Enhancing External Consistency in Real-Time Transactions *

Kwei-Jay Lin and Ching-Shan Peng

Department of Electrical and Computer Engineering
University of California, Irvine, CA 92717
{klin, cpeng}@ece.uci.edu

1 Introduction

Real-time computing brings two new requirements to database management. The first is the deadline constraint which requires better scheduling algorithms to meet transaction deadlines. The second requirement is the temporal validity, or *external consistency*[Lin89], requirement of data: a real-time database must prevent data from being corrupted not only by the executions of concurrent transactions but also by the delay from sharing computing resources and real-time scheduler decisions. In this paper, we study the external consistency requirement in real-time databases. We propose a semantic-based concurrency control scheme that prefers the external consistency to the traditional serializability.

Two new concepts are introduced in our work. An update of a physical object value is considered to be compatible to those read operations that need to access the most recent data. Secondly, if a transaction depends strongly on the validity of the value of an object, a new update may cause the transaction to be aborted since continuing the transaction may be harmful to the application. A real-life example is the stock market database. A trading program may be executed for a long time and its read operations should not block any new updates. Moreover, if there is a big sudden change in a particular stock price, it may be better to abort an on-going transaction rather than letting it make some invalid decisions.

2 RTDB model

Following the terminology used in [PL95], we define a real-time object-oriented database to have a database manager, a set of data objects, and a set of transactions (Figure 1). A database may be connected to external devices which are usually some kind of sensors or data collection channels. Without loss of generality, data objects are partitioned into two categories: *external data* objects (circles in the figure) that correspond to some physical objects or attributes of the real world, and *derived data* objects (boxes in the figure) that are created and defined by the database as part of transaction processing. External input devices will initiate *update transactions*[AGMK95] to *update* the external object values in the database. On the other hand, *user transactions* may read from external data, read from derived data, and/or write derived data. That is, a user transaction may include any combination of the above operations.

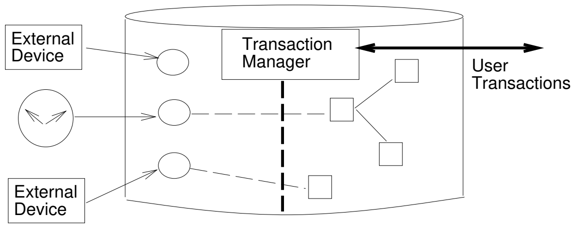

Figure 1: Real-Time OODB Model

The external consistency problem arises primarily between the device input operations and operations accessing that external data. The traditional concurrency control protocols try to resolve data contention based on the principle of serializability; the external consistency has not been taken into consideration. Therefore, the execution schedule produced may not guarantee the external consistency of data used by transactions and thus inappropriate for real-time applications.

An OODB consists of many objects. Each object has some internal state which is protected by the object type abstraction. Formally, using a notation similar to [DW93], an object type is defined by $< N, A, M, CM >$. The component N is the name of the object type and A is the attribute of the object. Each attribute has a value and a timestamp which refers to when the value becomes valid. M is a set of methods defined. The only way the object state can be accessed from transactions is to invoke one of the methods M defined by the object type. Each method has some input parameters, output parameters, and scheduling (or timing) parameters. At any time, more than one transactions may request and execute methods in the same object. In our model, several concurrent executions of the same or different methods are allowed as long as the Compatibility Matrix (CM) for the object has been so defined.

Suppose an object has n methods defined. The CM has

*This work was supported in part by contracts from the Office of Naval Research N00014-94-1-0034 and N00014-95-1-0262, Hughes Aircraft/UC MICRO 94-082 and 95-097, and the US Navy NSWCDD N60921-94-M-1261 and N60921-94-M-2714.

Table 1: An Example Compatibility Matrix (CM)

	Get(EX)	Get(UP)	Get(AB)	Set()
Get(EX)	$\surd$	$\surd$	$\surd$	X
Get(UP)	$\surd$	$\surd$	$\surd$	$\surd$
Get(AB)	$\surd$	$\surd$	$\surd$	X
Set()	X	$\surd$	$\surd^*$	X

$\surd^*$: The holding Get lock aborts.

$n \times n$ entries (e.g. Table 1). $CM(a, b)$ will be checked if an instance of method a is being executed while method b is requested by a transaction. The possible values in $CM(a, b)$ are $\surd$ (allowed) and X (reject). If it is an $\surd$, b can be scheduled for execution as long as some system time is available. Otherwise, b will be placed in the block queue. Also associated with an entry could be some special operation, like to restart a or b, or to promote the priority of a as defined in the Priority Inheritance Protocol [LSL90].

2.1 External consistency

Much research has been done on reasoning the correctness of various concurrency control protocols for databases. Traditionally, this is done by showing the "equivalence" of one execution history to another. For example, it has been shown that a "serializable" schedule is equivalent to a serial execution schedule of transactions on a database. Using the serializability criterion, many have worked on the real-time concurrency control issue using various protocols. For our discussion in this paper, we will not consider serializability as the only correctness criterion as we are more interested in the temporal validity of data used by transactions.

Define the *history* H of an object as a sequence of operations executed by the object on its attributes in responding to method invocations by transactions. Each operation is a read , update, or write on an object attribute. Each operation also has a timestamp on when the operation is performed, e.g. $r(x, 3)$ is a read operation on attribute x at time 3. If it is necessary to identify the transaction by which the operation is requested, a subscript can be denoted, e.g. $r_2(x, 3)$ is a read operation requested by transaction T_2. The following example history performs some operations on attributes x, y, z:

$$H = w(x, 1), w(y, 2), up(z, 3), r(y, 4), r(x, 6), r(z, 7)$$

Let us now define the concept of *external consistency* (EC).

History H_1 is more externally consistent than H_2, or $H_1 \succ H_2$, if

1. $H_1 - \cup_i up(-, i) = H_2 - \cup_i up(-, i)$;
2. for all attributes read in any operation of H_2, H_1 has a closer or the same update operation before that operation. Moreover, H_1 has a closer update before at least one of the read operations.

For example, $H_1 \succ H_2$ in the following two histories:

$$H_1 = up(x, 1), w(y, 2), r(y, 3), up(x, 4), r(x, 5)$$

$$H_2 = up(x, 1), w(y, 2), r(y, 3), r(x, 5)$$

From the definition, for each history H, we can define a $\succ$-lattice. $H_\top$ is defined by inserting an update operation immediately before each read in H (i.e. the update has the same timestamp as the read). $H_\perp$ is defined by removing all update operations from H except the very first update for each attribute. In the $\succ$-lattice, we can define the EC of a history H: H has a $EC = t$ if all read operations in H on any attribute are within t time units from the last update on that attribute. Therefore $H_\top$ has $EC = 0$.

Using the above definition of $\succ$-lattice, we can justify the correctness of a history H using any concurrency control protocol by showing its $EC < B$ where B is the external consistency bound. In this way, it is quite easy to see that many serializable histories are not acceptable from the external consistency's point of view.

2.2 Semantic-based concurrency control

A semantic-based CM example is shown in Table 1. The top row in the table represents the methods currently being executed and the leftmost column represents the method being requested. In order for the transaction manager to make an appropriate schedule, the compatibility matrix of the object is examined to check whether a semantic lock could be granted.

There are three different types of *Get* method depending on the different semantics. One can hold an exclusive read lock (EX), a read lock shared with update (UP), or a lock which must be aborted (AB) if a new *Set* method is requested. The advantage of providing these alternatives is to allow the cooperative transaction to satisfy the external consistency requirement. In other words, it's up to the transaction to decide which method is to be requested. The concurrency control protocol can enforce or relax the serializability property based on the transaction semantics. For example, if the transaction needs to maintain a strict serializability, it can request for an Get(EX) lock. If a timely result is more desirable, a Get(UP) lock can improve the concurrency such that it is not blocked by concurrent updates. A Get(AB) lock will release the lock to the update transaction and force the transaction to restart to achieve a better external consistency.

The flexibility of using the semantic-based CM can be seen from the following example. Suppose a transaction T_1 is to calculate a result using some external data x and y:

$$T_1(x, y) = f(x, y) + g(x, y)$$

Another transaction T_2 is to update the value of x

$$T_2(x) = h(x)$$

Assume that the functions f and g are invoked in sequence, and transaction T_1 is executing f while T_2 requests the update. If we use the traditional concurrency control protocols like 2PL, the lock held by x cannot be released so that there is no way to execute T_2 promptly. Consequently, T_1 cannot use the fresh data to process. Even though the serializability is maintained, the result might be out-of-date and useless. Instead, we can use *Get(UP)* to "anticipate" the coming update to x. When this situation happens, the update request on x can be granted so that T_2 can start to update. Transaction T_1 thus can use the more up-to-date information to process g.

We have studied the performance of the new concurrency control protocol by simulation. Our results [PL95] show that the CM protocol is quite effective in enhancing the external consistency of real-time transactions.

3 Future work

One possible future work is the integration of the concurrency control protocol with the real-time scheduling algorithm for transactions. For example, when the system uses the priority driven scheduling, PCP and SRP have been proposed to bound the potential blocking due to priority inversions. In our study, we have evaluated the CM performance by simulation. A more rigorous study on the scheduling condition will be needed so that the protocol can be used to implement hard real-time systems.

Another issue is that update transactions may need to have different priorities to arbitrate their data correctness. For example, some input device may provide more precise data than others, or some input channel or source may be more reliable than others. In our current model, CM has two parameters: method type and lock type. It is possible for us to add the third parameter to arbitrate the importance (or criticalness) to CM. In fact, there is no theoretical reason why any number of parameters cannot be used in CM. The tradeoff is between the power and the efficiency. More work is needed to select the best practical CM model.

References

[AGMK95] B. Adelberg, H. Garcia-Monila, and B. Kao. Applying update streams in a soft real-time database system. In *ACM SIGMOD'95*, 1995.

[DW93] L. B. C. DiPippo and V. F. Wolfe. Object-based semantic real-time concurrency control. In *IEEE Real-Time Systems Symposium*, Dec. 1993.

[Lin89] Kwei-Jay Lin. Consistency issues in real-time database systems. In *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences*, pages 654–661, January 1989.

[PL95] C.S. Peng and K.J. Lin. A semantic-based concurrency control protocol for real-time transactions. *Submitted for publication*, 1995.

[LSL90] R. Rajkumar, L. Sha and J. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.