マルチメディア通信と分散処理 94－9
分散システム/インターネット運用技術 14－9
（1999. 7. 15）

Asynchronous Checkpointing Protocol in Object-Based Systems

Katsuya Tanaka, Hiroaki Higaki, and Makoto Takizawa

Tokyo Denki University
E-mail {katsu, hig, taki}@takilab.k.dendai.ac.jp

We discuss how to take checkpoints in object-based systems. If some object is faulty, not only the object but also other objects which have received messages from the object are required to be rolled back to the checkpoints. An object-based consistent (O-consistent) checkpoints are semantically consistent in the object-based system while inconsistent with the traditional message-based definition. We also present an asynchronous algorithm for taking O-checkpoints.

分散オブジェクト環境における非同期なチェックポイント取得方法

田中 勝也　桧垣 博章　滝沢 誠

東京電機大学理工学部経営工学科
E-mail {katsu, hig, taki}@takilab.k.dendai.ac.jp

分散オブジェクト環境では、通信網で相互接続された複数のオブジェクトがメッセージの送受信により協調動作を行う。あるオブジェクトからの要求メッセージの受信において、オブジェクトは要求されたメソッドを起動し、応答メッセージを返す。さらに、起動されたメソッドが他のオブジェクトのメソッドを起動する場合もある。本論文では、こうした環境におけるシステムの意味的に正しい状態を定義し、正しい状態を非同期に決定するためのチェックポイントプロトコルを提案する。また、評価を行い、従来の方式よりも取得するチェックポイント数が削減されることを示す。

1 Introduction

Distributed applications are composed of multiple autonomous objects cooperating by exchanging messages through communication networks. On receipt of a request message with a method *op*, *op* is performed on an object *o* and a response message with the result of *op* is sent back. The method *op* may invoke methods on other objects, i.e. invocations of methods are *nested*. The conflicting relation among the methods is defined based on the semantics of the object *o* [4, 15]. If a pair of methods op_1 and op_2 conflict, the state obtained by performing op_1 and op_2 depends on the computation order of op_1 and op_2.

In order to make the system fault-tolerant, each object *o* takes a checkpoint where the state is saved in the stable storage *log*. If the object *o* is faulty, *o* is ***rolled back*** to 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 [2], i.e. message sent by no object but received by some object.

Papers [1,2,6,8–10,14] discuss how to take globally consistent checkpoints in the message-based systems. Koo and Toueg [6] present synchronous protocols for taking checkpoints and rolling back processes. Leong and Agrawal [8] present the concept of ***significant*** requests, i.e. the state of an object is changed by performing the request. If the object *o* is rolled back, only objects which have received significant requests sent by *o* are also rolled back. Thus, the number of objects to be rolled back can be reduced. However, in the object-based systems, different types of messages, i.e. *request* and *response* messages are exchanged among the objects. In the significant requests, the transmissions of requests and responses are not considered and invocations of methods are not nested.

The authors [13] define ***object-based*** consistent (***O-consistent***) checkpoints which can be taken based on the conflicting relation among the methods in various kinds of invocations like synchronous and asynchronous one even though it may be inconsistent with the traditional message-based definition. Higaki [5] discusses an asynchronous checkpointing protocol in the message-based system, where processes a asynchronously not only take checkpoints but also are restarted. In this paper, we discuss how to take O-consistent checkpoints in an asynchronous manner by extending the Higaki's algorithm to the object-based system.

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 a protocol for taking O-consistent checkpoints.

2 System Model

2.1 Objects

A distributed system is composed of multiple objects. Each object *o* is defined to be a pair of a data structure and a collection of methods. On receipt of a ***request*** message *m* with a method *op*, *op* is performed on *o*. Then, the ***response*** message with the result of *op* is sent back. In this paper, we consider a synchronous invocation, i.e. remote procedure call (RPC). The method *op* may invoke methods on other objects, i.e. invocation of *op* is ***nested***.

A message *m* is ***lost*** 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 again from the log. A message m is an *orphan* iff m is received but not sent in the system. Chandy [2] defines a consistent global state to be one where there is no orphan message. Here, it is not discussed what information each message carries. Hence, the system is referred to as *message-based*.

2.2 Types of methods

Let $op(s)$ denote a state obtained by applying a method op to a state s of an object o. $op_1 \circ op_2$ means that a method op_2 is performed after op_1 completes.

[Definition] A pair of methods op_1 and op_2 of an object o are *compatible* iff $op_1 \circ op_2(s) = op_2 \circ op_1(s)$ for every state s of o [4]. □

op_1 and op_2 *conflict* iff they are not compatible. The conflicting relation among the methods is assumed to be specified in the definition of the object o.

An object o supports two kinds of abstract methods, i.e. *update* type of method which changes the state of o and non-update method. For example, *deposit* of a *Bank* object is an update method and *check* is a non-update one. In this paper, we assume that the types of the methods are specified in the definition of the object.

A message m *participates* in a method op if m is a request or response of op. Let $Op(m)$ denote a method in which m participates. An object o_i sends a request m_1 of a method op_2 to another object o_j. On receipt of m_1, op_2 is performed on o_j. Let op_k^h denote an *instance* of a method op_k, i.e. computation of op_k in o_h. The object o_j sends a response m_2 of op_2^j to o_i. 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 Consistent Checkpoints

We discuss an *object-based consistent checkpoint* which can be taken from the objects 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 Message-based checkpoints

An object o_i takes a local checkpoint c^i where the state of o_i is stored in the log l_i. If the object o_i is faulty, o_i is rolled back to the local checkpoint c^i by restoring the state stored in the log l_i. Then, other objects have to be rolled back to the 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 after taking c^j, m is an *orphan*. The checkpoint c is *consistent* if there is no orphan [2] at c. This definition is referred to as *message-based consistent checkpoint*. In Figure 1, an object o_i sends a message m to o_j. In Figure 1(1), the checkpoint $\langle c^i, c^j \rangle$ is consistent with the message-based definition. In Figure 1(2), $\langle c^i, c^j \rangle$ is inconsistent because m is an orphan. Papers [1–3, 10] discuss how to take consistent checkpoints in the message-based system.

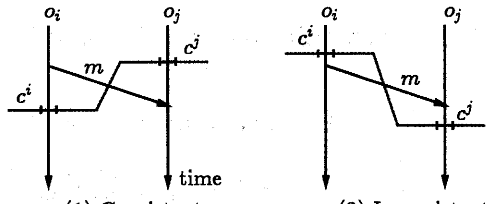

Figure 1: Message-based checkpoints.

Leong and Agrawal [8] discuss the concept of *significant* requests. For example, if a request m is *write* in Figure 1, m is significant. If o_i is rolled back, o_j has to be rolled back. However, o_j is not rolled back if m is *read*. In the object-based systems, request and response messages are exchanged among the objects. In addition, methods are invoked in a nested manner.

3.2 Influential messages

Suppose a method op_1^i in an object o_i invokes op_2^j in another object o_j. Figures 2 shows a pair of possible local checkpoints c_k^i and c_h^j to be taken in the objects o_i and o_j, respectively. Here, let $\pi_j(op^j, c^j)$ be a set of methods which (1) precede op^j and (2) succeed a local checkpoint c^j or are being performed at c^j in o_j. Suppose $\pi_j(op_2^j, c_1^j) = \{op_{21}^j, \ldots, op_{2l}^j\}$ in Figure 2.

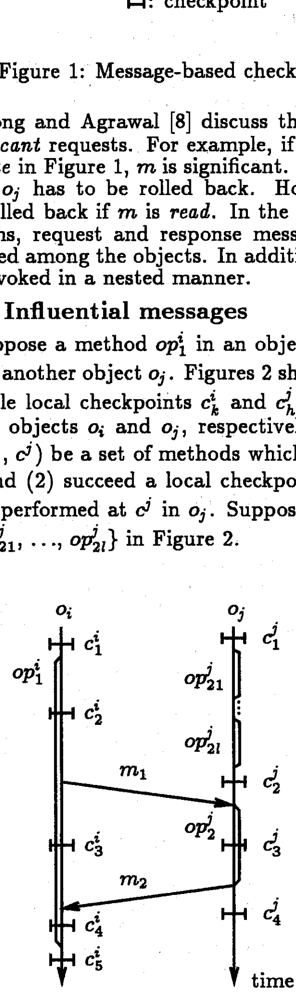

Figure 2: Possible checkpoints.

We discuss whether or not each checkpoint $\langle c_k^i, c_h^j \rangle$ can be taken in the object-based system. A checkpoint $c = \langle c^1, \ldots, c^n \rangle$ is *object-based consistent* (*O-consistent*) iff every object o_i can be rolled back to the local checkpoint c^i and then can be restarted from c^i from the object point of view. The O-consistent checkpoint c may be inconsistent with the message-based definition. For

o_i	o_j	Conditions
c_1^i	c_3^j*, c_4^j	op_2^j is non-update.
c_2^i	c_3^j*, c_4^j	op_2^j is non-update.
c_4^i c_5^i	c_1^j	op_2^j is non-update and no method in $\pi_j(op_2^j, c_1^j)$ conflicts with op_2^j.
	c_2^j, c_3^j*	op_2^j is non-update.

Table 1: O-consistent checkpoints for Figure 2.

example, the checkpoints $\langle c_1^i, c_3^j\rangle$ and $\langle c_1^i, c_4^j\rangle$ are message-based inconsistent in Figure 2. If op_2^j is non-update, the state denoted by c_2^j is the same as c_3^j and c_4^j. That is, $\langle c_1^i, c_3^j\rangle$ and $\langle c_1^i, c_4^j\rangle$ show the same state as $\langle c_1^i, c_2^j\rangle$. $\langle c_1^i, c_2^j\rangle$ 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 O-consistent. If op_2^j is update, $\langle c_1^i, c_3^j\rangle$ is not O-consistent because op_2^j has 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 performed. op_2^j is required to be undone, i.e. aborted while other methods being performed at c_3^j may not have to be undone.

There are two kinds of checkpoints, i.e. *complete* and *incomplete* ones. Suppose the object o_j is rolled back to the O-consistent 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 methods being performed at c_h^j have to be undone. However, no method invoked by the methods is required to be rolled back since the methods are non-update. Hence, $\langle c_1^i, c_3^j\rangle$ is O-consistent 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 O-consistent.

Let us consider checkpoints $\langle c_4^i, c_1^j\rangle$, $\langle c_4^i, c_2^j\rangle$, and $\langle c_4^i, c_3^j\rangle$ which are message-based inconsistent. If op_2^j is non-update, 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 O-consistent since $\langle c_4^i, c_4^j\rangle$ is message-based consistent. Here, suppose that op_2^j is *read* and there is some *write* op_{2h}^j preceding op_2^j and following 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 method 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 performed before op_{21}^j. Hence, if op_2^j is non-update and no method in the set of requests $\pi_j(op_2^j, c_1^i)$ conflicts with op_2^j, $\langle c_4^i, c_1^j\rangle$ is O-consistent. $\langle c_4^i, c_2^j\rangle$ is also O-consistent. $\langle c_4^i, c_3^j\rangle$ is O-consistent where c_3^j is incomplete.

The checkpoint including a local checkpoint c_5^i is similarly discussed. The checkpoints $\langle c_5^i, c_1^j\rangle$, $\langle c_5^i, c_2^j\rangle$, and $\langle c_5^i, c_3^j\rangle$ are also O-consistent.

Table 1 summarizes the message-based inconsistent but O-consistent checkpoints, where checkpoints marked * are incomplete if op_2^j is being performed.

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

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

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

2. If m is a response message, $Op(m)$ $(= op_2^j)$ is update or some method in $\pi_i(Op(m), c^j)$ conflicts with $Op(m)$.

If a method op_i is undone, only methods receiving influential messages from op_i are required to be undone.

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

For example, if op_2^j is update in Figure 2, the checkpoint $\langle c_4^i, c_3^j\rangle$ is not O-consistent since m_2 is influential. Otherwise, $\langle c_4^i, c_3^j\rangle$ is O-consistent. The O-consistent checkpoints may be inconsistent with the message-based definition. However, the objects can be rolled back to and be restarted from the O-consistent checkpoints.

4 Checkpointing Protocol

We discuss an asynchronous protocol for taking checkpoints among objects. In the asynchronous protocol discussed by Higaki [5], each object o_i initially takes a local checkpoint c_0^i. Here, a checkpoint $\langle c_0^1, \ldots, c_0^n\rangle$ is consistent. After taking a local checkpoint c_{t-1}^i, o_i autonomously takes a succeeding local checkpoint c_t^i. Then, o_i sends a message m marked *checkpointed* to another object o_j. Here, suppose that a local checkpoint c_{u-1}^j is taken in o_j and $\langle c_{t-1}^i, c_{u-1}^j\rangle$ is consistent. On receipt of m, the object o_j takes a local checkpoint c_u^j which saves the state of o_j which is just before receiving m. Then, suppose o_j sends a message m to another object o_k. m is marked *checkpointed*. o_k takes a checkpoint in the same way as o_j. In the object-based checkpointing, o_j does not take a local checkpoint c_u^j if $\langle c_t^i, c_{u-1}^j\rangle$ is O-consistent. We discuss how o_j decides if $\langle c_t^i, c_{u-1}^j\rangle$ is O-consistent.

Each time an object o_i sends a message m, a sequence number sq is incremented by one. In addition, a subsequence number ssq_j is incremented by one if m is sent to an object o_j. The message m carries the sequence number $m.sq$ and the subsequence numbers $m.ssq = \langle m.ssq_1, \ldots, m.ssq_n\rangle$. An object o_j manipulates variables $rsq_1, \ldots, rsq_n$ and $rssq_1, \ldots, rssq_n$ to receive messages. On receipt of m from o_i, o_j receives m if $m.ssq_j = rssq_i$. Then, $rssq_i := rssq_i + 1$ and $rsq_i := m.sq + 1$.

If o_i takes a local checkpoint c_t^i, the checkpoint number cp_i is incremented by one, i.e. $t = cp_i$. The message m which o_i sends to o_j after taking

c_t^i carries the checkpoint numbers $m.cp = \langle m.cp_1, \ldots, m.cp_n \rangle$ where $m.cp_i = cp_i$. m also carries the receipt sequence number $m.rq = \langle m.rq_1, \ldots, m.rq_n \rangle$ where $m.rq_k = rsq_k$ ($k = 1, \ldots, n$). Here, $m.rq_k$ shows a sequence number of messages which o_i receives from o_j just before taking the local checkpoint c^i.

On receipt of m from o_i, the object o_j finds a set M_j of messages $m_{j_1}, \ldots, m_{j_{l_j}}$ which o_j had sent to o_i after taking the current local checkpoint c_{u-1}^j. Here, $m_{j_h}.sq$ is smaller than $m.rq_j$ [Figure 3]. The messages are requests or responses. A message m_{j_h} in M_j is influential if the following conditions hold:

1. If m_{j_h} is a request, $m_{j_h}.op$ is update.
2. If m_{j_h} is a response, $m_{j_h}.op$ is update or conflicts with some update method in $\pi_j(m_{j_h}.op, c_{u-1}^j)$.

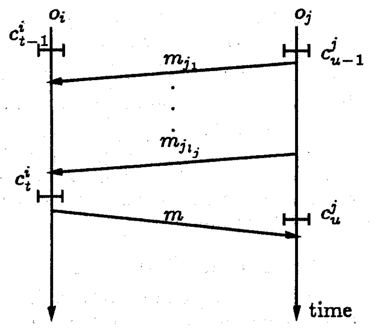

Figure 3: Checkpoints

If at least one of the messages $m_{j_1}, \ldots, m_{j_{l_j}}$ is influential, the object o_j takes a local checkpoint c_u^j showing a state of just before receiving m. Otherwise, o_j does not takes a local checkpoint. If o_j takes a checkpoint c_u^j, the checkpoint number cp is incremented by one in o_j. Messages sent by o_j after c_u^j are marked *checkpointed*.

Suppose that there are three objects o_i, o_j, and o_k. The object o_i initiates the checkpoint procedure. o_i sends a checkpointed message m_i marked to o_j after taking a local checkpoint c^i. o_j takes a checkpoint c^j on receipt on m_i. Then, o_j sends a checkpointed message m_j to o_k. On receipt of m_j, o_k takes a checkpoint c^k. Then, o_k sends a checkpointed message m_k to o_i. o_i does not need to take a checkpoint because $\langle c^i, c^j, c^k \rangle$ is a consistent checkpoint. If o_i takes a checkpoint here, the object o_i, o_j, and o_k cannot stop the checkpoint procedure. In order to resolve this problem, o_i sends a checkpointed message with the checkpoint numbers. Each c_t^i taken by o_i has a sequence number $no(c_t^i)$. Each time o_i takes a local checkpoint, o_i increments a checkpoint number by one. Here, a variable cp_i shows a checkpoint number of o_i. cp_j shows a checkpoint number of o_j which o_i knows. On receipt of a checkpointed message m from o_j, $cp_j := m.cp_j$ in o_i if $cp_j < m.cp_j$. Then, o_i sends a checkpointed message with $cp = \langle cp_i, \ldots, cp_n \rangle$ after taking a local checkpoint. If $cp_j > m.cp_j$, o_j does not take a checkpoint because o_j has already taken a checkpoint initiated by o_j.

5 Evaluation

We evaluate the object-based consistent (O-consistent) checkpoint protocol in terms of the numbers of O-consistent checkpoints and influential messages. We make the simulation on the following client-server environment:

1. There are n server objects $o_1, \ldots, o_n$ in the system.
2. One client object initiates transactions, possibly concurrently. Each transaction issues randomly one method to the server object.
3. Each method invokes randomly methods in other objects. The maximum level of invocation is three. The level is randomly decided when a transaction invokes the method.
4. Every pair of non-update methods are compatible but every update method conflicts with every other method.
5. One server object, say o_1, initiates the checkpoint procedure every time some number cn of methods are computed.

Here, let P_s denote a probability that a method invoked by o_i is non-update. Let $C_N(n, P_s, cn)$ and $C_O(n, P_s, cn)$ be the numbers of local checkpoints taken in the traditional way and the O-consistent checkpoint, respectively, for the number n of server objects, the probability P_s of the non-update methods, and the checkpoint frequency cn. Let $M_N(n, P_s, cn)$ and $M_O(n, P_s, cn)$ be the numbers of messages transmitted in the traditional way and the O-consistent checkpoint, respectively, for n, P_s, and cn.

In the simulation, the client object issues 800 methods. In Figure 4, the straight line shows the ratios $C_O(n, 0.5, 2)/C_N(n, 0.5, 2)$ and the dotted line indicates $M_O(n, 0.5, 2)/M_N(n, 0.5, 2)$ for n given $P_s = 0.5$ and $cn = 2$. That is, 50% of methods invoked are non-update. The checkpoint procedure is initiated each time every two methods are invoked in o_1. Figure 4 shows that the number of checkpoints to be taken can be reduced by taking only the object-based consistent (O-consistent) checkpoints. For example, only 60% of traditional checkpoints are taken in the O-consistent checkpoint if there are seven server objects, i.e. $n = 7$.

In Figure 5, the straight line shows $C_O(5, P_s, 2)/C_N(5, P_s, 2)$ and the dotted line shows $M_O(5, P_s, 2)/M_N(5, P_s, 2)$ for P_s, $n = 5$, and $cn = 2$. The more non-update methods are invoked, the fewer number of influential messages are transmitted and the fewer number of checkpoints are taken in the O-consistent checkpoint.

Figure 6 shows $C_O(10, 0.8, cn)/C_N(10, 0.8, cn)$ and $M_O(10, 0.8, cn)/M_N(10, 0.8, cn)$ for cn given $n = 10$ and $P_s = 0.8$. That is, 80% of the methods are non-update. Figure 6 shows that the number of checkpoints taken by the server objects are not increased even if the checkpoint procedure is more often initiated. This means that the objects which are required to be more available can often initiate the checkpoint procedure.

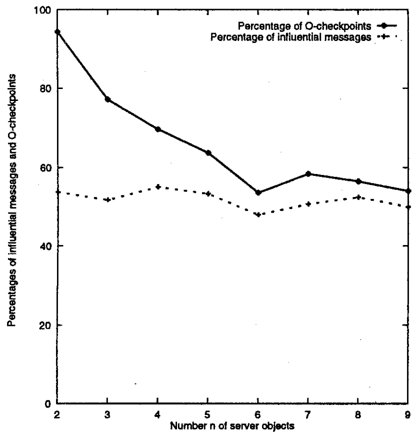

Figure 4: O-consistent checkpoints and influential messages for n.

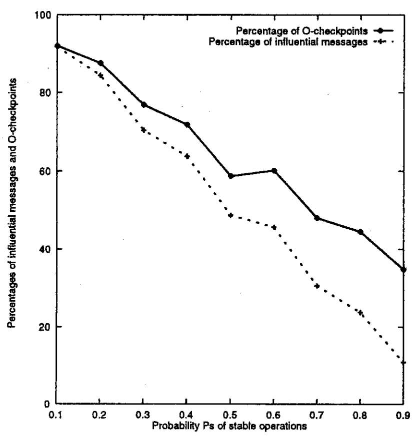

Figure 5: O-consistent checkpoints and influential messages for P_s ($n = 5$).

6 Concluding Remarks

We have discussed how to take *object-based consistent* (*O-consistent*) checkpoints which can be taken from the application point of view but may be inconsistent with the traditional message-based definition. We have defined *influential messages* on the basis of the semantics of requests and responses where the methods are nested. Only objects receiving influential messages are rolled back if the senders of the influential messages are rolled back. The *O-consistent checkpoint* is one where

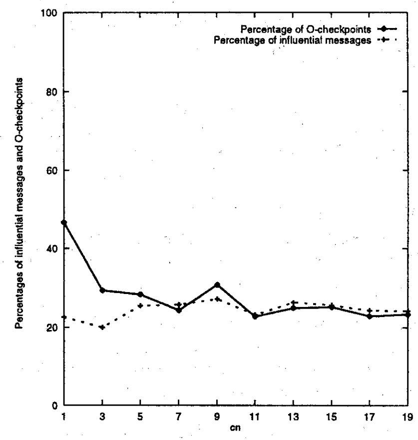

Figure 6: O-consistent checkpoints and influential messages for cn ($n = 10$).

there is no orphan influential message. We have presented the asynchronous protocol for taking O-consistent checkpoints where every method is synchronously invoked. By the evaluation, we have shown fewer number of checkpoints are taken in the O-consistent checkpoint than the message-based checkpoint. We have shown how much we can reduce the number of checkpoints to be taken if each object takes only O-consistent checkpoints.

References

[1] Bhargava, B. and Lian, S. R., "Independent Checkpointing and Concurrent Rollback for Recovery in Distributed Systems — An Optimistic Approach," *Proc. of IEEE SRDS-7*, pp. 3-12, 1988.

[2] Chandy, K. M. and Lamport, L., "Distributed Snapshots : Determining Global States of Distributed Systems," *ACM Trans. on Computer Systems*, Vol. 3, No. 1, pp. 63–75, 1985.

[3] Fischer, M. J., Griffeth, N. D., and Lynch, N. A., "Global States of a Distributed System," *IEEE Trans. on Software Engineering*, Vol. SE-8, No. 3, pp.198–202, 1982.

[4] Garcia-Molina, H., "Using Semantics Knowledge for Transaction Processing in a Distributed Database," *Proc. of ACM SIGMOD*, Vol. 8, No. 2, pp. 188-213, 1983.

[5] Higaki, H., Sima, K., Tanaka, K., Tachikawa, T. and Takizawa, M., "Checkpoint and Rollback in Asynchronous Distributed Systems," *Proc. of The 16th IEEE INFOCOM*, pp. 1000–1007, 1997.

[6] Koo, R. and Toueg, S., "Checkpointing and Rollback-Recovery for Distributed Systems," *IEEE Trans. on Computers*, Vol. C-13, No. 1, pp. 23–31, 1987.

[7] Lin, L. and Ahamad, M., "Checkpointing and Rollback-Recovery in Distributed Object Based Systems," *Proc. of IEEE SRDS-9*, pp. 97–104, 1990.

[8] Leong, H. V. and Agrawal, D., "Using Message Semantics to Reduce Rollback in Optimistic Message Logging Recovery Schemes," *Proc. of IEEE ICDCS-14*, pp.227–234, 1994.

[9] Manivannan, D. and Singhal, M., "A Low-Overhead Recovery Technique Using Quasi-Synchronous Checkpointing," *Proc. of IEEE ICDCS-16*, pp.100–107, 1996.

[10] Ramanathan, P. and Shin K. G., "Checkpointing and Rollback Recovery in a Distributed System Using Common Time Base," *Proc. of IEEE SRDS-7*, pp. 13–21, 1988.

[11] Tachikawa, T. and Takizawa, M., "Communication Protocol for Group of Distributed Objects," *Proc. of IEEE ICPADS'96*, pp. 370–377, 1996.

[12] Tanaka, K. and Takizawa, M., "Distributed Checkpointing Based on Influential Messages," *Proc. of IEEE ICPADS'96*, pp. 440–447, 1996.

[13] Tanaka, K., Higaki, H., and Takizawa, M., "Object-Based Checkpoints in Distributed Systems," *Journal of Computer Systems Science and Engineering*, Vol. 13, No.3, May 1998, 125–131.

[14] Wang, Y. M. and Fuchs, W. K., "Optimistic Message Logging for Independent Checkpointing in Message-Passing Systems," *Proc. of IEEE SRDS-11*, pp. 147-154, 1992.

[15] Weikum, G., "Principles and Realization Strategies of Multilevel Transaction Management," *ACM TODS*, Vol. 16, No. 1, pp.132-180, 1991.