3 G－4

Group Protocol for Transaction-Oriented Applications *

Tomoya Enokido, Takayuki Tachikawa, and Makoto Takizawa †
Tokyo Denki University ‡
e-mail{eno, tachi, taki}@takilab.k.dendai.ac.jp

1 Introduction

In group communication protocols, larger computation and communication overheads are consumed to causally order all the messages transmitted in the network. Transactions in clients manipulate objects in servers by sending read and write requests to the servers. In this paper, we define significant messages by using the conflicting relation among the transactions. We newly propose an object vector to causally order only the significant messages. The scheme of the object vector is invariant in the change of the group membership. We also show a TBCO (transaction-based causally ordered) protocol which adopts the object vector, by which the number of messages to be causally ordered are reduced.

We present a system model in section 2. In section 3, we define significant messages. In section 4, we present the *object vector* and the TBCO protocol. In section 5, we evaluate the TBCO protocol.

2 System Model

A system is composed of processors p_1, ..., p_n ($n \geq 1$) interconnected by the communication network. Each object o_i is stored in one processor. In the network, messages may be lost and may be delivered out of order. On receipt of a request message m with an operation op_i, o_i computes op_i. We assume that each o_i is replicated. Each replica is allocated in a processor. Let o_i^t denote a replica of o_i in p_t. A transaction T_i in one *client* processor p_s issues *read* and *write* requests to the *server* processors to manipulate replicas of an objects o_i. p_s sends a *read* request to one processor p_t which has a replica of o_i. p_t sends back the response with the data derived. p_s sends *write* requests to all the replicas of o_i. Here, $op_1 \rightarrow_{T_i} op_2$ iff an operation op_1 precedes op_2 in T_i. T_i is an atomic sequence of *read* and *write* operations[1]. Let T be a set $\{ T_1, \ldots, T_m \}$ of transactions in the system. Let $op_i^t(x)$ denote an operation op of T_i on a replica x in p_t. op is $r(read)$ or $w(write)$. A subsequence of operations of T_i on the replicas in a processor p_t is named a *subtransaction* T_{it}. $op_1 \rightarrow_{T_{it}} op_2$ iff op_1 precedes op_2 in T_{it}. The interleaved computation of subtransactions T_{1t}, ..., T_{mt} in p_t is a *local history* H_t of p_t. op_1 *precedes* op_2 in p_t ($op_1 \rightarrow_{H_t} op_2$) iff op_1 is computed before op_2 in p_t. A global history H of T is a collection of the local histories H_1, ..., H_n. op_1 precedes op_2 in H ($op_1 \rightarrow_H op_2$) iff $op_1 \rightarrow_{H_t} op_2$, $op_1 \rightarrow_{T_i} op_2$, or $op_1 \rightarrow_H op_3 \rightarrow_H op_2$ for some p_t, T_i and op_3. H has to be *serializable*[1] to keep the replicas consistent. In order to make H serializable, the conflicting operations on the same object have to be computed in the same order as some serial history.

3 Significant Messages

A processor p_t may not receive messages or may receive messages out of order. If p_t loses a message m sent by p_u, p_u is required to resend m to p_t. After p_t receives m, p_t has to *wait* for all messages causally preceding m.

[**Definition**] A message m is *insignificant* iff

(1) if $m = r_t^i(x)$, there is some read $r_t^j(x)$ in RQ_t such that $r_t^j(x)$ precedes $r_t^i(x)$ and there is no write $w_t^k(x)$ between $r_t^j(x)$ and $r_t^i(x)$ in RQ_t.

(2) if $m = w_t^i(x)$, there is some write $w_t^j(x)$ in RQ_t such that $w_t^i(x)$ precedes $w_t^j(x)$ and there is no read $r_t^k(x)$ between $w_t^i(x)$ and $w_t^j(x)$ in RQ_t. □

Insignificant messages can be omitted from RQ_u. Significant messages which are not insignificant have to be delivered in causal order.

4 TBCO Protocol

4.1 Object vector

A group is considered to be composed of subtransactions and servers. The membership of the group changes each time transactions are initiated and terminated. If the vector clock is used, the group has to be frequently resynchronized. In this paper, we propose an *object vector* to causally order the significant messages sent by the transactions. Each transaction T_i is given a unique identifier $t(T_i)$ satisfying the following properties :

(1) If T_i starts before T_j in a processor, $t(T_i) < t(T_j)$.

(2) If a processor p_t initiates T_i after receiving a message from T_j, $t(T_i) > t(T_j)$.

The transaction identifier is given by the linear clock[2]. p_t manipulates a variable *tid* showing the linear clock whose initial value is 0 as follows : (1) $tid := tid + 1$ if a transaction is initiated in p_t. (2) On receipt of a message from T_j, $tid := \max(tid, tid(T_j))$. When T_i is initiated in p_t, $t(T_i)$ is given a concatenation of *tid* and the processor number $pno(p_t)$ of p_t. For every pair of T_i and T_j initiated at p_t and p_u, $tid(T_i) > tid(T_j)$ if (1) $t(T_i) > t(T_j)$ or (2) $t(T_i) = t(T_j)$ and $pno(p_t) > pno(p_u)$. Each event e is given an event number $no(e)$ as follows :

(1) If e_1 is the initial event of T_i, $no(e_1) = 0$.

(2) If e_2 is *write* and $e_1 \rightarrow e_2$ in T_i, $no(e_1) < no(e_2)$.

(3) If e_2 is *read* and there is no *write* event e_3 such that $e_1 \rightarrow_{T_i} e_3 \rightarrow_{T_i} e_2$, $no(e_1) = no(e_2)$.

Each event e in T_i is given a global event number $tno(e)$ as the concatenation of $t(T_i)$ and $no(e)$. Every replica o_a^j of o_a has the same version number $v(o_a^j) = v(o_a)$. $v(o_a^j)$ is updated to be $tno(w)$ each time $w(o_a^j)$ is computed. Here, if $tno(op) > v(o_a^j)$, op can be computed. Otherwise, op aborts because op is obsolete.

---

*トランザクションを考慮したグループ通信プロトコル

†榎戸 智也 立川 敬行 滝沢 誠
‡東京電機大学

4.2 Message transmission and receipt

If $m.op = read$, $m.dst$ denotes one processor which has a replica o_a^i of o_a. If $m.op = write$, $m.dst$ shows all the processors which have the replicas of o_a. If m' is a response message of m, $m'.tno = m.tno$. A processor p_s constructs a request message m with op_h from T_h as follows : (1) $m.tno = \langle m.t, m.no \rangle := \langle t(T_h), no(op_h) \rangle$ (2) $m.op := op_h$ (3) $m.o := o_a$ (4) $m.src := p_s$ (5) $m.dst :=$ set of destination processors (6) $m.V_a := V_a$ ($a = 1, \ldots, u$) (7) $m.d :=$ data (8) $sq_t := sq_t + 1$ for every p_t in $m.dst$ (9) $m.sq_j := sq_j$ ($j = 1, \ldots, n$). Each time p_s sends a message to p_t, sq_t is incremented by one. Then, p_s sends m to every destination p_t in $m.dst$. p_t can detect a gap between messages received from p_s by checking the sequence number sq_t, i.e. messages lost or unexpectedly delayed. p_t manipulates variables $rsq_1, \ldots, rsq_n$ to receive messages. On receipt of a message m from p_s, there is no gap if $m.sq_t = rsq_s$. If $m.sq_t > rsq_s$, there is a gap m' where $m.sq_t > m'.sq_t \geq rsq_s$. m is *correctly* received by p_t if p_t receives every message m' where $m'.sq_t < m.sq_t$. p_t manipulates the object vector V as $V_a = \max(V_a, m.V_a)$ for $a = 1, \ldots u$. If $m.op$ completes to manipulate a replica o_a^t, p_t sends back a response m' of m to p_s. If $m.op = read$, m' carries data derived from o_a^t in $m'.d$ and $m'.V_a$ the version number $v(o_a)$ of o_a^t. T_s has a vector $V = \langle V_1, \ldots, V_u \rangle$ where each variable V_a is initially 0. On receipt of the response m' from p_t, T_s manipulates V, i.e. $V_a := m'.V_a$ ($= v(o_a^j)$) if $V_a < m'.V_a$. Otherwise, T_s aborts.

4.3 Message delivery

Every pair of messages m_1 and m_2 in RQ_u is ordered according to the following ordering rule.

[Ordering rule] $m_1 \Rightarrow m_2$ if one of the following conditions holds :

(1) $m_1.V < m_2.V$.
(2) $m_1.V = m_2.V$ and $m_1.t < m_2.t$ and $m_1.op$ conflicts with $m_2.op$.
(3) $m_1.V$ and $m_2.V$ are not compared and $m_1.o = m_2.o (= o_a)$ and
  (3.1) $m_1.V_a < m_2.V_a$, or
  (3.2) $m_1.V_a = m_2.V_a$ and $m_1.t < m_2.t$ and $m_1.op$ *conflicts* with $m_2.op$. □

Messages m_1 and m_2 are *concurrent* ($m_1 \parallel m_2$) if the ordering rule is not satisfied. Concurrent messages are stored in RQ_u in the receipt order.

[Theorem] m_1 causally precedes m_2 ($m_1 \rightarrow m_2$) if m_1 precedes m_2 in the ordering rule ($m_1 \Rightarrow m_2$). □

We discuss what messages in RQ_t can be delivered.

[Delivery procedure] While each top message in RQ_t is *stable* and *ready* to deliver, m is dequeued from RQ_t and m is delivered if m is significant, otherwise m is neglected. □

[Definition] m is *stable* in RQ_t iff

(1) there is a message m_1 from p_u in RQ_t where $m_1.sq_t = m.sq_t + 1$ and
(2) p_t correctly receives a message m_1 in RQ_t from every p_u where $m \rightarrow m_1$. □

[Definition] A top message m in RQ_t is *ready* in p_t if p_t computes no operation conflicting with $m.op$ in a replica $m.o$. □

If some p_u sends no message to p_t, the messages in RQ_t cannot be stable. In order to resolve this problem, each p_u sends a message without data to p_t if p_u had not sent any message to p_t for some predetermined time units.

5 Evaluation

The TBCO protocol is realized in threads of a Super Server 6400 with 10 Ultra Sparcs. Each processor p_t is bound to one Ultra Sparc ($t = 1, \ldots, n$). TCP is used to exchange messages among the processors. There are three objects which are fully replicated in all the processors. Each processor randomly initiates totally twenty transactions where each of which issues ten arbitrary kinds of operations on arbitrary objects.

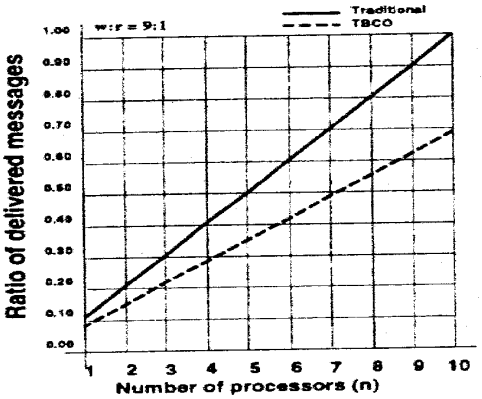

Figure 1: Ratio of messages.

In Figures 1, the vertical axis indicates the ratio of the number of messages delivered to the number of messages transmitted in case of ten processors ($n = 10$) in the traditional message-based protocol. The dotted line shows TBCO and the solid line indicates the traditional message-based protocol. The Figure 1 shows case that 90% of the operations issued by each transaction are writes. The figure 1 show the number of messages delivered can be reduced by the TBCO protocol. About 30% of messages transmitted in the network are reduced for the write ratio 90%.

6 Concluding Remarks

This paper has discussed what messages have to be causally ordered in replicated objects with read and write from the application point of view. We have proposed the novel *object vector* for causally preceding messages based on the transaction concept. In the TBCO protocol, only the messages to be meaningfully preceded for the applications can be causally ordered. We have also discussed a way for omitting messages which are not significant for the applications. We have shown the TBCO protocol implies fewer operations computed than the protocols which causally order all the messages transmitted in the network.

References

[1] Bernstein, P. A., Hadzilacos, V. and Goodman, N., "Concurrency Control and Recovery in Database Systems," *Addison-Wesley*, pp.25–45, 1987.

[2] Lamport, L., "Time, Clocks, and the Ordering of Events in a Distributed System," *Comm. ACM*, Vol.21, No.7, pp.558–565, 1978.

[3] Raynal, M. and Ahamad, M., "Exploiting Write Semantics in Implementing Partially Replicated Causal Objects," IRISA Research Report, PI-1080, 1997.