

RELIABLE TRANSACTION PROCESSING FOR REAL-TIME DISTRIBUTED DATABASE SYSTEMS

Yong I. Yoon and Song C. Moon

Computer Science Department, Korea Advanced Institute of Science and Technology,
Cheongryang P.O. Box 150, Seoul, 130-650 KOREA. E-mail: {yiyoon,moon}@csd.kaist.ac.kr

When real-time applications require distributed transaction processing, both correct completion and timeliness should be satisfied. All previous commit protocols, however, fail to provide the timeliness for real-time processing. That is, methods for the timely completion do not always guarantee the correct completion or vice versa. In this paper, we propose a novel commit protocol, called an *integrated commit protocol (ICP)*, in which the correct completion is satisfied with timely completion for distributed real-time transactions. The basic idea of ICP is that commit procedures for correct completion use the results of remote transactions which are timely executed. The timely execution is provided by the dynamic multilevel scheduling policy and the multi-version timestamp ordering scheme.

1. INTRODUCTION

Applications including telecommunication systems and military systems stimulated database research towards real-time database systems [6]. They require a *timely completion* such that data manipulation operations should be executed within a specified deadline. The real-time database systems need to operate in a distributed fashion to support inherent distributed nature of the applications. A distributed program requires a group of cooperating transactions distributed in the whole system. It is essential for the system to complete correctly so that the group of transactions behave consistently in the presence of failures. A key property of the correct completion is *failure atomicity* that means either a program has the intended results, or it has no results at all [1].

The distributed transaction processing requires lots of message exchanges over the networks, which can hinder the timely completion. Because the speed of communication network limits the performance of distributed systems, it is necessary to reduce the number of intersite communications for real-time processing. Other important aspects for the timeliness are how to specify the time constraints and how to execute the transactions within the time constraints. In addition, it must guarantee the correct completion of the transactions. To summarize, we must provide a new commit protocol in which correct completion and timely completion are both satisfied in a distributed real-time transaction processing environment.

2. RELATED WORK AND MOTIVATION

2.1 Related Work

The atomic commitment protocol (ACP) is proposed to guarantee failure atomicity for correct completion of the distributed transactions. A well-known algorithm

for ACP is the two-phase commit protocol (2PCP) [5], and its variations are proposed in [3,4,8]. Mohan [8] and Ceri [1] improved the performance of 2PCP by reducing the number of messages exchanged in non-real-time systems. Chu [3] and Davidson [8] suggested other variations of the commit protocol for real-time applications. We describe these latter two commit protocols and their problems because only they are concerned with the real-time processing.

Resilient Commit Protocol

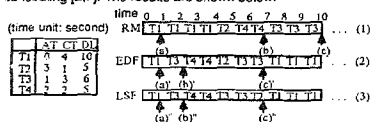
Chu suggested a resilient commit protocol under the assumption that both reliable networks and periodic exchange of "I am alive" messages among sites are used. The assumption of reliable networks eliminates acknowledgement messages because the messages are managed by the network subsystems. Without checking the agreement of each participant, this protocol satisfies the failure atomicity. It enables the fast computing of the protocol due to the elimination of the agreement phase in 2PCP. This commit protocol, however, did not consider the timeliness which must be one of the primary goals of real-time systems. To support the timeliness, the time constraints must be reflected in the commit protocol.

Timed Atomic Commitment Protocol

The timed atomic commitment protocol provides time specifications to check deadlines in each phase of 2PCP. The time constraints are defined in the transaction itself. For the timely completion, Davidson assumed both a sufficiently long deadline and a fair scheduling policy. Under these assumptions, the timed atomic commitment protocol satisfied timeliness for real-time applications. Since this protocol was based on basic 2PCP [5], there are two phases of message exchange for commit processing. But, to support the fast computing, it is required to reduce the number of message exchanges to commit quickly. Also, scheduling policies for real-time processing may produce different orders of transaction execution, affected by time constraints given, as shown in Example 1. Ther-

efore, a proper scheduling policy must be suggested to satisfy the timeliness of commit processing.

Example 1: Consider the following set of transactions with arrival time AT, computation time CT, and deadline DL in a single site. These transactions are applied to three scheduling policies for real-time processing: rate monotonic (RM) scheduling, least slack time first (LSF) scheduling, and earliest deadline first (EDF) scheduling [2,7]. The results are shown below.



RM schedules transactions with a shorter computation time first. It produces a schedule $T_1 \rightarrow T_2 \rightarrow T_4 \rightarrow T_3$ as shown in (1). At (a), T_3 cannot preempt T_1 because T_1 's CT, which is 3, is equal to T_3 's CT. So, T_1 is continuously dispatched. When T_1 is completed at time 4, T_2 is dispatched because T_2 's CT is the shortest of T_2 , T_3 , and T_4 . At (b) and (c), T_4 and T_3 fail because of its missed DL, respectively. EDF is based on the deadline of transactions and produces a schedule $T_4 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$ as shown in (2). At (a'), T_3 preempts T_1 because T_3 's DL is shorter than T_1 's DL. At (b'), T_4 preempts T_3 because the deadline of T_4 is shorter than that of T_3 . At (c'), T_3 fails because of its missed deadline. Lastly, LSF is based on slack time S , defined as $S = DL - (\text{current time} + CT - \text{executed time})$. It schedules transactions with a least slack time first, and thus produces a schedule $T_4 \rightarrow T_3 \rightarrow T_2 \rightarrow T_1$ as shown in (3). At (a''), T_3 preempts T_1 because T_3 's S , which is 2, is shorter than T_1 's S . At (b''), T_4 preempts T_3 because T_4 's S , which is 1, is the shortest slack time. At (c''), T_2 fails because of its missed deadline. Note this example shows that the failed transactions turn out to be different for these scheduling policies. It implies that the timely completion of transactions can be changed by the scheduling policy used. □

2.2 Motivation

We have shown that the above two commit protocols support the correct completion but cannot guarantee the timeliness. To support the distributed real-time transaction processing, it is necessary to consider the ways to specify time constraints, the timely execution of the transactions, and fast computing of the commit protocol. We propose a new commit protocol that includes the time specification method, the timely execution policy, and fast computing capability. The goal of the proposed protocol is to satisfy the correct completion and timely completion for the distributed real-time transactions.

3. A MODEL FOR DISTRIBUTED PROCESSING

A model for the distributed real-time transaction processing consists of a calling transaction, called a coordinator, and its cooperating transactions, called participants. A coordinator T_1 dynamically creates a set of participants, i.e., $\{T_{11}, T_{12}, \dots, T_{1n}\}$, and manages their correct completion. T_1 keeps information on the participants to guarantee the correct completion. Each participant returns its execution result which will be used to decide a commit/abort action by its coordinator. Example 2 shows that a distributed transaction requires its cooperating transactions.

Example 2: Consider an electronic switching system (ESS) such as TDX-10, which is considered as a hard real-time system, consists of several sites, each composed of OMP, MMP, and several ASPs. In this ESS, CIs are used as criteria to generate CDs by CDGT when a call happens in an ASP. For example, CI is used to determine whether a call is local or long-distance and CD is computed on basis of CI. CIs are replicated in all ASPs and are periodically changed. To change CIs, CMT invokes CHT and CHT invokes several CICTs. There is a transaction hierarchy among these transactions as shown in Figure 1. □

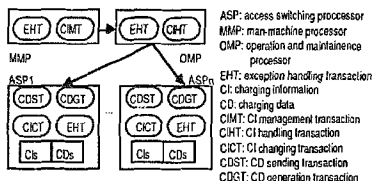


Figure 1

4. INTEGRATED COMMIT PROTOCOL

The basic idea of new commit protocol is that the commit procedures for correct completion use the results of participants which are timely executed. We call the new protocol an *integrated commit protocol* (ICP). The ICP is composed of three parts: *language constructs* to specify the real-time requirements, *scheduling policy* to support the timely execution, and *commit procedures* to satisfy the failure atomicity. In this section, we describe these three parts.

4.1 Language Constructs

The distributed real-time transaction processing requires some time constraints. Transaction deadline (TD) is required to ensure a transaction completion. Remote execution deadline (RED) is required to ensure the completion of a remote request. Theorem 1 shows a relationship between the time constraints. [Theorem 1] A participant's deadline TD_i is less than the coordinator's remote execution deadline RED_i. [Proof] Obviously, the result of participants may be used by the coordinator after RED_i as shown in Figure 2. □

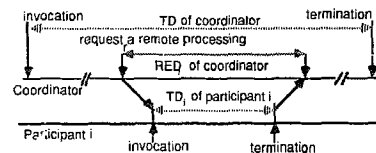


Figure 2

The constraints must be reflected to the distributed real-time processing. To do this, we need language constructs for the specification of the time constraints, the invocation of remote transactions, and handling

the missed time constraints. Remote procedure call (RPC) supports the invocation of remote transaction, execution of the remote transaction, and handling of its results [9]. But, the original RPC scheme does not include the timing requirements. Thus, we extend the RPC scheme to support the real-time processing. We focus on the issues to extend the RPC scheme.

Specifying the time constraints

In Figure 3, TD is explicitly specified by using the input arguments in the transaction definition. RED is also explicitly specified in the procedure call syntax.

'Send and no wait' and keeping the return messages
To support asynchronous remote processing, each coordinator is not required to wait until its return message is arrived. To do so, the 'send-and-no wait' call syntax is proposed. Each coordinator may keep the return messages until these are used to guarantee the correctness of the participants. To keep the return messages, a vectorized array is defined in the call syntax of the coordinator as shown in Figure 3.

Handling the exception and failure recovery

To handle exceptions for reliable processing, the exception handling routines are defined in the application programs. The routines contain the compensative operations to support the forward recovery and to resolve the inconsistency caused by exceptions or failures. A transaction can have several exception handling routines and compensative routines for timeout exception, error recovery, and undoing the updates. The routines are performed as urgent transactions to handle the exceptions or failures immediately.

Transaction: T_i (arguments)

- Declaration of **exception handling routines**;
- Specification of the **transaction deadline** TD_i ;
- Initialization of vector V_i ;
- Specification of **RED**;
- $V = \text{RPC}(\text{RED and other arguments})$;
- Check the timeliness of the remote transaction;

END T_i ;

Figure 3

4.2 Timely Execution Method

There are three types of transaction: internal transactions for local processing, external transactions for remote processing, and urgent transactions for exception handling. These transactions must be executed within their time constraints.

Scheduling Policy

In our model, transactions keep their time information, i.e., TD and RED, and their urgency information defined at the time of their creation. We suggest a dynamic multi-level priority scheduling policy based on the urgency and on the priorities assigned by the EDF scheduling policy. The reason is that we are able to know the deadline and the arrival time for each transaction, and that the EDF scheduling policy produces feasible schedules for real-time processing when both the deadline and the arrival time are given [2,7]. When an urgent transaction arrives, it is immediately executed. If two transactions have the

same level of urgency, then the execution order is decided by their priorities. The multi-level scheduling policy allows the preemption between transactions as shown in Example 3. The preemption induces the restart for the preempted transaction to resolve the inconsistency for shared data.

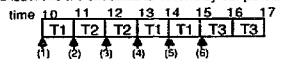
Conflict Resolving Method

To prevent the restart of preempted transaction, we adopt multi-version concurrency control protocol with the multi-level scheduling policy. The reason is that the multi-version timestamp ordering scheme is able to prevent the restarts of preempted transactions as shown in Example 3. Also, it is desirable for aggressive recovery using the compensative routines. For example, in Example 3, if the coordinator for T_2 decides an abort action, T_2 undo the updated version X^* by removing X^* .

Example 3: Consider Example 2. Suppose that, in $ASPI$, there are three kinds of transaction: T_1 for CDGT, T_2 for CICT, and T_3 for CDST. Additionally, suppose that there are two kinds of data: X for CI and Y for CD. Y is generated using X by CDGT when a call happens. CDST is periodically invoked to send Y to OMP. Each transaction has operations and time information as follows:

	AT	CT	DL
T_1	10	3	15
T_2	11	2	13
T_3	12	2	17

These transactions are executed as follows by our policies.



(1) through (6) denote the flow of scheduling. X^* and Y^* denote the versions of X and Y at time 10, respectively. At (1), T_1 read X to compute a new Y . At (2), T_2 preempts T_1 because T_1 's DL is longer than T_2 's DL. At (3), Read/Write conflict occurs between T_1 and T_2 because T_1 is preempted during read X^* and T_2 wants to write X^* . Then, T_2 creates a new version X^* for X . At (4), T_2 unilaterally commits because T_2 is executed normally. At (5), T_1 writes Y^* to change the current CI. At (6), T_3 reads Y^* that is changed by T_1 . □

4.3 Commit Procedures

When transactions are timely executed, their correct completion must be guaranteed despite of failures. The commit procedures of *integrated commit protocol* supports the failure atomicity to satisfy the correct completion with the timeliness. Figure 4 shows the basic flow of ICP.

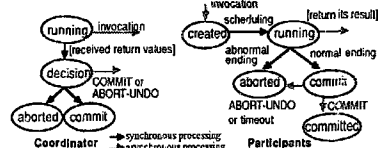


Figure 4

The return messages from participants are stored in the vectorized array which consists of three lists: all participants list P_n , committed participants list P_c , and aborted participants list P_a . P_n keeps a set of

participants invoked by a coordinator. P_c and P_a keep the result values from participants.

The decision for correct completion is made through the results of participants kept in P_c and P_a . We define a theorem for the decision. When P_c is the same as P_n , the coordinator decides *commit* action. Otherwise, the coordinator decides *abort* action by Theorem 2. When the coordinator decides an action for atomicity, the coordinator sends a message for the decided action to participants. For example, if the *abort* action, the coordinator sends ABORT-UNDO message to P_c and sends ABORT message to $P_n - (P_a + P_c)$. If the *commit* action, the coordinator sends COMMIT message to P_n . Algorithm 1 show the flow.

[Theorem 2] If P_c is not equal to P_n , then the coordinator always decides the Abort action.

[Proof] It is obvious that each participant must be executed before the coordinator decides a *commit/abort* action by using the result of participants, because of Theorem 1. If P_c is not equal to P_n , then TD of some participant exceeds RED of coordinator. Then, we must abort the participants for the correct completion with the timely completion. \square

Algorithm 1: Decision Procedures

```

IF ( $P_c \neq P_n$ )
  THEN decide Abort action ;
    SEND ABORT TO  $P_n - (P_a + P_c)$  and
    SEND ABORT-UNDO TO  $P_c$  ;
  ELSE decide Commit action ;
    SEND COMMIT TO  $P_n$  ;
FI ;

```

In the distributed transaction processing, there are several failures: transaction failures, site failures, and lost messages. We show recovery procedures to survive these failures for reliable processing.

Transaction failures

For the transaction processing, each transaction updates some data objects temporarily. If software fault occurs, the system indicates an *abort* signal to the faulty transaction. The transactions take *abort* action to keep the consistency for the temporarily updated data. We solve this case by using the undo routine that is defined in the faulty transaction.

Site failures

If the coordinator fails, participants issue timeout exception because no message from the coordinator is received. Then the participants execute the compensative routines which will decide the *abort* action. The failed coordinator is recovered by Algorithm 2. First, the recovery manager checks the deadline of the failed transaction to guarantee the timeliness. If missed, the recovery manager decides the *abort* action and send ABORT message to P_n . If not missed, the recovery manager restores to normal state and continuously executes its normal functions.

When a participant fails, the coordinator decides the *abort* action through Theorem 2. But, to keep the failure atomicity for alive participants, the failed participant is excluded from its participants list P_n until the failed participant is recovered. The failed participant is recovered by restoring all the current coordinator site's status and sends an alive message

to its coordinator. The coordinator includes the recovered transaction into its participant list P_n .

Algorithm 2: Recovery Procedures

```

IF DeadlineOfTransaction is not expired;
  THEN restore to normal state;
    execute the normal function;
  ELSE decide Abort action;
    SEND ABORT TO  $P_n$ ;
    terminate the transaction ;
FI ;

```

Lost Messages

When the return messages from participants are lost, the coordinator regards that the participants are failed. The coordinator decides *abort* action and sends an ABORT message to each participant. When the action message is lost, the participants also regard that the coordinator is failed and perform the compensative routines to decide the abort action.

5. CONCLUSION AND FURTHER STUDY

We have proposed a new commit protocol, called ICP, for the reliable distributed real-time transaction processing. ICP used the timely executed results in the commit procedures for the correct completion and the timely completion. Furthermore, ICP obtains fast computing through using of the result values of participating transactions because the use of result values reduces the number of message exchanges for commit processing in 2PCP.

We plan to implement ICP in the real world environments like ESS. Then, we will show our protocol satisfies both timeliness and correct completion.

REFERENCES

- [1] Ceri, S. and Pelagatti, G., Distributed Database: Principles and Systems, McGraw-Hill, 1984.
- [2] Abbott, R. and Garcia-Molina, H., "Scheduling Real-Time Transactions: Performance Evaluation," Proceedings of VLDB, 1988, pp. 1-12.
- [3] Chu, W., "Resilient Commit Protocol," IEEE 5th Real-Time Systems Symposium, 1985, pp. 25-29.
- [4] Davidson, S. and Lee, I., "Timed Atomic Commitment," MS-CIS-88-80, Dept. of Computer and Information Science, Univ. of Pennsylvania, Oct. 1989.
- [5] Gray, J., "Notes on Database Operating Systems," IBM Research Report, RJ 2188, February 1978.
- [6] Singhal, M., "Issues and Applications to Design of Real-Time Database Systems," ACM SIGMOD Records, Vol. 17, No. 1, March 1988, pp. 19-33.
- [7] Huang, J., et. al., "Experimental Evaluation of Real-Time Transaction Processing," IEEE 10th Real-Time Systems Symposium, December 1989, pp. 144-153.
- [8] Mohan, C. and Lindsay, B., "Efficient Commit Protocols for the Tree of Process Model of Distributed Transactions," IBM Research Report, RJ 3881, June 1983.
- [9] Nelson, B.J., "Remote Procedure Calls," CMU-CS-81-119, Dept. of Computer Science, Carnegie-Mellon University, 1981.