Group Protocol for Distributed Object-Based Systems*

YOUHEI TIMURA, KATSUYA TANAKA AND MAKOTO TAKIZAWA
Department of Computers and Systems Engineering
Tokyo Denki University
Hiki-gun, Saitama 350-0394, Japan
E-mail: {timura, katsu, taki}@takilab.k.dendai.ac.jp

Distributed applications are realized by the cooperation of a group of multiple objects. In the group cooperation, objects send and receive messages in various ways. A message is multicast to objects in the group. In addition, multiple types of messages are sent in parallel to multiple destinations. Then, an object waits for messages from all objects and some of the source objects, i.e., in conjunctive and disjunctive ways. In this paper, we define a novel precedent relation on request and response messages exchanged among objects in the presence of the transmission and receipt methods. We present a communication protocol for supporting a group of processes with the ordered delivery of messages in the precedent relation. By using the protocol, it is easy to realize distributed object-based applications like database replications.

Keywords: distributed systems, object-based systems, causality of messages, ordered delivery, group communication protocol

1. INTRODUCTION

In a distributed application like a teleconference, a *group* of multiple processes cooperate to achieve some set of objectives. A process sends a messages to multiple processes and receives messages for multiple processes in the *group*. Many papers [5, 6, 11, 13, 14] discuss how to support a group of multiple processes with the causally / totally ordered delivery of messages transmitted at a network level. Group protocol is using the vector clock [6, 10] implies $O(n^2)$ computational and communication overheads for n processes in the group. The overheads can be reduced if the messages required to be ordered by the applications are causally and atomically delivered. In this paper, we discuss what messages transmitted in object-base applications to be ordered in the network.

An application is realized by a collection of processes, each of which manipulates data like files and exchanges messages with other processes, i.e., a *process-based* application. On the other hand, in object-based applications like CORBA [12], data and methods are encapsulated in an object, and methods are invoked by a message-passing mechanism. A transaction in an application sends a *request message* with a method to an object. The method is performed on the object. A *response message* is sent back to the sender of the request. In addition, the method may invoke other methods, i.e., *nested invocation*. Here, each of the request and response messages is sent to one destination.

Received September 22, 2001; accepted April 15, 2002.
Communicated by Jang-Ping Sheu, Makoto Takizawa and Myongsoon Park.
* The research is partially supported by Research Institute for Technology, Tokyo Denki University.

A transaction may simultaneously invoke multiple methods on objects; this is a *parallel* invocation. Suppose there are multiple replicas of a database object. A transaction invokes the same SQL [1] method on each replica. Here, a request message is sent to multiple objects, i.e., *multicast*. The paper discuss how to use group communication protocols in order to realize the replication of database and processes. In another example, a transaction invokes a *book-car* method on a *rent-a-car* object and *book-room* on a *hotel* object. The transaction can invoke both the methods in parallel. Here, different types of request messages are sent to multiple objects simultaneously. This is referred to as *parallel-cast* (paracast). At the network level, a pair of request messages *book-car* and *book-room* may be transmitted edserially. There is no precedent relation between the messages from the application point of view. Objects wait for multiple responses after multiple methods are invoked in parallel. There are conjunctive and disjunctive ways to receive multiple messages. In the conjunctive receipt, the object waits for all the messages. Hence, even if the object sends a message while receiving these messages, there is no causally precedent relation between the messages. In the disjunctive receipt, the object waits for only a message which arrives at the computer earlier than the others and is not required to receive all the other messages. In this paper, we discuss a new type of causally precedent (*significantly precedent*) relation among messages in a network system, where messages are unicast, multicast and paracast, and received by single-message and multi-message conjunctive and disjunctive receipts at application level. We also discuss a protocol which supports the significantly precedent delivery in the object-based system.

In section 2, we present a system model. In section 3, we discuss how messages are exchanged among objects. In sections 4 and 5, we discuss the *significantly precedent* relation of messages and a protocol. In section 6, we show how many messages are ordered.

2. SYSTEM MODEL

Objects are encapsulations of data and methods for manipulating the data. A transaction invokes a method on an object by sending a request to the object. A *thread* for the method is created and is performed on the object. Here, other methods may be invoked by the method, i.e., nested invocation. Then, on completion of the method, the response is sent back to the transaction.

Objects are distributed in computers interconnected with reliable networks. A *computer* does not necessarily mean a physical computer. A database server is an example of a computer where objects are tables and records. Each computer p_t has a *transaction* object *tran* which supports an *init-tran* method. An application initiates a transaction by invoking *init-tran* on the *tran* object. Transactions are realized by threads of the *init-tran* method on the *tran* object. There is another specific type of object, a *communication* object *com* which supports objects in the computer p_t with communication methods. The *com* object supports communication methods for sending and receiving messages. In order to send and receive messages, methods on the *com* object are invoked. The *com* object forwards the messages to *com* objects which support the destination objects in the network. The destination objects cooperate to deliver messages to the objects in G. The cooperation of the *com* objects is coordinated by the group protocol.

In the traditional group protocols [6], a group is composed of processes where messages are causally / totally delivered independently of what kinds of data are carried by the messages. In this paper, a group is composed of objects and transaction *tran* objects. Only methods on objects in G are assumed to be invoked in each transaction.

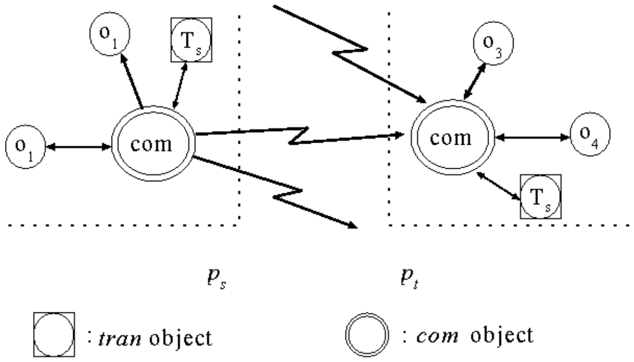

Fig. 1. *com* object.

There are ways to invoke multiple methods. In the *serial* invocation, at most one method is invoked at a time. On the other hand, multiple methods can be simultaneously invoked in the *parallel* invocation. Here, request messages are sent to multiple objects. For example, an SQL request is issued to multiple replicas of a database server. Suppose a pair of methods t_1 and t_2 are synchronously invoked by a transaction. The transaction waits for responses from the objects. There are *conjunctive* and *disjunctive* ways to receive the responses. In the conjunctive receipt, the transaction blocks until both of the responses are received. In the disjunctive receipt, the transaction blocks until at least one response is received. The transaction does not receive the other response. In the conjunctive receipt, the requests are required to be atomically delivered to the transaction. On the other hand, at least one request can be required to be delivered in the disjunctive receipt.

According to traditional theories [5], a method t_1 *conflicts* with another method t_2 on an object if the result obtained by performing methods t_1 and t_2 on the object depends on the order of computation of t_1 and t_2. Otherwise, t_1 is *compatible* with t_2. For example, *deposit* and *withdraw* are compatible on a *Bank* object. By using the locking mechanism [5], a pair of conflicting methods t_1 and t_2 are serially performed. In this paper, we assume the conflicting relation is symmetric and transitive. That is, if t_1 conflicts with t_2, then t_2 conflicts with t_1. In addition, if t_2 conflicts with t_3, then t_1 conflicts with t_3.

3. INTER-OBJECT COMMUNICATION

3.1 Transmission

A communication object *com* in each computer supports objects with the following communication methods for transmitting messages:

Transmission methods

1. **ucast**(m, o_t).
2. **mcast**(m: $\langle o_1, \ldots, o_h \rangle$).
3. **pcast**(m_1: $\langle o_{11}, \ldots, o_{1l_1} \rangle$; …;
   m_k: $\langle o_{k1}, \ldots, o_{kl_k} \rangle$).

In the first method **ucast**(m, o_t) a message m is unicast to an object o_t. The second method **mcast**(m, $\langle o_1, \ldots, o_h \rangle$) shows that a message m is multicast to multiple objects $o_1, \ldots, o_h$ ($h \geq 1$). For example, a message m is sent to a pair of objects o_1 and o_2 by **mcast**(m, $\langle o_1; o_2 \rangle$) [Fig. 2(1)]. The last method, **pcast**(m_1: $\langle o_{11}, \ldots, o_{1l_1} \rangle$; …; m_k: $\langle o_{k1}, \ldots, o_{kl_k} \rangle$) means that messages $m_1, \ldots, m_k$ are paracast, i.e., each message m_i is multicast to objects $o_{i1} \ldots, o_{il_i}$ ($l_i \geq 1$) ($i = 1, \ldots, k$). A pair of messages m_1 and m_2 are sent to objects o_1 and o_2, respectively, by invoking **pcast**(m_1:$\langle o_1 \rangle$, m_2:$\langle o_2 \rangle$) [Fig. 2(2)].

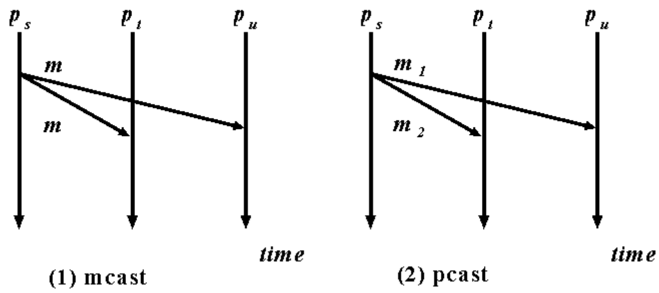

Fig. 2. **mcast** and **pcast**.

3.2 Receipt

Suppose a thread t performed on an object in parallel invokes multiple methods. The thread t waits for response messages from multiple objects after sending the requests to the objects. There are multiple ways to receive messages, *single-message* and *multi-message* receipts, where an invoker thread waits for only one message and multiple messages, respectively. A *com* object supports objects in a computer with the following types of primitive methods for receiving messages:

Receipt methods

1. **srec**(o_1).
2. **crec**($o_1, \ldots, o_k$) ($k \geq 1$).
3. **drec**($o_1, \ldots, o_k$) ($k \geq 1$).

By invoking **srec**(o_1) on the *com* object, one message is received from an object o_1, i.e., single-message receipt. If a thread t invokes **crec**, t blocks until messages from all the objects $o_1, \ldots, o_k$ are received. For example, **crec**(o_1, o_2) is invoked in Fig. 3(1). Here, both m_1 and m_2 are received. If **drec** is invoked, the thread t blocks just until one message is received from one object in $o_1, \ldots, o_n$. If the thread t receives one response message, say from o_1, t does not receive the other response messages. In Fig. 3(2), **drec**(o_1, o_2) is invoked. On receipt of m_2, **drec** completes, i.e., m_2 is received but m_1 is not received.

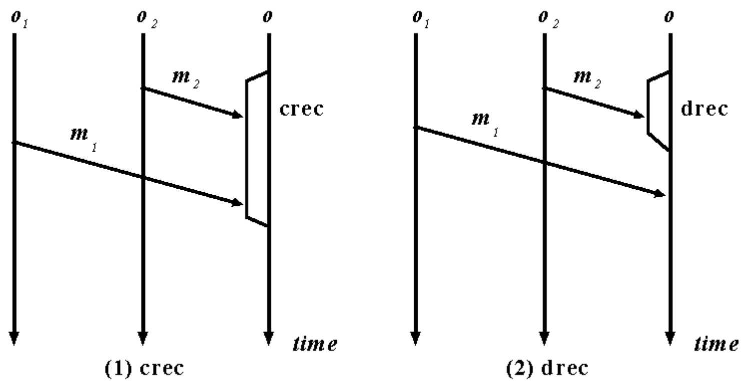

Fig. 3. **crec** and **drec**.

Fig. 4 shows how messages are exchanged through *com* objects. Here, there are two computers p_u and p_t. A thread t_s on an object o_s multicasts a message m to multiple destinations. t_s invokes **mcast**(m,$\langle \ldots, o_d, \ldots \rangle$) on the *com* object com_s. A thread t_s on an object o_d receives the message m by invokey **srec**(o_s).

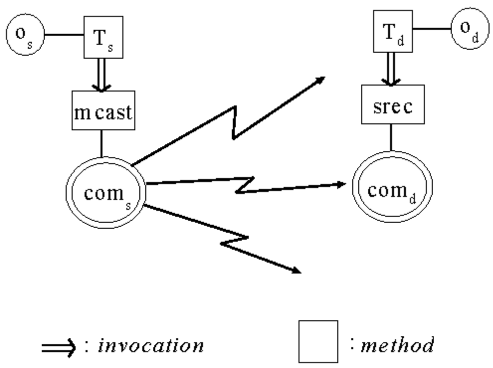

Fig. 4. Inter-object communication.

4. DELIVERY OF MESSAGES IN OBJECTS

4.1 Causally Ordered Delivery of Messages

In the object-based system, *request* and *response* messages sent and received by objects are exchanged among *com* objects in computers which support the objects. The *com* object sends a message m sent by a thread of a method on an object in a computer p_s to the *com* object in a computer p_t which supports the destination object of m. Here, this is referred to as "p_s sends m to p_t." If the *com* object in p_t receives m from p_s, we say "p_t receives m from p_s." The *com* object in p_t receives messages from multiple computers while sending messages to multiple computers. The messages are ordered and then delivered to the objects by the *com* object in computer p_t.

A message m_1 *causally precedes* another message m_2 if the sending event of m_1 *happens before* the sending event of m_2 according to the traditional definition [6, 9]. A message m_1 *totally precedes* another message m_2 iff m_1 and m_2 are delivered to every common destination object in the same order. In addition, m_1 totally precedes m_2 if m_1 causally precedes m_2.

4.2 Transmission

A thread on an object sends messages to objects by invoking the transmission methods **ucast**, **mcast**, and **pcast** on the *com* object. The *com* object delivers messages to destination *com* objects in a network. For example, if a thread t in a computer p_s multicasts a message m to objects o_t and o_u in computers p_t and p_u by **mcast**(m, $\langle o_t, o_u \rangle$), then *com* in p_s sends a pair of instances m_1 and m_2 of message m to p_t and p_u by using of TCP, respectively. We discuss how these message instances transmitted in the network are ordered. Suppose a pair of message instances m_1 and m_2 are sent from computer m_1 and p_s. The message instances m_1 and m_2 transmitted in the network are related according to the following relations, depending on through which transmission method **ucast**, **mcast**, or **pcast** the messages m_1 and m_2 are transmitted:

1. m_1 and m_2 are **mcast** instances of m ($m_1 \approx m_2$) iff m_1 and m_2 are different instances of the same message m which are sent by **mcast**.
2. m_1 and m_2 are **pcast** instances of m ($m_1 \equiv m_1$) iff m_1 and m_2 are paracast by **pcast**.
3. m_1 and m_2 are serially sent ($m_1 \prec\prec m_2$) iff m_1 is sent before m_2 by different transmission methods t_1 and t_2, respectively, and t_2 is invoked after t_1 completes.

It is trivial that neither $m_1 \approx m_2$ nor $m_1 \equiv m_2$ iff $m_1 \prec\prec m_2$. Let us consider an example in which a transaction T_1 in a computer p_s sends a request message r_1 to some object o_1, and another transaction T_2 in p_s sends a request message r_2 to an object o_2. The requests r_1 and r_2 can be independently delivered since different objects o_1 and o_2 are manipulated by r_1 and r_2, respectively. We now define a precedent relation "$\rightarrow$" among a pair of messages m_1 and m_2 sent by a computer p_s. Here, let "$m_1 \prec m_2$" show that a computer sends a message instance m_1 before m_2 in the network.

Definition 1 Let m_1 and m_2 be message instances sent by objects in a computer p_s. m_1 *precedes* m_2 in p_s (i.e., $m_1 \rightarrow m_2$) if m_1 is sent before m_2 in p_s ($m_1 \prec m_2$) and one of the following conditions holds:

1. m_1 and m_2 are sent by a same thread, and ($m_1 \prec\prec m_2$), i.e., neither $m_1 \equiv m_2$ nor $m_1 \approx m_2$.
2. m_1 and m_2 are sent by different conflicting threads t_1 and t_2, respectively.
3. $m_1 \rightarrow m_3 \rightarrow m_2$ for some m_3. ❑

A pair of messages m_1 and m_2 are *independent* ($m_1 \mid m_2$) iff neither $m_1 \rightarrow m_2$, $m_2 \rightarrow m_1$, $m_1 \approx m_2$, nor $m_1 \equiv m_2$. For the request messages r_1 and r_2 presented in the example, $r_1 \mid r_2$ because r_1 and r_2 do not conflict. Each message m is assigned a unique identifier *m.id*. For every pair of instances m' and m'' of m, $m'.id = m''.id$.

In **pcast** and **mcast**, multiple message instances $m_1, \ldots, m_k$ are transmitted. Let $M(m_i)$ be a set $\{m_1, \ldots, m_k\}$ of the message instances to be sent with a message m_i. $M(m_i)$ is referred to as a *message group*. At the network levle, the message instances are serially transmitted by using a protocol like TCP. Suppose the message instances are sent in the order $m_1, \ldots, m_k$. Here, let m_1 be the *first* message *first*(m_1) and m_k be the last message *last*(m_i) in the message group.

Messages to be multicast or parallel-cast at the application level may not be simultaneously sent at the network level. Suppose that three computers p_s, p_t, and p_u are exchanging message instances m_1, m_2, and m_3 at the network level as shown in Fig. 5. According to the traditional causality theory, m_1 causally precedes m_3 because m_1 causally precedes m_2 at the network level in Fig. 5(1). However, m_1 and m_3 are causally concurrent while m_1 causally precedes m_2 in Fig. 5(2). If $m_1.id = m_2.id$, then m_1 and m_2 are **mcast** instances of a same message ($m_1 \approx m_2$). Otherwise, m_1 and m_2 are **pcast** instances ($m_1 \equiv m_2$). If $m_1 \equiv m_3$ or $m_1 \approx m_2$, then m_1 must causally precede m_3 in Fig. 5(2). $m_1 \Rightarrow m_3$ and $m_2 \Rightarrow m_3$ if $m_1 \approx m_2$ or $m_1 \equiv m_2$ in Fig. 5.

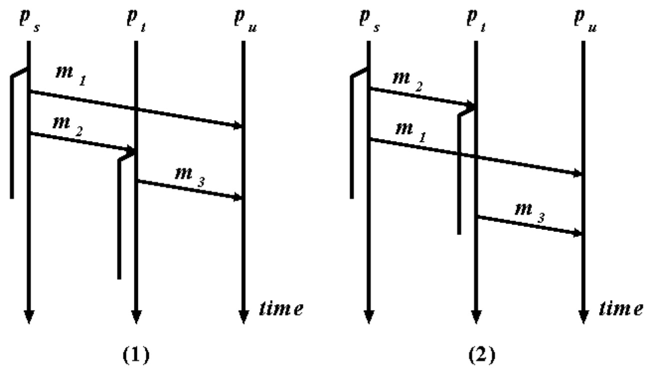

Fig. 5. Message ordering.

4.3 Receipt

A thread t on an object o invokes a **crec** or **drec** method to receive messages $m_1, \ldots, m_k$ from multiple objects $o_1, \ldots, o_k$, [Fig. 6]. The objects $o_1, \ldots, o_k$ are referred to as *sources* of the receipt method **crec** or **drec**. Let $M(m_i)$ be a collection $\{m_1, \ldots, m_k\}$ of messages to be received with a message m_i at a multi-message receipt, named *message group*. For every message m_j in $M(m_i)$, $M(m_j) = M(m_i)$. In the conjunctive receipt method **crec**$(o_1, \ldots, o_k)$ messages are received from all the source objects $o_1, \ldots, o_k$. That is, a method thread invoking **crec** blocks until all the messages are received from $o_1, \ldots, o_k$. Suppose a thread in a computer p_t finishes receiving the messages in $M(m_i)$ on time when t receives a message m_k after receiving all other messages in $M(m_i)$. Here, m_k is referred to as *most significant* for the messages $m_1, \ldots, m_k$ in $M(m_i)$ for **crec**.

Let *msg*(m_i) be a most significant message m_k in $M(m_i)$. A method instance invoking **drec** blocks until at least one message is received from the source objects. Suppose the computer p_t receives a message m_1 before all other messages $m_1, \ldots, m_k$ in $M(m_1)$. Mes-

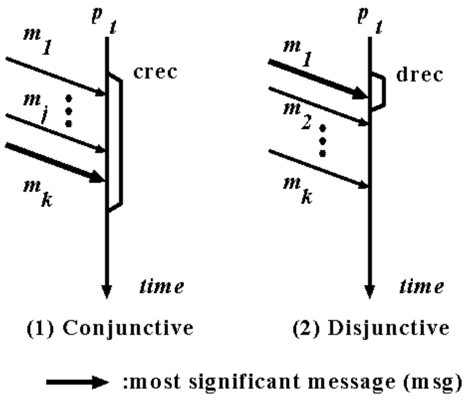

Fig. 6. Multi-message receipt.

sage m_1 is referred to as the *first message* in $M(m_1)$. In **drec**, the object finishes receiving messages $m_1, \ldots, m_k$, only if the first message m_1 is received before all the other messages. The first message m_1 is referred to as the *most significant* for the messages in $M(m_1)$ for **drec**. Here, the other messages $m_2, \ldots, m_k$ are not so significant that the messages are not required to be received. Let $msg(m_i)$ be the most significant message m_i.

Suppose that a computer p_t receives a pair of message instances m_1 and m_2 in a network. Let "$m_1 \prec\prec m_2$" show that p_t receives m_1 before m_2 at the network level. A message m is referred to as *single-received*, *conjunctive-received,* and *disjunctive-received* iff m is received by invoking **srec**, **crec**, and **drec**, respectively, on the *com* object. Table 1 shows conditions that "$m_1 \rightarrow m_2$" holds in case m_1 and m_2 are received by a computer. For example, an entry (**srec**, **crec**) shows a condition "$m_1 \prec msg(m_2)$" for a case that m_1 and m_2 are received by **srec** and **crec**, respectively. This means, m_1 is received before the most significant message of m_2 if $m_1 \rightarrow m_2$.

Definition 2 Let m_1 and m_2 be message instances received by objects in a computer p_t. m_1 *precedes* m_2 in p_t ($m_1 \rightarrow m_2$) if the condition shown in Table 1 is satisfied for m_1 and m_2. ❑

Table 1. Receipt-receipt conditions.

m_1 \ m_2	**srec**	**crec**	**drec**
srec	$m_1 \prec m_2$	$m_1 \prec msg(m_2)$	m_1 *prec* m_2 and $m_2 = msg(m_2)$
crec	$msg(m_1) \prec m_2$	$msg(m_1) \prec msg(m_2)$	$msg(m_1) \prec m_2$ and $m_2 = msg(m_2)$
drec	$m_1 \prec m_2$ and $m_1 = msg(m_1)$	$m_1 \prec msg(m_2)$ and $m_1 = msg(m_1)$	$m_1 \prec m_2$ $m_1 = msg(m_1)$ and $m_2 = msg(m_2)$

Here, m_1 and m_2 are *independent* ($m_1 \mid m_2$) iff neither $m_1 \rightarrow m_2$ nor $m_2 \rightarrow m_1$.

4.4 Receipt and Transmission

If a computer p_s sends a message instance m_2 after receiving another message m_1 at the network level, then "$m_1 \prec m_2$". Table 2 shows conditions that "$m_1 \rightarrow m_2$" holds for the case where m_1 is sent and m_2 is received by a computer.

Definition 3 Let m_1 and m_2 be message instances received and sent by a computer p_s. m_1 *precedes* m_2 in p_s ($m_1 \rightarrow m_2$) if the conditions shown in Table 2 are satisfied. ❑

Table 2. Receipt and transmission conditions.

m_1 \ m_2	**ucast**	**mcast, pcast**
srec	$m_1 \prec m_2$	$m_1 \prec first(m_2)$
crec	$msg(m_1) \prec m_2$	$msg(m_1) \prec first(m_2)$
drec	$m_1 \prec m_2$ and $m_1 = msg(m_1)$	$msg(m_1) \prec first(m_2)$

The relation "$m_1 \prec m_2$" shows that "m_1 causally precedes m_2", which holds at the network level. The precedent relation "$\rightarrow$" defined is referred to as the *significantly precedent* relation among messages. In a system where messages are sent by **mcast** or **pcast** and received by **crec** or **drec**, messages are required to be delivered in the significantly precedent relation "$\rightarrow$". That is, a message m_1 is required to be delivered before another message m_2 if $m_1 \rightarrow m_2$. If multiple messages could be atomically sent to multiple destinations in the network, the following property holds.

Theorem 1 If every set of **mcast**/**pcast** message instances are atomically sent at network level, then m_1 causally precedes m_2 if $m_1 \rightarrow m_2$ for every pair of messages m_1 and m_2. ❑

If **mcast** and **pcast** methods are not realized to be atomic, the "$m_1 \rightarrow m_2$" may hold even if m_1 does not causally precede m_2 as discussed in this section. For example, m_1 and m_3 are parallel-cast. Here, $m_1 \rightarrow m_3$, but m_3 does not causally precede m_2.

Suppose there are four computers p_s, p_t, p_u, and p_v [Fig. 7]. An object in p_s multicasts a message instance m_1 to p_t and p_u. An object in p_u multicasts m_2 to p_t and p_u. Suppose the message instances are received by **drec** in p_t and p_u. Here, $M(m_1) = M(m_2) = \{m_1, m_2\}$. m_2 is the most significant in p_t, but m_1 is the most significant in p_u. Computer p_t sends message m_3 after receiving m_2, while p_u sends m_4 after receiving m_1. Here, $m_2 \rightarrow m_3$ and $m_1 \rightarrow m_4$, but $m_1 \mid m_3$ and $m_2 \mid m_4$. Next, suppose that p_u receives m_2 before m_1 and sends m_4 after receiving m_2. Here, $m_2 \rightarrow m_3$ and $m_2 \rightarrow m_4$. This example shows that the precedent relation among messages depends on the order of receipt of the messages if multiple messages are received by **drec**.

5. PROTOCOL

A *com* object supports inter-object communication facilities in each computer. Here, "object" means not only an object but also a transaction object in a computer. If a method

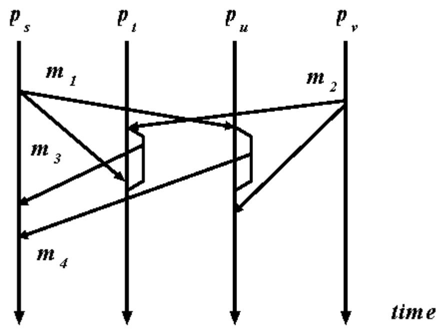

Fig. 7. Disjunctive receipt.

is invoked on an object, a thread of the method is created. The thread sends messages to other objects, e.g., invokes methods on the objects and receives responses. The thread invokes communication methods on the *com* object in a computer to exchange messages with other objects. For example, if **mcast** is invoked, a message is multicast to multiple objects.

In the object-based computation, a thread t is created on an object o. The thread t exchanges messages with other objects by invoking the communication methods on the *com* object. Each thread t has an unique identifier $id(t)$ in the system.

A transaction is realized as a thread of the *init-tran* on the *tran* object in a computer. The transaction identifier is incremented by one each time a transaction is initiated. Hence, $tid(T_1) < tid(T_2)$ if a transaction T_1 is initiated before another transaction T_2 in a computer. Each thread has a variable *iseq* named *invocation sequence number*. $iseq = 0$ when the thread is created. *iseq* is incremented by one each time the thread invokes **ucast**, **mcast**, or **pcast**.

For ordering a pair of message instances m_1 and m_2 in the significant precedent relation $\rightarrow$, it is significant to decide whether or not a pair of the sender threads of m_1 and m_2 conflict. Each thread t is assigned a *compatibility identifier* $cid(t)$. There is a variable c, initially 0, for each object o. Suppose a thread t is initiated. Here, if no method is performed on the object o, $cid(t) := c$. Next, suppose the thread t commits. If any other method is not being performed on the object o, c is incremented by one. If $cid(t_1) = cid(t_2)$, t_1 and t_2 are compatible, i.e., being or can be concurrently performed. Otherwise, t_1 and t_2 conflict or one of t_1 and t_2 is started before the other finishes.

Suppose a message m is sent by a thread t on an object o. The message m has an identifier $m.id$ which is a concatenation of id_1, id_2, and id_3 where $id_1 = cid(t)$, $id_2 = id(t)$, and id_3 is an invocation sequence number(*iseq*) in t, i.e. $id = id_1{:}\ id_2{:}\ id_3$.

For a pair of identifiers a (= $a_1{:}a_2{:}a_3$) and b (= $b_1{:}b_2{:}b_3$), $a < b$ iff

1. $a_1 < b_1$.
2. $a_2 < b_2$ if $a_1 = b_1$.
3. $a_3 < b_3$ if $a_1 = b_1$ and $a_2 = b_2$.

$a = b$ iff $a_1 = b_1$, $a_2 = b_2$, and $a_3 = b_3$. If a pair of messages m_1 and m_2 are sent by **mcast** or **pcast**, $m_1.id = m_2.id$. If a thread sends m_1 before m_2 by different transmission invocations, then $m_1.id_1 = m_2.id_1$ and $m_1.id_2 = m_2.id_2$, but $m_1.id_3 < m_2.id_3$.

For a pair of messages m_1 and m_2 sent in a computer p_t, m_1 is sent before m_2 if one of the following conditions holds:

1. $m_1.id_1 < m_2.id_1$.
2. $m_1.id_3 < m_2.id_3$ if $m_1.id_2 = m_2.id_2$.

If $m_1.id_1 = m_2.id_2$, then m_1 and m_2 are sent by threads which are compatible. The *com* object of a computer p_t maintains an *object* vector $V = \langle v_1, \ldots, v_n \rangle$, where each element v_i takes a message identifier and is used for an object o_i ($i = 1, \ldots, n$) in the group G. Suppose that a thread t on an object o_i in the computer p_t invokes a transmission method, i.e., **ucast**, **mcast**, and **pcast** on the *com* object. Then, message instances are sent by the transmission method and the messages carry the vector V. Here, $m.V$ shows the object vector $\langle V_1, \ldots, V_n \rangle$ carried by a message m.

Next, suppose a thread t on an object o_i invokes **srec**, **crec**, or **drec** to receive messages. The receipt method terminates if the most significant message is received. On completion of the receipt method, the object vector V is updated as $v_j := \max(v_j, m.V_j)$ for $j = 1, \ldots, n$ and $j \neq i$. If **crec** is invoked, V is updated when the last message is received. If **drec** is invoked, V is then updated when the first message is received. The thread invokes a receipt method **srec**, **crec**, or **drec** in order to receive the responses after invoking **ucast**, **mcast**, and **pcast**. In the receipt method, the messages whose $id_3 = m.iseq$ are received as the response. On receipt of a request message m, m is performed and the response m' is sent back. The response message m' carries $m'.id_3 = m.id_3$. **crec**/**drec** receives only messages whose id_3, *i.e.*,*iseq* is the *iseq* of **mcast**/**pcast**.

On receipt of a request message m, request m is performed as a thread on an object o_i in a computer p_t. If the thread commits, the object vector V is changed as $v_j := \max(v_j, m.v_j)$ for $j = 1, \ldots, n$ and $j \neq i$ in a computer p_t.

Messages are ordered by the following rule:

Ordering rule A message m_1 *precedes* another message m_2 ($m_1 \Rightarrow m_2$) iff either one of the following conditions holds:

1. m_1 and m_2 are sent by an object o_i;
   - $m_1.V_i < m_2.V_i$.
2. m_1 is sent by o_i and m_2 is sent by o_j ($i \neq j$);
   - a pair of messages m_1 and m_2 are conflicting requests, and $m_1.V \leq m_2.V$, or
   - m_1 is a response message and m_2 is a request message. ❑

Theorem 2 For every pair of messages m_1 and m_2, then $m_1 \rightarrow m_2$ if $m_1 \Rightarrow m_2$. ❑

Example Suppose there are three computers p_s, p_t, and p_u [Fig. 8]. In each computer, the object vector V is initially $\langle 0, 0, 0 \rangle$. A transaction T sends a pair of requests m_1 and m_2 to p_t and p_u by invoking a communication method **mcast** or **pcast** on the *com* object in p_s. Here, $m_2.id$(= 011) = $m_1.id$ and $m_1.V$(= $\langle 011, 0, 0 \rangle$) = $m_2.V$. On receipt of a request message m_2, the thread s for the request method m_2 is initiated in the computer p_t and is assigned with the object vector of p_t. "011" means that $cid(T) = 0$, $id(T) = 1$, and the event number of the invocation of the communication method is 1. The object vector of s is $\langle 011, 0, 0 \rangle$ when s is initiated but the object vector of p_t is still $\langle 0, 0, 0 \rangle$. Suppose thread s

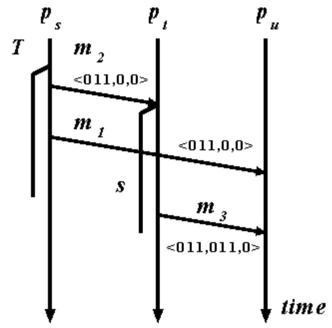

Fig. 8. Example.

sends a request message m_3. The value of $m_3.id$ is "011" and $m_3.V = \langle 011, 011, 0\rangle$. In the ordering rule, message m_1 precedes another message m_3 ($m_1 \Rightarrow m_3$) because $m_1.V = \langle 011, 0, 0\rangle < m_3.V = \langle 011, 011, 0\rangle$. According to the traditional definitions, there is no precedent relation among messages m_1 and m_3 ($m_1 \mid m_3$). ❑

6. EVALUATION

As discussed in this paper, even if a pair of message instances are causally ordered according to the traditional definition, some of the message instances are not required to be causally delivered in this protocol. We show how many request messages are ordered in the protocol. The protocol is implemented as Unix processes on Sun workstations. In the evaluation, "computer" means a workstation and these computers are interconnected with a 100 base-T Ethernet. Each workstation has one or two objects and each object supports four types of methods. Transactions are initiated in each computer. Each transaction invokes some methods and the methods are invoked in a nested manner. In this evaluation, every method is invoked at three levels, so the invocation depth three.

It is significant to consider how many types of methods conflict. Each object supports four types of methods, say, t_1, t_2, t_3, and t_4. A conflict ratio C of methods on an object is defined to be $|\{\langle t_i, t_j\rangle \mid t_i \text{ conflicts with } t_j\}| / |\{\langle t_i, t_j\rangle\}|$. Fig. 9 shows how many messages are not ordered for conflict ratio C. Here, a message ratio M means a ratio of the number of messages ordered by the protocol to the number of messages ordered by traditional group protocols. The horizontal axis of Fig. 9 shows conflict ratios from 0% to 100%. At 100% every pair of methods conflict, and at 0% every pair of methods is compatible. The vertical axis indicates the message ratio M%, i.e., how what percentage of message instances are not ordered according to the ordering rule in the protocol. For example, 60% means that 40% of message instances transmitted at the network are ordered and 60% are not. 100% shows a traditional protocol where message instances are ordered at a network level independently of what each message carries.

Messages are transmitted by transmission methods **ucast**, **mcast**, and **pcast**. In the evaluation, messages are received by the conjunctive receipt method **crec**. We consider the following cases:

1. All the requests are transmitted by **ucast**
2. Half of the requests are transmitted by **ucast** and half are transmitted by **mcast** or **pcast**.
3. All the requests are transmitted by **mcast** or **pcast**.

Each line shows one of the cases. Fig. 9 shows that the more messages invoked by **mcast** or **pcast**, the fewer messages are required to be ordered. For example, when the conflict ratio is 60%, 50.0% of messages are ordered for case 1, 66.3% for case 2, and 73.2% for case 3. Thus the number of messages to be ordered can be reduced by using the protocol.

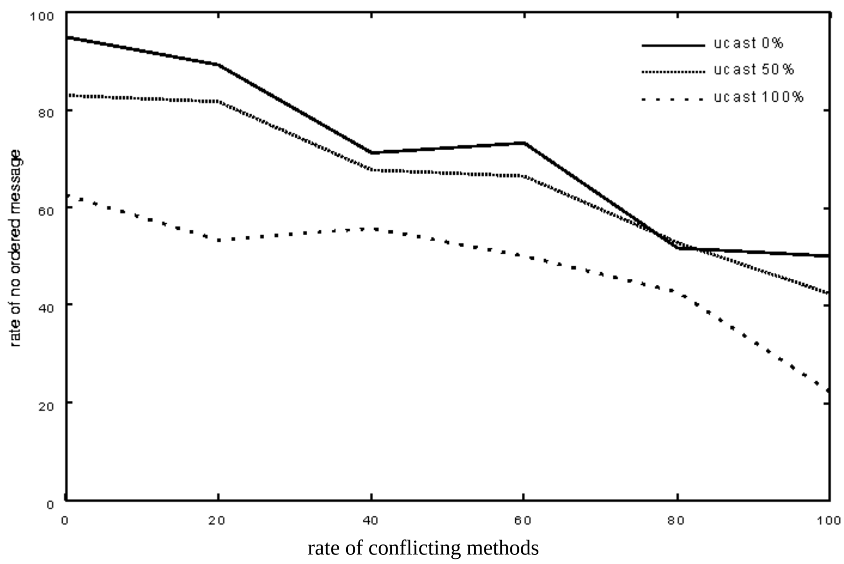

Fig. 9. Evaluation.

7. CONCLUDING REMARKS

In the object-based system, methods are invoked not only serially, but also in parallel, and multiple responses are received in various ways. One message is multicast to multiple destinations and different types of messages are *parallel-cast* to multiple destinations. Multiple messages are received in both conjunctive and disjunctive ways. We defined new types of causal relations among messages transmitted by multicast **mcast** and parallel-cast **pcast**, and received by conjunctive receipt **crec** and disjunctive receipt **drec**, in addition to **ucast** and single-message receipt **srec**. We presented the protocol for ordering message instances transmitted at the network according to the precedent relation. We are now implementing the protocol for supporting objects with causally ordered delivery of messages discussed in this paper.

REFERENCES

1. American National Standards Institute, "Database language SQL," Document ANSI X3.135, 1986,
2. F. Adelstein and M. Singhal, "Real-time causal message ordering in multimedia systems," in *Proceedings of IEEE International Conference on Distributed Computing Systmes-15*, 1995, pp. 36-43.
3. M. Ahamad, M. Raynal, and G. Thia-Kime, "An adaptive protocol for implementing causally consistent distributed services," in *Proceedings of IEEE International Conference on Distributed Computing Systmes-18*, 1998, pp. 86-93.
4. R. Baldoni, A. Mostefaoui, and M. Raynal, "Efficient causally ordered communications for multimedia real-time applications," in *Proceedings of IEEE High Performance Distributed Computing-4*, 1995, pp. 140-147.
5. P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.
6. K. Birman, A. Schiper, and P. Stephenson, "Lightweight causal and atomic group multicast," *ACM Transactions on Computer Systems*, Vol. 9, 1991, pp. 272-314.
7. Defense Communications Agency, "DDN protocol handbook," Vol. 1-3, NIC 50004-50005, 1985.
8. T. Enokido, H. Higaki, and M. Takizawa, "Object-based ordered delivery of messages in object-based systems," in *Proceedings of International Conference on Parallel Processing'99*, 1999, pp. 380-387.
9. L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, Vol. 21, 1978, pp. 558-565.
10. F. Mattern, "Virtual time and global states of distributed systems," *Parallel and Distributed Algorithms*, M. Cosnard and, P. eds., North-Holland, 1989, pp. 215-226.
11. A. Nakamura and M. Takizawa, "Causally ordering broadcast protocol," in *Proceedings of IEEE International Conference on Distributed Computing Systems-14*, 1994, pp. 48-55.
12. Object Management Group Inc., "The common object request broker: architecture and specification," Rev. 2.1, 1997.
13. T. Tachikawa, H. Higaki, and M. Takizawa, "Significantly ordered delivery of messages in group communication," *Computer Communications Journal*, Vol. 20, 1997, pp. 724-731.
14. T. Tachikawa, H. Higaki, and M. Takizawa, "Group communication protocol for realtime applications," in *Proceedings of IEEE International Conference on Distributed Computing Systems-18*, 1998, pp. 40-47.
15. Y. Timura, K. Tanaka, and M. Takizawa, "Group protocol for supporting object-based ordered delivery," in *Proceedings of IEEE International Conference on Distributed Computing Systems-2000 Workshop*, 2000, pp. C-7-C-14.
16. R. Yavatkar, "A protocol for coordination and temporal synchronization in multimedia collaborative applications," in *Proceedings of IEEE International Conference on Distributed Computing Systems-12*, 1992, pp. 606-613.

Youhei Timura was born in 1978. He received his B.E. and M.E. degrees in Computers and Systems Engineering from Tokyo Denki University, Japan in 2000 and 2002, respectively. His research interests include group communication protocols.

Katsuya Tanaka was born in 1971. He received his B.E. and M.E. degrees in Computers and Systems Engineering from Tokyo Denki University, Japan in 1995 and 1997, respectively. From 1997 to 1999, he worked for NTT Data Corporation. He received the D.E. degree from Dept. of Computers and Systems Engineering, Tokyo Denki University, Japan, in 2000. His research interest includes distributed systems, transaction management, recovery protocols, and computer network protocols. He is a member of IEEE CS and IPSJ.

Makoto Takizawa was born in Tokyo, Japan, 1950. He received his B.E. and M.E. degrees in Applied Physics from Tohoku University, Japan, in 1973 and 1975, respectively. He received his D.E. in Computer Science from Tohoku University in 1984. From 1975 to 1986, he worked for Japan Information Processing Developing Center (JIPDEC) supported by the MITI. He is currently a full professor of the Department of Computers and Systems Engineering and also a director of Information Division of TDU Research Institute for Technology. From 1989 to 1990, he was a visiting professor of the GMD-IPSI, Germany. He is also a regular visiting professor of Department of Computer Science and DAKE Centre of Keele University, England since 1990. He was a program co-chair of IEEE ICDCS-18 held in Amsterdam, 1998. He is serving on the program committees of many international conferences like ICPADS, ICOIN, ICNP, DEXA, ISORC, SRDS. He won the best paper awards at the 4th IEEE ICPADS, 1996, and the best paper awards at ICOIN-9, 1994 and ICOIN-12, 1998. He was a chair of IPSJ (Information Processing Society of Japan) SIGDPS in Japan from 1997 to March of 2000. He is now a member of the executive board of IPSJ. His research interest includes communication protocols, group communication protocols, distributed systems, distributed database systems, transaction management, fault-tolerant systems, and security. He is a member of IEEE, ACM, IPSJ, and IEICE.