and then smooth.

4.3 Analysis of Adaptive Concurrency Control Mechanism. In the functional test, it is verified that the adaptive concurrency control mechanism can adjust the concurrency control strategy according to the current system condition, that is, it has the ability of adaptive.

In the performance test, the time efficiency of the adaptive concurrency control mechanism is mainly tested. In the case of a small amount of transactional operation, the adaptive concurrency control mechanism has a slight improvement in the efficiency of concurrency in the case of low concurrency (0%, 10%) and high concurrency (30%, 40%, 50%). This is because in the case of low concurrency, the adaptive concurrency control mechanism mainly adopts the timestamp-based concurrency control, which reduces the locking time compared with the original concurrent control, but the efficiency improvement is not obvious. In the case of high concurrency, the adaptive concurrency control mechanism mainly adopts the multi-granularity locking mechanism, which improves the concurrency degree compared with the original concurrent control, greatly reduces the time of restart and wait of the transaction, and the efficiency is improved obviously. And adaptive concurrency control mechanism in the case of special concurrency (20%), but not as good as the original control of high efficiency, This is because in this degree of concurrency, the adaptive concurrency control mechanism wastes time in a number of policy conversions, resulting in a reduction of efficiency. At the same time, in the case of increasing coherence, the slope of the original concurrency control mechanism is becoming larger and higher, and the slope of the adaptive concurrency control mechanism is stable, At the same time, in the case of increasing coherence, the slope of the original parallel control mechanism is larger, while the adaptive concurrency control mechanism slope shows a situation is first stable, after the jitter, and then smooth. This is because the

original concurrency control mechanism has remained the same, and can not meet the requirements of concurrent affairs. The adaptive concurrency control mechanism also increases the concurrency degree from the stable use of timestamp-based concurrency control to the stable use of multi-granularity locking mechanism, which can meet the requirements of concurrent transaction. The middle of the jitter is also caused by a number of strategic changes.

5. Conclusion

This paper studies the SQLite and concurrency control. According to the operating environment of SQLite in intelligent mobile devices, an adaptive concurrency control mechanism is proposed to improve the concurrency and efficiency of SQLite database, and the improved concurrency control mechanism is designed, implemented and tested. The SQLite system, which adopts the adaptive concurrency control mechanism, can alleviate the competition situation of the database in the limited resource environment, and can meet the demand of the complex environment.

Funded by the National Key Research Program (Project *No. 2016YFB0800700*)

References

[1] J. Song, S. Wei and K. Le: Research and Improvement of Locking Mechanism in Embedded Database Based on SQLite[J]. COMPUTER SCIENCE AND TECHNOLOGY, 2010, 47(suppl.): 228-234.

[2] Y. Song: Research and Improvement in Locking Mechanism of Embedded Database SQLite[J]. COLLIERY ENGINEERING, 2010,29(9):192-194.

[3] M. Jamal and N. Ahmad Zafar: Extending agent-based Mobile Petri Nets with access control[J]. International Conference on Communication, Computing and Digital Systems (C-CODE), Islamabad, 2017, pp. 133-138.

[4] C. P. Neumann, A. M. Wahl and R. Lenz: Adaptive Version Clocks and the OffSync Protocol[J]. IEEE 10th International Symposium on Parallel and Distributed Processing with Applications, Leganes, 2012, pp. 835-836.

[5] W. Zhao, H. Zhang, X. Luo and Y. Zhu: Enable Concurrent Byzantine Fault Tolerance Computing with Software Transactional Memory[J]. 8th International Conference on Advanced Software Engineering & Its Applications (ASEA), Jeju, 2015, pp. 67-72.

[6] C. Sang, Q. Li and L. Kong: Tenant Oriented Lock Concurrency Control in the Shared Storage Multi-tenant Database[J]. IEEE 16th International Enterprise Distributed Object Computing Conference Workshops, Beijing, 2012, pp. 179-189.

[7] T. Kang, S. Zhang and L. Kong: A Multi-tenant Level Lightweight Lock Mechanism for Multi-tenant Database[J]. 11th Web Information System and Application Conference, Tianjin, 2014, pp. 3-7.

[8] F. Zendaoui and W. K. Hidouci: Performance evaluation of serializable snapshot isolation in PostgreSQL[J]. 12th International Symposium on Programming and Systems (ISPS), Algiers, 2015, pp. 1-11.

[9] E. Kooli and N. Aissaoui: Predictive speculative concurrency control for real-time database systems[J]. Information and Communication Technologies Innovation and Application (ICTIA), Sousse, 2014, pp. 1-7.

[10] A. Ghosh, R. Chaki and N. Chaki: Checkpoint based multi-version concurrency control mechanism for remote healthcare system[J]. International Conference on Advances in Computing, Communications and Informatics (ICACCI), Jaipur, 2016, pp. 382-389.

[11] P. Tomar and Suruchi: Efficient concurrency control mechanism for distributed databases[J]. 3rd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, 2016, pp. 3415-3418.

[12] S. A. Bakura and A. Mohammed: Lock-free hybrid concurrency control strategy for mobile environment[J]. IEEE 6th International Conference on Adaptive Science & Technology (ICAST), Ota, 2014, pp.1-5.