Vol. 38 No. 6 Transactions of Information Processing Society of Japan June 1997

Regular Paper

Object-based Consistency in Distributed Checkpoints

KATSUYA TANAKA,† HIROAKI HIGAKI† and MAKOTO TAKIZAWA†

In distributed applications, multiple objects cooperate to achieve some objectives. The objects may suffer from kinds of faults. If some object o is faulty, o is rolled back to the checkpoint and objects which have received messages from o are also required to be rolled back. In this paper, we define influential messages whose receivers are required to be rolled back if the senders are rolled back in the object-based computation model. By using the influential messages, an object-based (O) checkpoint is defined to denote a semantically consistent global state of the system while inconsistent with the traditional message-based definition. We show that fewer number of O-checkpoints are taken than the message-based ones.

1. Introduction

In distributed applications like teleconferences[5], a group of multiple autonomous objects are required to cooperate by exchanging messages through communication networks to achieve some objectives. An object o is modeled as a pair of a data structure and a collection P_o of operations. The computation on the objects is based on the message-passing mechanism. On receipt of a request message with an operation *op* in P_o, a thread of *op* is created and is computed in o, and sends back a response message with the result of *op*. During the computation of *op*, *op* may invoke operations on other objects, i.e. operations are *nested*.

In order to tolerate the fault of each object o, o takes a checkpoint where the state of o is saved in the stable storage *log*. If o is faulty, o is *rolled back* to the checkpoint by restoring the state information stored in the log taken at the checkpoint and then the computation on o is restarted. If o is rolled back, objects which have received messages sent by o have to be rolled back so that there is no orphan message[4], i.e. message sent by no object but received by some object. If the sending event of a message m_1 *happens before*[10] m_2, m_1 *causally precedes* m_2[3]. If o is rolled back, objects which have received messages causally preceded by the messages sent by o have to be rolled back in order to prevent from the orphan messages.

Papers[2),4),9),12),13),15),18)] discuss how to take the consistent global checkpoints in the message-based systems. Koo and Toueg[9] present synchronous protocols for taking the global consistent checkpoint and rolling back the processes, which are similar to the two-phase commitment protocol[1),7)]. Leong and Agrawal[12] present the concept of *significant* operation messages if the state of an object is changed by the operations. If o is rolled back, only objects which have received significant messages sent by o are rolled back. Thus, the number of objects to be rolled back can be reduced. However, in the object-based systems, objects send different types of messages, i.e. *request*, *response*, and *data* messages. In the significant messages, the transmissions of requests, responses, and data messages are not considered and operations are not nested.

In this paper, we define *influential* messages by taking into account the kinds of messages sent by the objects and the nested operations. Then, we discuss *object-based* (O) checkpoints which can be taken from the object point of view even though it may not be consistent with the traditional message-based definition. This means that some objects do not need to take the checkpoints even though they have to take the checkpoints in the message-based system. If an object o is rolled back, only objects which have received influential messages sent by o are rolled back.

In Section 2, we first present the system model. In Section 3, we discuss the influential messages and define the object-based checkpoint. In Section 4, we show how many checkpoints can be reduced by taking only the object-based checkpoint.

2. System Model

2.1 Objects

A distributed system is composed of multiple objects interconnected by a communication

† Department of Computers and Systems Engineering, Tokyo Denki University

network. Each object o is defined to be a pair of a data structure and a collection P_o of operations. Another object o' can manipulate o only through an operation op in P_o. On receipt of a *request* message m with op from o', op is computed on o. Then, op sends back the *response* message with the result of op. op is computed as a sequence of *actions* on o. The actions are *private* primitive units of computation of o, i.e. cannot be directly invoked from the outside of o. op may invoke operations on other objects, i.e. op is *nested*. Thus, op is realized by a sequence of actions and operations on other objects. The operations are *descendants* of op if they are invoked by op or by descendants of op. op *commits* only if all the actions and operations in op complete successfully. If some action or operation invoked in op are faulty, all the effects of op are removed, i.e. op aborts.

A global state S of the system is given by a tuple $\langle s_1, \ldots, s_n \rangle$, where each s_i is a local state of o_i, and a state of the network. The network state is a collection of messages transmitted but not received by the objects. A message m is referred to as *lost* in S iff m is sent but not received by some destination object and m is not in the network. If m is logged in the network, the receiver of m can take m from the log after S. A message m is an *orphan* at S iff m is received but not sent in the system. Chandy and Lamport [4] define a consistent global state to be one where there is no orphan. Here, it is not discussed what information each message carries. Hence, it is called a *message-based* system.

2.2 Types of Operations

Each object o is manipulated by the operations supported by o. For every state s of o, $op(s)$ denotes a state obtained by applying op to s. For every pair of operations op_1 and op_2, $op_1 \circ op_2$ means that op_2 is applied after op_1.

[Definition] Operations op_1 and op_2 of an object o are *compatible* if and only if (iff) $op_1 \circ op_2(s) = op_2 \circ op_1(s)$ for every state s of o. □

op_1 and op_2 *conflict* [1] iff they are not compatible, i.e. $op_1 \circ op_2(s) \neq s$ for some state s of o. The conflicting relation among the operations is assumed to be specified in the definition of o.

op_1 and op_2 are *mutually exclusive* in o iff op_1 and op_2 cannot be simultaneously computed in o. Unless op_1 and op_2 are mutually exclusive, they can be *interleaved* in o. We assume that op_1 and op_2 are mutually exclusive if they conflict. If op_1 is issued to o while op_2 is being computed in o, op_1 has to wait until op_2 completes. For example, two *withdraw* operations are mutually exclusive and conflict.

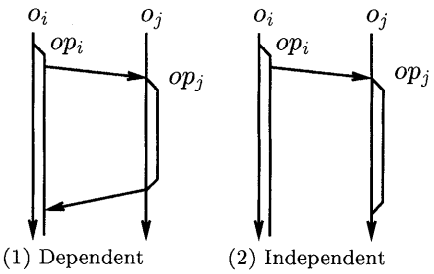


Fig. 1 Closed computation.

An object o supports two kinds of abstract operations, i.e. one changes the state of o and the other not. For example, *deposit* changes the state of a *Bank* object and *check* does not change the state. An operation op supported by o is *transforming* if op changes the state of o, i.e. $op(s) \neq s$ for some state s of o. op is *stable* if neither op nor any descendant of op changes any object. In this paper, the specification of op defines whether or not each operation op is stable. *check* is stable and *withdraw* is transforming.

Suppose that a stable operation op_1 and an unstable one op_2 are invoked by an untransforming one op. Before invoking op_2, neither op nor any descendant of op changes any object. On receipt of the response of op_2, op knows that op_2 is unstable. op is *pending* if op is unstable but every descendant of op which is computed so far is stable. That is, op has not change the state, yet, but will change it later.

2.3 Invocations

Suppose that an operation op_i of an object o_i invokes op_j of o_j. There are two ways to compute op_j: (1) *dependent* and (2) *independent* computations. In the dependent one, op_i waits for the completion of op_j after invoking op_j (**Fig. 1** (1)). The invocation of op_j and the action for waiting for the completion of op_j in op_i are named *fork* and *wait*, respectively. This also shows the remote procedure call. In the independent one, op_i does not wait for the completion of op_j (Fig. 1 (2)). op_j is computed independently of op_i. The independent computation is called *detach*.

There are two kinds of messages transmitted among the objects: (1) *control* and (2) *data* messages. The control messages mean requests and responses to be used to invoke the oper-

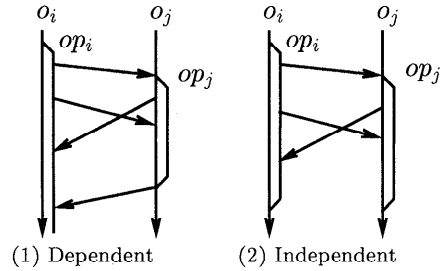


Fig. 2 Open computation.

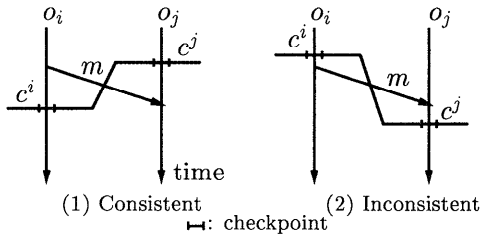


Fig. 3 Consistent checkpoint.

ations and notify the completion of the operations, respectively. After the operations are invoked, they may communicate with other operations by exchanging data messages through *send* and *receive* primitive actions.

There are two kinds of computation of op_j depending on whether op_i and op_j exchange messages: (1) *closed* and (2) *open* computations. Suppose that op_i sends a data message m to op_j. The computation of op_i depends on op_j after receiving m. Hence, if op_i is aborted, op_j is also required to be aborted. Suppose that op_i invokes op_j. If op_i and op_j do not communicate with one another, op_j is closed for op_i (Fig. 1). Otherwise, op_j is open for op_i (**Fig. 2**).

A message m *participates* in an operation op if (1) m is a request or response of op or (2) m is a data message received in op. Let $Op(m)$ denote an operation in which m participates. o_i sends a request m_1 of an operation op_2 to o_j. On receipt of m_1, o_j computes op_2. Let op_k^h denote the computation of op_k in o_h. o_j sends back a response m_2 of op_2^j to o_i if op_2^j is dependent. Here, m_1 and m_2 participate in op_2^j, i.e. $Op(m_1) = Op(m_2) = op_2^j$.

3. Object-based Checkpoints

We discuss an *object-based* (O) *checkpoint* which can be taken from the object point of view but may not be consistent with the message-based definition. We do not assume that the computations of the objects are deterministic.

3.1 Consistent Checkpoints

We assume that each object o_i may stop by fault. o_i takes a local checkpoint c^i where the state of o_i is stored in the log l_i. If the state of o is so large that it takes time or it is difficult to store the state in the log, the operations computed in o are stored in the log. If o_i is faulty, o_i is rolled back to c^i by restoring the state stored in the log l_i to o_i. If o_i is rolled back to c^i, other objects have to be rolled back to the local checkpoints if they had received messages sent by o_i. A *global checkpoint* c is a tuple $\langle c^1, \ldots, c^n \rangle$ of the local checkpoints. From here, a term *checkpoint* means a *global* one. If o_i sends a message m before taking c^i but o_j receives m from o_i after taking c^j, m is an *orphan*. c is *consistent* if there is no orphan [4] at c. In **Fig. 3**, an object o_i sends a message m to o_j. In Fig. 3 (1), the checkpoint $\langle c^i, c^j \rangle$ is consistent with the message-based definition. In Fig. 3 (2), $\langle c^i, c^j \rangle$ is inconsistent because m is an orphan. Many papers [2),4),6),15)] discuss how to take the consistent checkpoints in the message-based system.

Leong and Agrawal [12)] discuss the concept of *significant* messages. For example, if a message m is *write* in Fig. 3, m is significant because the local state of o_j is changed on receipt of m. If o_i is rolled back, o_j has to be rolled back. However, o_j is not rolled back if m is *read*, i.e. not significant. In the object-based computation, kinds of messages, i.e. request, response, and data messages are exchanged among the operations in the objects. In the significant messages, it is discussed if only the request messages are significant. The message-based systems do not discuss what information messages carry and how messages are used among the objects.

3.2 Dependent Invocation

Suppose that an operation op_1^i in o_i invokes op_2^j in o_j. There are four ways to invoke op_2^j:

(1) closed dependent,
(2) open dependent,
(3) closed independent, and
(4) open independent computations of op_2^j.

Figure 4 and **Fig. 5** show possible checkpoints c_k^i and c_h^j to be taken in o_i and o_j, respectively, in the dependent invocation. Here, let $\pi c_j(op^j, c^j)$ be a set of operations which (1) precede op^j and (2) succeed a checkpoint c^j or are being computed at c^j in o_j. Suppose that $\pi c_j(op_2^j, c_1^j) = \{op_{21}^j, \ldots, op_{2l}^j\}$.

3.2.1 Closed Invocation

We discuss whether or not each inconsistent checkpoint $\langle c_k^i, c_h^j \rangle$ shown in Fig. 4 can be taken. A checkpoint c is *object-based* (*O-checkpoint*) iff every object can be rolled back to c and then can be restarted from c from the object point of view. First, suppose that the operations are invoked in the closed dependent way.

Here, the O-checkpoint c may be inconsistent with the message-based definition. For example, $\langle c_1^i, c_3^j \rangle$ and $\langle c_1^i, c_4^j \rangle$ are inconsistent. If op_2^j is stable, the state denoted by c_2^j is the same as c_3^j and c_4^j. $\langle c_1^i, c_2^j \rangle$ is consistent with the message-based definition because there is no orphan. Hence, $\langle c_1^i, c_3^j \rangle$ and $\langle c_1^i, c_4^j \rangle$ are object-based. Even if op_2^j is not stable, $\langle c_1^i, c_3^j \rangle$ is object-based if op_2^j is pending because op_2^j has not yet changed the state at c_3^j. If o_i is rolled back to c_1^i, op_1^i is undone. Suppose that o_j takes c_3^j where op_2^j is partially computed. op_2^j is required to be undone while other operations being computed at c_3^j may not have to be undone.

There are two kinds of checkpoints, i.e. *complete* and *incomplete* ones. Suppose that o_j is rolled back to the O-checkpoint c_h^j. If c_h^j is complete, the state of o_j is just restored. If c_h^j is incomplete, the operations being computed at c_h^j have to be undone. However, no operation invoked by the operations needs to be rolled back since the operations are stable. Hence, $\langle c_1^i, c_3^j \rangle$ is object-based where c_3^j is incomplete. $\langle c_2^i, c_3^j \rangle$ and $\langle c_2^i, c_4^j \rangle$ are also object-based.

Let us consider the inconsistent checkpoints $\langle c_4^i, c_1^j \rangle$, $\langle c_4^i, c_2^j \rangle$, and $\langle c_4^i, c_3^j \rangle$. If op_2^j is stable, c_2^j denotes the same state as c_4^j since op_2^j does not change the state. Hence, $\langle c_4^i, c_2^j \rangle$ is object-based since $\langle c_4^i, c_4^j \rangle$ is consistent. Here, suppose that op_2^j is *read* and some *write* op_{2h}^j preceding op_2^j is computed after c_1^j. op_2^j reads data written by op_{2h}^j. Hence, c_1^j denotes a state different from c_4^j. If no operation following c_1^j and preceding op_2^j conflicts with op_2^j, op_2^j sends the same result even if op_2^j is computed before op_{21}^j. Hence, if op_2^j is stable and no operation in $\pi c_j(op_2^j, c_1^i)$ conflicts with op_2^j, $\langle c_4^i, c_1^j \rangle$ is object-based. $\langle c_4^i, c_2^j \rangle$ is also object-based. $\langle c_4^i, c_3^j \rangle$ is object-based where c_3^j is incomplete.

$\langle c_5^i, c_h^j \rangle$ is similarly discussed. $\langle c_5^i, c_1^j \rangle$, $\langle c_5^i, c_2^j \rangle$, and $\langle c_5^i, c_3^j \rangle$ are also object-based. **Table 1** summarizes the message-based inconsistent but object-based (O) checkpoints, where checkpoints marked * denote incomplete ones if op_2^j is pending.

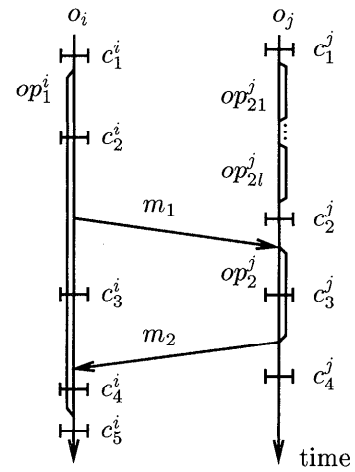


Fig. 4 Closed dependent.

Table 1 O-checkpoints for Fig. 4.

o_i	o_j	Conditions
c_1^i	c_3^j*	op_2^j is pending or stable.
	c_4^j	op_2^j is stable.
c_2^i	c_3^j*	op_2^j is pending or stable.
	c_4^j	op_2^j is stable
c_4^i c_5^i	c_1^j	op_2^j is stable and no operation in $\pi_j(op_2^j, c_1^j)$ conflicts with op_2^j.
	c_2^j, c_3^j*	op_2^j is stable.

3.2.2 Open Invocation

op_2^j is open for op_1^i in Fig. 5, where op_2^j and op_1^i communicate by transmitting data messages. Here, c_{31}^h means a point where no data message is communicated after invoking the operation, and c_{33}^h means a point where data messages are communicated until the end of the operation ($h = i, j$). For c_1^i, c_2^i, c_4^i, c_5^i and c_1^j, c_2^j, c_4^j, the same result as **Table 2** is obtained. $\langle c_2^i, c_{3h}^j \rangle$ ($h = 1, 2, 3$) is object-based if op_2^j is pending or stable. $\langle c_{31}^i, c_{33}^j \rangle$ cannot be taken because m_4 is an orphan. Thus, the data messages are not allowed to be orphans.

3.3 Independent Invocation

Next, suppose that the invocation of op_2^j is independent. First, suppose that op_2^j is closed for op_1^i (**Fig. 6**). $\langle c_1^i, c_4^j \rangle$ and $\langle c_2^i, c_4^j \rangle$ are object-based if op_2^j is stable. In addition, $\langle c_1^i, c_3^j \rangle$ and $\langle c_2^i, c_3^j \rangle$ are object-based if op_2^j is pending or stable where c_3^j is incomplete. **Table 3** shows the

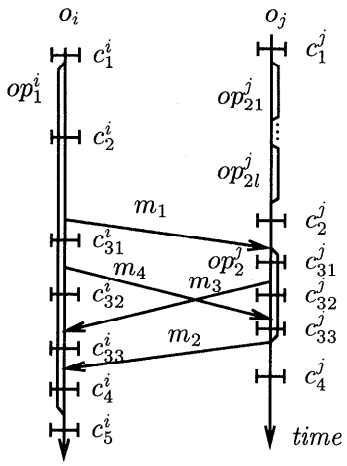


Fig. 5 Open dependent.

Table 2 O-checkpoints for Fig. 5.

o_i	o_j	Conditions
c_1^i	c_{31}^j*, c_{32}^j*, c_{33}^j*	op_2^j is pending or stable.
c_2^i	c_4^j	op_2^j is stable
c_{31}^i	c_4^j	op_2^j is stable.
c_4^i c_5^i	c_1^j	op_2^j is stable and no operation in $\pi_j(op_2^j, c_1^j)$ conflicts with op_2^j.
	c_2^j, c_{31}^j*, c_{32}^j*, c_{33}^j*	op_2^j is stable.

message-based inconsistent but O-checkpoints.

Table 4 shows the message-based inconsistent but O-checkpoints where op_2^j is open (**Fig. 7**).

3.4 Influential Messages

Following the points discussed in this section, we define the influential messages as follows.

[Definition] Suppose that op_2^j sends a message m to op_1^i. Let c^i be a checkpoint most recently taken by o_i. m is *influential* iff one of the following conditions is satisfied:

(1) If m is a request message, $Op(m)$ $(= op_1^i)$ is unstable.

(2) If m is a response message, $Op(m)$ $(= op_2^j)$ is unstable or some operation in $\pi c_i(Op(m), c^j)$ conflicts with $Op(m)$.

(3) If m is a data message,

(3-1) $Op(m)$ $(= op_2^j)$ is being computed, or

(3-2) $Op(m)$ is unstable or conflicts with some operation in $\pi c_i(op_1^i, c_i)$. □

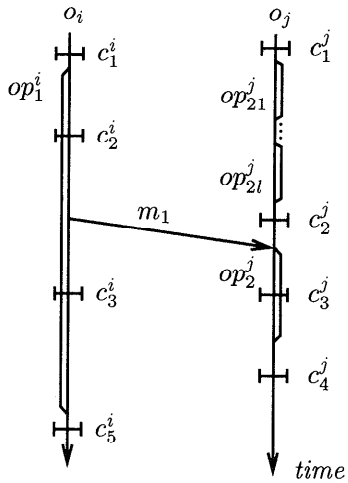


Fig. 6 Closed independent.

Table 3 O-checkpoints for Fig. 6.

o_i	o_j	Conditions
c_1^i	c_3^j*	op_2^j is pending or stable.
c_2^i	c_4^j	op_2^j is stable.

If an operation op_i is undone, only operations receiving influential messages from op_i are required to be undone.

Now, we define an O-checkpoint where the system state is consistent even if the objects are rolled back to the O-checkpoints and then are restarted.

[Definition] A global checkpoint $c = \langle c^1, \ldots, c^n \rangle$ is *object-based* (*O-checkpoint*) iff no influential message is an orphan at c. □

For example, suppose that op_2^j is not stable in Fig. 4. Here, $\langle c_4^i, c_3^j \rangle$ is not object-based since m_2 is influential. If op_2^j is stable, $\langle c_4^i, c_3^j \rangle$ is object-based. The O-checkpoints may not be consistent with the message-based definition. However, the objects can be rolled back to the O-checkpoints and be restarted from the O-checkpoints.

By using the synchronous protocols similar to the two-phase commitment protocol [7),9)], the O-checkpoints can be taken and the objects can be rolled back and restarted. In the checkpointing algorithm, each object o_i has to decide whether or not a message m received by o_i is influential. First, o_i has to know if an operation op^i receiving m is stable. o_i can know from the specification if op^i is stable. Secondly, o_i has to identify the operations to be included in

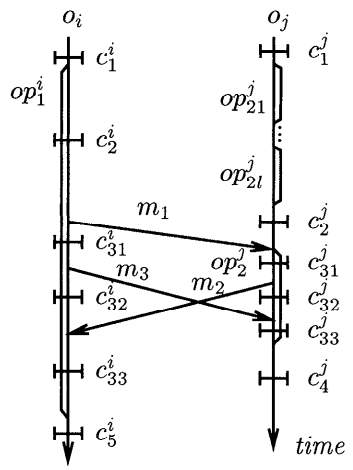


Fig. 7 Open independent.

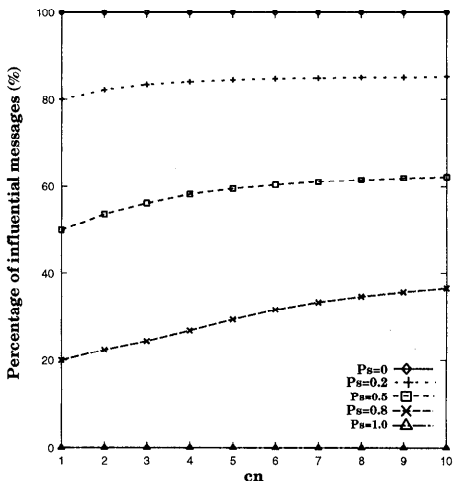


Fig. 8 Number of influential messages.

Table 4 O-checkpoints for Fig. 7.

o_i	o_j	Conditions
c_1^i	c_{31}^j *, c_{32}^j *, c_{33}^j *	op_2^j is pending or stable.
c_2^i	c_4^j	op_2^j is stable.
c_{31}^i	c_4^j	op_2^j is stable.
c_5^i	c_1^j	op_2^j is stable and no operation in $\pi_j(op_2^j, c_1^j)$ conflicts with op_2^j.
	c_2^j, c_{31}^j *	op_2^j is stable.

$\pi c_i(op^i, c^i)$ for the checkpoint c^i most recently taken in o_i. Here, o_i keeps in record the operations computed after c^i in the log l_i. When op^i is initiated to be computed in o_i, o_i looks for operations conflicting with op^i in the log l_i. Thus, all information to make a decision on the influential messages is fixed when the operations receiving the messages are initiated. That is, on receipt of a message m, o_i can decide whether or not m is influential by using the information obtained already.

4. Evaluation

First, we show how many influential messages are transmitted from one object to others. In order to make the evaluation simpler, we make the following assumptions:

(1) There are two objects o_i and o_j in the system.

(2) o_i invokes an operation in o_j every u time units.

(3) o_i invokes randomly one of four kinds of operations, i.e. open dependent, open independent, closed dependent, and closed independent ones.

(4) Every two stable operations are compatible but every unstable operation conflicts with every other operation.

(5) In the open invocation of o_j, o_i sends one message to o_j and o_j sends one to o_i.

(6) o_j takes a checkpoint after every cn operations are invoked.

Here, let P_s denote a probability that an operation invoked by o_i is stable. **Figure 8** shows the percentages of influential messages for the total number of messages which o_j receives. For example, if 80% of operations are stable in o_j, i.e. $P_s = 0.8$, only 20–40% of messages are influential. If $P_s = 0.5$, 50–60% of the messages are influential.

Next, we show how many checkpoints taken by o_j are object-based. o_j takes one checkpoint each time cn operations are computed in o_j. If o_j receives no influential message from o_i after taking the checkpoint most recently taken, the checkpoint is object-based. **Figure 9** shows the probability that each checkpoint taken every cn time units is object-based. Figure 9 shows that the more frequently the object initiates the checkpoint procedure, the fewer number of O-checkpoints are taken than the consistent message-based checkpoints. **Figure 10** shows the number of O-checkpoints for the checkpoint frequency $f = \frac{1}{cn}$. $f = 1$ means that the checkpoint procedure is initiated each time one operation is computed in o_j. For $P_s = 0.8$, the number of O-checkpoints gets only 1.51 times

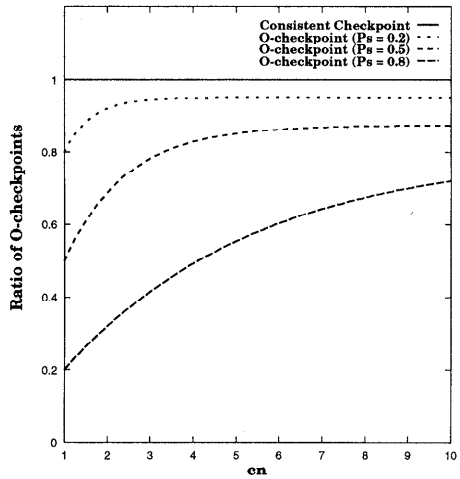


Fig. 9 Ratio of O-checkpoints.

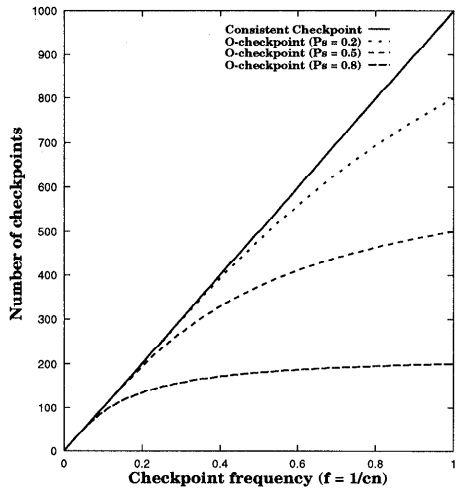


Fig. 10 Number of O-checkpoints.

more even if the checkpoint procedure is initiated two times more frequently for $f = 0.1$. If the checkpoint procedure is four times more frequently initiated, only 2.02 times more O-checkpoints are taken for $f = 0.1$. This means that the objects required to be more available can take the checkpoints more frequently. Even if some object takes the checkpoint more frequently, the objects which do not receive influential messages do not take the checkpoints. These results obtained here can be adopted to more cases including more than two objects.

5. Concluding Remarks

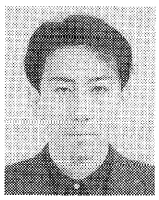
This paper has discussed how to take the *object-based* (O) checkpoints which can be taken from the application point of view and may not be consistent with the traditional message-based definition. We have defined the *influential messages* on the basis of the semantics of requests, responses, and data messages where the operations are nested. Only objects receiving influential messages are rolled back if the senders of the influential messages are rolled back. By using the influential messages, we have defined the *object-based checkpoint* which can be taken from the object point of view but may be inconsistent with the message-based definition. We have shown how much we can reduce the number of checkpoints to be taken if each object takes only O-checkpoints.

References

1) Bernstein, P.A., Hadzilacos, V. and Goodman, N.: *Concurrency Control and Recovery in Database Systems*, Addison-Wesley (1987).
2) Bhargava, B. and Lian, S.R.: Independent Checkpointing and Concurrent Rollback for Recovery in Distributed Systems – An Optimistic Approach, *Proc. 7th Symp. on Reliable Distributed Systems*, pp.3–12 (1988).
3) Birman, K.P. and Joseph, T.A.: Reliable Communication in the Presence of Failures, *ACM Trans. Computer Systems*, Vol.5, No.1, pp.47–76 (1987).
4) Chandy, K.M. and Lamport, L.: Distributed Snapshots : Determining Global States of Distributed Systems, *ACM Trans. Computer Systems*, Vol.3, No.1, pp.63–75 (1985).
5) Ellis, C.A., Gibbs, S.J., and Rein, G.L.: Groupware, *Comm. ACM*, Vol.34, No.1, pp.38–58 (1991).
6) Fischer, M.J., Griffeth, N.D., and Lynch, N.A.: Global States of a Distributed System, *IEEE Trans. Softw. Eng.*, Vol.SE-8, No.3 (1982).
7) Gray, J.: *Notes on Database Operating Systems, An Advanced Course*, Lecture Notes in Computer Science, Vol.60, pp.393–481 (1978).
8) Higaki, H. and Takizawa, M.: Group Communication Protocol for Flexible Distributed Systems, *Proc. IEEE ICNP-96*, pp.48–55 (1996).
9) Koo, R. and Toueg, S.: Checkpointing and Rollback-recovery for Distributed Systems, *IEEE Trans. Computers*, Vol.C-13, No.1, pp.23–31 (1987).
10) Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System, *Comm. ACM*, Vol.21, No.7, pp.558–565 (1978).
11) Lin, L. and Ahamad, M.: Checkpointing and Rollback-recovery in Distributed Object Based Systems, *Proc. 9th Symp. on Reliable Distributed Systems*, pp.97–104 (1990).

12) Leong, H.V. and Agrawal, D.: Using Message Semantics to Reduce Rollback in Optimistic Message Logging Recovery Schemes, *Proc. IEEE ICDCS-14*, pp.227–234 (1994).
13) Manivannan, D. and Singhal, M.: A Low-overhead Recovery Technique Using Quasi-synchronous Checkpointing, *Proc. IEEE ICDCS-16*, pp.100–107 (1996).
14) Nakamura, A. and Takizawa, M.: Causally Ordering Broadcast Protocol, *Proc. IEEE ICDCS-14*, pp.48–55 (1994).
15) Ramanathan, P. and Shin K.G.: Checkpointing and Rollback Recovery in a Distributed System Using Common Time Base, *Proc. 7th IEEE Symp. on Reliable Distributed Systems*, pp.13–21 (1988).
16) Tachikawa, T. and Takizawa, M.: Communication Protocol for Group of Distributed Objects, *Proc. IEEE ICPADS'96*, pp.370–377 (1996).
17) Tanaka, K. and Takizawa, M.: Distributed Checkpointing Based on Influential Messages, *Proc. IEEE ICPADS'96*, pp.440–447 (1996).
18) Wang, Y.M. and Fuchs, W.K.: Optimistic Message Logging for Independent Checkpointing in Message-passing Systems, *Proc. IEEE Symp. on Reliable Distributed Systems*, pp.147–154 (1992).

(Received July 22, 1996)
(Accepted March 7, 1997)



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. He is now working for NTT Data. His research interests include distributed transaction management, and distributed recovery algorithms, and distributed object systems.

Hiroaki Higaki was born in Tokyo, Japan, on April 6, 1967. He received the B.E. degree from the Department of Mathematical Engineering and Information Physics, the University of Tokyo in 1990. From 1990 to 1996, he was in NTT Software Laboratories. Since 1996, he is in the Department of Computers and Systems Engineering, Tokyo Denki University. He received the D.E. degree from Tokyo Denki University in 1997. His research interest includes distributed algorithms, distributed operating systems and computer network protocols. He received IPSJ Convention Award in 1995. He is a member of IEEE CS, ACM and IEICE.

Makoto Takizawa was born in 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 1983. From 1975 to 1986, he worked for Japan Information Processing Developing Center (JIPDEC) supported by the MITI. He is currently a Professor of the Department of Computers and Systems Engineering, Tokyo Denki University since 1986. From 1989 to 1990, he was a visiting professor of the GMD-IPSI, Germany. He is also a regular visiting professor of Keele university, England since 1990. He was a vice-chair of IEEE ICDCS, 1994 and serves on the program committees of many international conferences. His research interest includes communication protocols, group communication, distributed database systems, transaction management, and groupware. He is a member of IEEE, ACM, IPSJ, and IEICE.