FORMAL VERIFICATION OF AN ACCESS CONCURRENCY CONTROL ALGORITHM FOR TRANSACTION TIME RELATIONS

Achraf Makni, Rafik Bouaziz, Faïez Gargouri
Faculty of Economic Sciences and Management of Sfax BP 1088, 3018 Sfax, Tunisia

Keywords: Optimistic concurrency control, Strong consistency, Transaction time relation, Formal verification, PROMELA/SPIN.

Abstract: We propose in this paper to formally check the access concurrency control algorithm proposed in (Bouaziz, 2005). This algorithm is based on the optimistic approach and guarantees strong consistency for the transaction time relations. The specification of our model under PROMELA language allowed us to ensure the feasibility of the validation. We then could, using the SPIN model checkers, avoid errors of blocking type and check safety properties specified by temporal logic formulas.

1 INTRODUCTION

The access concurrency control (CC) to a database (DB) is an essential component in a database management system (DBMS). It must guarantee that the simultaneous execution of transactions produces the same results as a sequential execution (Gardarin, 1988). In some environments, this mission must be reinforced to ensure the strong consistency of the databases. To do so, the CC must guarantee that the simultaneous execution of transactions produces the same results as the sequential execution of these transactions in their strict order of arrival (Rahgozar, 1987).

The CC takes new dimensions when applied to the temporal DB (TDB). TDB have as objective the management of the data history, such as it is the case for the transaction time relations (TTR). The objective of these relations consists in providing for the applications, not only the current data, but also all the previous DB states which succeed in time. To be able to maintain these states, the update operations should not be destructive. The TTR store the passed versions by stamping them using the two following physical times:

- *beginning transaction time* (BTT): the execution time of the transaction which inserts the corresponding tuple. This time is *a priori* known.
- *end transaction time* (ETT): the execution time of the transaction which updates or removes the considered tuple. It can not be *a priori* known.

We propose in this paper to check formally the optimistic concurrency control algorithm OCCA_SC/TTR (Bouaziz, 2005), which ensures the strong consistency (SC) for TTR. To do so, we chose to use the SPIN tool (Holzmann, 1997), which is one of the powerful model checkers. It is an appropriate tool for analysing the logical consistency of concurrent systems, especially for the data communication protocols. SPIN is largely used, not only in the research areas, by the fact that it is freeware, but also in the industrial area (Gnesi, 2000), (Havelund, 2001), (Brinksma, 2002), (Berstel, 2005).

This paper is organized as follows. Section 2 describes the structure of OCCASC/TTR algorithm. Section 3 presents some results of the verification step using SPIN model checker.

2 DESCRIPTION OF OCCA_SC/TTR ALGORITHM

The CC methods are classified according to the two main categories; pessimistic methods and optimistic ones. In the pessimistic methods the checking consistency is carried out at the time of each

Makni A., Bouaziz R. and Gargouri F. (2006).
FORMAL VERIFICATION OF AN ACCESS CONCURRENCY CONTROL ALGORITHM FOR TRANSACTION TIME RELATIONS.
In *Proceedings of the Eighth International Conference on Enterprise Information Systems - DISI*, pages 269-272
DOI: 10.5220/0002487602690272

transaction operation. In the optimistic methods, checking consistency is carried out only at the end of transaction. For the TTR, we can find, in the literature, some CC algorithms based on the pessimistic approach (Finger, 1997) (Elloumi, 1998) (Castro, 1998). But, to our knowledge, only the OCCA_SC/TTR algorithm (Bouaziz, 2005) was proposed to study the CC for these relations according to the optimistic approach in order to ensure the strong consistency of the database.

The OCCA_SC/TTR algorithm allows to maintain the strong consistency of the DB in a TTR environment, to minimize the abortion degree of transactions, to avoid the starvation problem and to detect conflicts as soon as possible.

For each transaction T_i, the concurrency controller maintains two sets: RS_i (Read Set), the set of objects read by T_i, and WS_i (Write Set), the set of objects written by T_i.

During the transaction execution, when the concurrency controller receives:

- a Read (T_i, g) operation, it adds the g granule to RS_i;
- a Read (T_i, g, pt) operation, it adds the g granule to RS_i only if pt indicates the current version of g; but, this read operation cannot, in any case, produce conflicts;
- a Write (T_i, g) operation, it adds the g granule to WS_i;
- a Rollback operation, it eliminates the read and written objects from RS_i and WS_i;
- a Commit operation, it checks if there is or not a conflict between the transaction to be validated and the transactions which are not yet validated.

In our work, we started from the validation strategy of BOM algorithms (broadcast optimistic method) with critical section (CS). This strategy stipulates that at each execution of a COMMIT order concerning a transaction T_i, at one moment t, concurrent transactions, which are still in their reading phases, must do a validation test with T_i ($WS_i \cap RS_j$). If there is a conflict, the transaction to be aborted is the one having the least priority.

To be able to ensure strong consistency, we propose to proceed **to the stamping of the transactions by the moments of their arrival** and to attribute to the last coming one the least priority. We propose also to add **a certification phase** which precedes the validation one of each transaction. During this phase, the concurrency controller checks that T_i has the most priority. In this case, the concurrency controller passes it to the validation phase. In the opposite case, T_i is put in a waiting list to be certified later on.

Once arrived at its validation phase, T_i will be automatically validated. The new versions of granules manipulated by this transaction will be stored in the database and will take as stamp the transaction time of T_i (equal to t_i, the arrival T_i moment).

A research of the conflicts, which can exist between T_i and any transaction T_j in reading or in certification phase, is then carried out. T_j is necessary younger than T_i and thus having the less priority. Consequently, if there is a conflict, T_j must be aborted to be taken again with the same stamp.

After validating the transaction T_i, CC must always check if there is a transaction T_k, waiting for certification, which becomes the most priority. Indeed, the setting on waiting for certification of a transaction is due to the existence of others having more priority and not yet validated. Then propose to add an awaking function.

The CS, during which all the manipulated granules in writing by T_i must be locked, extends during the two writing and validation phases of the transaction T_i. But we successfully reduced this period using the "EOT marker" technique for a correct definition of the conflicts. The period of enf of transaction marking is much shorter than the whole validation phase, also including the conflict checking.

3 OCCA_SC/TTR VALIDATION

The systems analyzed by SPIN are described with the PROMELA language (PROcess MEta LAnguage). PROMELA is a specification language for finite state systems. A system specified by PROMELA is represented by a set of parallel processes and communicating via global variables or/and communication channels. PROMELA also allows checking properties specified in linear temporal logic (LTL).

We use, in the following, an example of three transactions (T_1, T_2 and T_3) and two granules (x and y). The transaction T_1 manipulates in reading and in writing both granules, whereas the transactions T_2 and T_3 manipulate in reading and writing respectively the x granule and the y one. Thus, the conflict risk between the transactions is limited between T_1/T_2 and T_1/T_3. These transactions are maintained in the `liste_tr` array.

transaction `liste_tr[nb_tr];`

We defined also a new "transaction" type which gathers the transaction characteristics.

```
typedef transaction
{
```

```
  byte nom;
  byte ordre;  /* represents
     the transaction stamp
     (arrival moment)  */
  byte ordre_validation;
    /* Indicates a transaction
    validation order */
   .
   .
   .
  };
```

The SPIN model checking can proceed in two steps. In the first one, "deadlock" or "unreachable code" errors are detected. In the second step, the validity of the quality properties of the system is checked through the application of an adequate LTL formula. In the case of error, SPIN gives the shortest way which leads to this error.

3.1 First Checking Step

With the first version of our system, SPIN detects the possibility of a blocking situation. The shortest way which leads to this error is described below.

The priority orders allotted to our three transactions is: $T_1 > T_2 > T_3$.

The transactions T_3 and T_2 aren't certified regards to the reading transactions. T_1 continues its execution, it is certified and it starts the validation phase. Since there is a conflict with T_2 and T_3, theses latter are aborted. If the transaction T_3 takes again its execution and demands its validation from the CC before T_2, it will be blocked again in the certification test regards to the transactions in reading phase, since T_2 has now the most priority. So, when T_2 starts again its execution, it passes the two certification tests successfully and starts the validation phase. Since there is no conflict between T_2 and T_3, the latter is awaked by the CC after the T_2 validation. T_3 will passes only the certification test regards to the transactions in reading phase, but the result is negative, because T_2 is still considered in reading phase. When the T_2 state takes the value "finish", the transaction T_3 will be blocked, although it has the priority in the system. Besides, it will persist in this state, since the CC cannot any more awake it.

We note, by what precedes, that the attribution of the "finish" value to the element "state", defined in the "transaction" type, should not be carried out after the awake of a concurrent transaction which has now priority. This will lead again to blocking this transaction.

In order to resolve the problem we propose that a transaction must take the finish state before calling the awaking procedure.

No error was reported by SPIN for this new version. The checking of the model was effected by using the exhaustive research mode and the partial order reduction algorithm.

3.2 Definition and Application of LTL Formulas

Let's remind that we already defined the two elements "order" and "ordre_validation" in the "transaction" type. The "order" element is defined in accordance with the transaction arrival order. Whereas the element "ordre_validation" represents the transaction validation order. Each transaction stamp value is assigned to element "order" before starting the parallel execution of all transactions.

To make sure that our system guarantees SC, we must have at the end of the execution, for any transaction arranged with the element of i index in the liste_tr array, the element "order" equal to the element "ordre_validation". So, we defined the property p as follows:

```
  #define p
(liste_tr[0].ordre ==
liste_tr[0].ordre_validation)
```

The LTL Formula which we applied is as follows: "`<>[]p`".

"<>[]p" means that there is at least a state from which we will have the property p true forever.

In our system, the priority and validation orders are initially different. This is true since the assignment of priority order is carried out at the beginning of the execution. Whereas, the assignment of validation order is carried out when a transaction is validated. This justifies the use of the operator eventually.

No error is detected in this checking phase when applying the formula `<>[]p`.

After having checked that SC is ensured, we will check, hereafter, that in the case of a conflict, the transaction with the least priority will be aborted. Our second formula is then based on the values which x and y granules can take.

To do so, we defined two global variables `xval` and `yval`. These last represent the values which can take each x and y granule. We suppose that each transaction, when modifying a variable, gives it a specific value: when T_1 modifies x, the value of the granule will take the value tr1. At the execution end, the granule's final value must be equal to the value assigned by the least priority transaction. If it is not the case, it means that there is a not solved conflict between two transactions (the least priority transaction was not aborted).

Let us remind that the two transactions T_1 and T_2, which correspond respectively to the index 0 and 1 in the `liste_tr` array elements, manipulate the x granule in reading and writing. If these two transactions are executed simultaneously, a conflict can occur. The LTL formula, described below, allows to check if this conflict is solved or not (if it appears).

The LTL formula is as follows:

```
[]((<>(a&&b)-><>c)&&(<>(!a&&d)-><>e))
```

The properties a, b, c, d and e are defined as follows:

```
#define a
(liste_tr[0].ordre <liste_tr[1].ordre)
#define b (liste_tr[0].state==finish)
#define c (xval==tr2)
#define d (liste_tr[1].state==finish)
#define e (xval==tr1)
```

This LTL formula treats the two possible cases between T_1 and T_2 according to their priority orders.

Case 1:

if $T_1 > T_2$ ("a" = true) and if T_1 is finished ("b"=true) → we must be sure to have :
"c" = true in a future state (xval="tr2").

Case 2:

if $T_1 < T_2$ ("a" != true) and if T_2 is finished ("d"=true) → we must be sure to have :
"e" = true in a future state (xval="tr1").

The application of this formula gives a valid result.

4 CONCLUSION

In this paper, we checked that OCCA_SC/TTR operates correctly. We showed formally, using SPIN tool, that the general working of our system is correct. Nevertheless, this formal verification permits us to find some insufficiencies and to resolve an error problem relating to the moment when a transaction must have the finished state. We showed that the state of a transaction T_i must have the value "finish" before making awake another transaction T_j.

In addition, the definition and the application of the two LTL formulas, using SPIN, enabled us to check that the strong consistency of the database is maintained, on the one hand, and that in the case of a conflict between two transactions, this conflict is solved by aborting transaction having the least priority, on the other hand.

Our future work aims at the validation of this algorithm, using a complete study case, and to show that it ensures better performances compared to those of pessimistic algorithms presented in the literature.

REFERENCES

Bernstein, P, A., Hadzilacos, V., Goodman, N., 1987. Concurrency control and recovery in DBS. ADDISON-WESLEY Edition.

Berstel, J., Reghizzi, S, C., Roussel, G., Pietro, P, S., 2005. A Scalable Formal Method for Design and Automatic Checking of User Interfaces. *ACM Transactions on Software Engineering and Methodology*.

Brinksma, E., Mader, A., Fehnker, A., 2002. Verification and Optimization of a PLC Control Schedule. *Journal on Software Tools for Technology Transfer*.

Bouaziz, R., Makni, A., 2005. ACCO_CF/RTT: Un algorithme de contrôle de concurrence optimiste pour les relations temporelles de transaction. *Information Sciences for Decision Makin, Janvier 2005*.

Castro, C., 1998. On concurrency management in temporal relational databases. *In SEBD'98*.

Elloumi, S, D., Bouaziz, R., Moalla, M., 1998. Contrôle de concurrence multiversion dans les bases de données temporelles. *In BDA'98. International Conference on advanced databases*.

Finger, M., McBrien, P., 1997. Concurrency Control for Perceivedly Instantaneous Transactions in Valid-Time Databases. *In TIME'97*.

Gardarin, G., 1988. Base de données : Les systèmes et leurs langages. EYROLLES Edition.

Gnesi, S., Lenzini, G., Latella, D., Abbaneo, C., Amendola, A., Marmo, P., 2000. An Automatic SPIN Validation of a Safety Critical Railway Control System. *In DSN'00. International Conference on Dependable Systems and Networks. Published by IEEE Computer Society Press*.

Havelund, K., Lowry, M., Penix, J., 2001. Formal Analysis of a Space Craft Controller using SPIN. *IEEE Transaction on Software Engineering.*

Holzmann, G, J., 1997. The model cheker spin. *IEEE Transaction on Software Engineering*.

Rahgozar, M., 1987. Contrôle de concurrence par gestion des événements. PhD thesis.