분산 트랜잭션 처리 시스템에서 2-단계 확인 프로토콜을 근거로 하는 검사점 설정 및 오류 복구 알고리즘

박 윤 용[†] · 전 성 익[††] · 조 주 현[†††]

요 약

본 논문은 분산 트랜잭션 처리 시스템에서 분산 트랜잭션들이 사용한 자원들을 일관성 있게 유지하는 검사점 설정 및 오류 복구 기법에 관한 연구이다. 기존의 방법과 비교하여 제안하는 검사점 설정 방법은 검사점을 설정하는 동안 수행되고 있는 분산 트랜잭션들에 간섭 현상과 저장 비용을 최소화 할 수 있고, 검사점을 설정하기 위한 별도의 메시지를 사용하지 않기 때문에 추가의 검사점을 설정하기 위한 메세지 비용이 없다. 또한 제안하는 알고리즘은 도미노 현상과 순환적 재시작 현상을 제거할 수 있다. 본 논문에서는 제안하는 알고리즘의 정확성과 성능을 설명하였다.

A Checkpointing and Error Recovery Algorithm Based on 2-Phase Commit Protocol for Distributed Transaction

Yoon-Young Park[†] · Soung-Ik Jun[††] · Ju-Hyun Cho[†††]

ABSTRACT

In this paper, we present a new checkpointing algorithm to preserve the consistency of resources in distributed transaction processing systems, and the error recovery algorithms to recover from the failure. In comparison with the existed algorithms, the checkpointing algorithm proposed in this paper can minimize the interference of the distributed transaction and the stroage cost during checkpointing, and does not need the extra message to make the checkpoint. Also we show that the error recovery algorithms prevent the distributed transaction with a partial fault from spreading the fault, which calls domnio-effect and prevent them from restarting cyclically. And we describe the correctness and the performane of the proposed algorithms.

1. 서 론

본 논문은 분산 트랜잭션 처리(DTP:Distributed Transaction Processing) 시스템에서 분산 트랜잭션들의 사용한 자원들의 일관성을 유지하는 검사점(checkpoint) 설정 기법과 고장(failure) 발생시 복구하는 기법에 관한 연구이다. 기존의 DTP 시스템은 오류나 원하지 않는 데이타를 탐지 할 수 있는 강력한 데이타 베이스 무결성 검사 기법이 존재하지만 오류를 내

† 정 회 원:선문대학교 전자계산학과 조교수, Computer Science Dept., SunMoon University.
†† 정 회 원:한국 전자 통신 연구소 선임 연구원, 실시간 운영 체제 연구실
††† 정 회 원:한국 전자 통신 연구소 책임 연구원, 실시간 운영 체제 연구실
논문접수:1995년 10월 10일, 심사완료:1995년 12월 7일

재하고 있는 데이타가 존재할 가능성이 있다. 또한 분산 시스템의 각 사이트에 존재하는 하드웨어 또는 소프트웨어의 고장은 데이타 베이스의 일관성을 파괴할 수 있다. DTP 시스템에서는 이러한 오류와 고장으로부터 데이타들을 일관성 있게 유지하기 위한 다양한 복구 기법을 제공하고 있으며, 검사점은 복구 기법중에서 가장 널리 사용되는 방법이다.

검사점의 목적은 일관성이 보장된 데이타 베이스의 상태를 안정 기억 장치에 저장시키고, 고장이 발생한 경우에 저장된 데이타를 사용하여 데이타 베이스를 복구하게 하는 것이다. DTP시스템에서 검사점을 설정하는데 일반적으로 고려해야 할 사항들은 다음과 같다. 첫째, 분산 시스템 전체에 일관성이 유지되도록 검사점이 설정되어야 한다. 일관성이 유지되지 않는 검사점은 고장시 복구하는 비용을 증가시키고 최악의 경우 도미노 효과등이 발생하여 전체 시스템을 고장 상태로 만들 수 있다.

둘째로 분산 시스템에서 검사점을 설정하기 위한 통신 비용과 검사점의 내용을 저장하기 위한 저장 공간이 최소화 되도록 검사점을 설정하여야 한다. 또한 검사점은 시스템이 정상적인 작업 중에 수행되어 지기 때문에, 현재 실행되고 있는 트랜잭션에 대한 간섭(interference)이 최소화되도록 검사점을 설정해야 한다[1]. 이것은 검사점이 실행되고 있는 중에도 사용자들이 트랜잭션을 시스템에 제출 할 수 있고, 검사점이 설정되는 작업과 병행하여 트랜잭션이 실행될 수 있기 때문에 중요하게 고려되어야 한다. 특히 DTP 시스템에서는 트랜잭션들이 시스템 내의 독립적인 사이트들 사이에서 협동(coordination)하여 실행되기 때문에 간섭이 최소화되고 일관성이 보장된 검사점을 설정하는 것을 더욱 어렵게 한다.

기존의 분산 시스템에서 사용되는 검사점 설정 방법은 동기화된 방법과 비동기화 된 검사점 설정 방법으로 구분된다. 기존의 동기적(synchronous) 검사점 설정 기법들은 각 사이트에서 검사점을 설정하는 시점에 다른 사이트의 검사점들과 완전 동기화를 이루도록 검사점을 설정한다. 따라서 고장이 발생한 경우에 빠른 시간에 일관성이 보장된 검사점을 찾아서 복구를 쉽게 할 수 있다. 그러나 이러한 방법은 검사점을 설정하는 시간이 많이 걸리고, 분산 트랜잭션들의 수행에 간섭 현상을 초래하게 되는 단점이 있다.

비동기적인(asynchronous) 검사점 기법들은 검사점을 각 사이트에서 독립적으로 설정함으로써 쉽게 검사점을 설정 할 수 있지만 고장이 발생하는 경우 일관성이 보장된 검사점을 찾아서 복구하는 작업을 어렵게 한다. 이러한 기법들은 최악의 경우 일관성이 보장된 검사점을 찾기 위해 시스템 초기 시점까지 복귀 작업을 하는 도미노 현상(domino-effect)을 발생하게 한다[2, 3].

본 논문에서는 DTP 시스템에서 일반적으로 사용되고 있는 2-단계 확인(2-PC:2-Phase Commit) 프로토콜을 이용하여 추가의 부담(overhead) 없이 일관성이 보장되는 검사점을 설정 할 수 있는 알고리즘을 제시하였다. 제안하는 알고리즘은 기존의 방법에 비해 구현이 간단하고, 현재 실행 중인 트랜잭션들의 실행에 간섭 현상이 없으며, 도미노 현상을 완벽히 제거할 수 있다. 또한 제안하는 검사점 설정 알고리즘은 기존의 2-PC 프로토콜 자체에서 사용하는 메시지 만을 사용하여 검사점을 설정할 수 있기 때문에 추가의 메시지 부담이 없다. 고장이 발생한 경우에도 전역적으로 유지되는 가장 최근의 검사점으로 복귀함으로서 철회 비용을 최소화하였다. 이상의 내용을 정리하면 〈표 1〉과 같다[1, 4, 5, 6, 7, 8, 9, 10].

〈표 1〉 검사점 기법의 비교
〈Table 1〉 Comparision of Checkpointing Method

	동기적 기 법	비동기적 기 법	제안하는 기 법
통신 비용	과도	없음	없음
복구 알고리즘	간단	복잡	간단
검사점의 일관성 유지	유지	유지못함	유지
간섭 비용	과도	없음	없음
저장 장치 비용	동기화가 보장된 검사점까지 저장	시간에 따라 증가	두개의 검사점 내용만 저장
도미노 현상	제거(가능)	발생	제거
순환적 재시작 현 상	발생	발생	제거

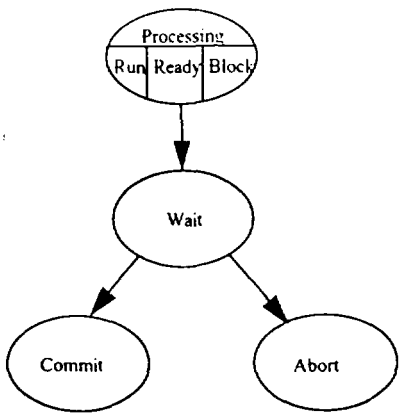
본 논문의 구성은 다음과 같다. 2장에서는 시스템 모델및 기본 가정을 기술하였고, 3장에서는 제안하는

알고리즘의 기본 아이디어 및 사용되는 자료 구조를 설명하였다. 4장에서는 검사점 설정 및 복구 알고리즘을 제시하였고, 5장에서는 알고리즘의 정확성에 관하여 설명하였다. 6장에서는 제시하는 알고리즘의 성능을 간략히 기술하였다.

2. 시스템 모델 및 기본 가정

분산 트랜잭션은 조정자 트랜잭션과 참여자 트랜잭션으로 구분되어 수행되고, 조정자 트랜잭션은 각 사이트에 존재하는 원격 자원을 요청하기 위하여 참여자 트랜잭션을 생성하고 참여자 트랜잭션의 수행에 원자성을 보장하는 작업을 한다. 일반적으로 분산처리 시스템에서는 트랜잭션의 원자성을 보장하기 위하여 2-단계 확인(2-PC: 2-Phase Commit) 프로토콜을 사용하고 있다. 이와 같은 트랜잭션의 원자성 완료 프로토콜의 성능과 신뢰도를 향상시키기 위한 연구가 있었다[11, 12, 13].

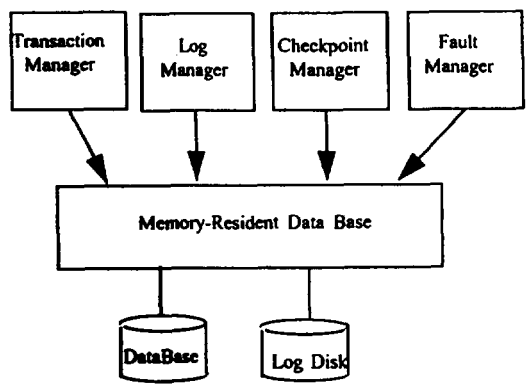
임의의 분산 트랜잭션 T_i는 (그림 1)과 같은 상태 변화를 한다. 처리 상태(processing state)는 트랜잭션 T_i의 조정자 트랜잭션과 임의의 참여자 트랜잭션들이 수행(run), 수행 대기(ready), 블럭(block) 상태임을 나타낸다. 대기 상태(wait state)는 T_i의 참여자 트랜잭션들이 자신의 사이트에서 수행을 종료하고, 다른 사이트에 있는 T_i의 모든 참여자 트랜잭션들과 전역적인 완료 또는 취소등의 동기화된 결정을 기다리는 상태이다.



(그림 1) 트랜잭션의 상태 천이도
(Fig. 1) State Transition of Transaction

완료 상태(commit state)는 2-PC 프로토콜에 의해서 트랜잭션 T_i의 모든 참여자 트랜잭션의 수행이 완료된 상태이고, 취소 상태(abort state)는 T_i의 임의의 참여자 트랜잭션이 취소되어 T_i가 취소되는 상태이다. 이와 같은 트랜잭션의 상태는 시스템에서 고장이 발생하는 경우에 복구하는 방식을 결정하는 중요한 요소가 된다. 일반적으로 고장으로부터 복구시에 완료된 트랜잭션인 경우는 수행한 내용을 재수행(redo)하게 되고, 취소된 트랜잭션인 경우는 철회(undo) 작업을 통하여 복구 작업을 수행한다.

본 논문에서 고려하고 있는 분산 시스템은 각 사이트가 (그림 2)과 같은 구조를 가지고, 디스크 접근 시간을 감소하기 위하여 대부분의 데이타 베이스 내용을 주 기억 장치에 저장시키는 메모리 상주 데이타 베이스 시스템(memory resident database system)을 사용한다. 로그 관리자(log manager)는 지역 데이타 베이스의 자원에 대한 트랜잭션들이 수행한 연산들의 로그를 유지하고 관리하며, 로그의 효율적인 저장을 위하여 고 성능의 로그 전용 디스크를 사용한다. 검사점 관리자(checkpoint manager)는 메모리에 존재하는 데이타 베이스의 자원들 중에서 완전한 계산 결과를 일정한 절차에 따라서 안정 기억 장치(stable stroage)에 저장하는 역활을 한다.



(그림 2) 시스템 구조
(Fig. 2) System Architecture

고장 관리자(fault manager)는 사이트 x에서 고장이 발생된 경우 로그 관리자에 의해 저장된 연산 로그와 검사점 관리자에 의해 저장된 가장 최근의 검사점의

내용을 이용하여 시스템의 상태를 복구하게 된다. 특히 DTP 시스템의 대부분의 응용 분야에서는 고도의 시스템 가용성(availability)을 보장해야 하기 때문에 고장 상태에서의 빠른 복구가 중요하다. 이와 같은 빠른 복구를 하기 위해서는 검사점이 분산 시스템 전체에 일관성 있게 설정되어야 하고, 각 검사점의 일관성을 보장하기 위해서는 임의의 트랜잭션의 실행 결과가 하나의 검사점에 완전히 포함되거나 또는 전혀 포함되지 않아야 한다. 이와 같이 트랜잭션들이 사용한 자원 중에서 완료된 내용만을 저장하는 검사점을 완료 검사점(CCP:Commit CheckPoint)이라 한다.

본 논문에서는 다음과 같은 기본적인 가정을 사용하였다.

(가정 1) 병행 제어 기법은 기본적으로 2 단계 로킹 기법을 사용한다.

(가정 2) 완료 교착 상태(commit deadlock, global deadlock)는 2-PC 프로토콜에서 제공되는 타임아웃을 사용하여 해결한다[12, 13].

(가정 3) 메시지는 손실없이, 순서적으로, 기한부 시간(deadline time)이내에 전송된다.

3. 기본 개념 및 자료 구조

임의의 사이트 x에서 지역 검사점 번호(LCPN:Local CheckPoint Number)가 n이 되는 완료 검사점을 $CCP_x(n)$으로 표기한다. 또한 $CCP_x(n-1)$과 $CCP_x(n)$ 사이의 시간 간격을 검사점 설정 간격(checkpoint interval)이라 정의하고, 기본적으로 시스템이 정의한 시간 간격인 δ 간격으로 각 사이트에서 다른 사이트와 동기화 과정 없이 지역적으로 완료 검사점 CCP를 설정한다. 사이트 x에서 지역 검사점 번호가 n인 검사점까지 설정되었다면, 사이트 x의 현재 완료 검사점은 $CURRENT_CP_x = CCP_x(n)$으로 표기하고, 사이트 x의 현재 지역 검사점 번호는 $LCPN_x = n$ 으로 표기하겠다.

본 논문에서 제안하는 검사점 설정 알고리즘은 DTP 시스템 내의 임의의 두 사이트의 현재 검사점 번호의 차가 다음의 (식-1)과 같이 1을 넘지 않도록 검사점 설정 간격 δ를 정의한다고 가정한다(단, x, y는 시스템 내의 임의의 사이트).

$$| LCPN_x - LCPNy | \leq 1; \qquad \text{(식-1)}$$

그러므로 일관성이 보장되는 검사점을 설정하는 문제는 분산 시스템의 임의의 사이트 x에서 검사점을 설정할 때 분산 트랜잭션들이 현재 완료 검사점, $CURRENT\_CP_x = CCP_x(n)$, 이후에 실행한 연산 내용을 $CCP_x(n+1)$에 포함시킬 것 인가를 결정하는 문제로 전환된다. 즉 일관성을 보장하는 검사점 설정 문제는 j개의 사이트에 분산된 참여자 트랜잭션 T_{i1}, T_{i2}, ..., T_{ij}로 구성된 분산 트랜잭션 T_i의 전역 검사점 번호(GCPN:Global CheckPoint Number)를 결정하는 문제로 단순히 전환된다.

$LCPN_x = n$이 되는 사이트 x에서 실행되고 있는 임의의 트랜잭션 T_i의 참여자 트랜잭션 T_{ix}의 LCPN은 다음의 (식-2)와 같이 결정된다. (식-2)에서 $CCP_x(n)$은 이미 검사점 작업이 완료된 것을 의미하고, 참여자 트랜잭션 T_{ix}가 실행한 내용이 저장될 검사점은 $CCP_x(n+1)$이 되기 때문에 $LCPN_x$의 값에 1을 더하게 된다.

$$LCPN_x(T_{ix}) = LCPN_x + 1; \qquad \text{(식-2)}$$

트랜잭션 T_i의 GCPN은 아래의 (식-3)과 같이 결정된다. $GCPN(T_i)$를 결정하는 것은 기존의 원자성 완료 프로토콜인 2-PC 프로토콜을 이용하여 추가의 메시지 부담없이 결정할 수 있다[8, 9].

$$GCPN(T_i) = \max(LCPN_k(T_{ik})) \text{ for } \forall k \in T_i\text{의 모든 참여자 트랜잭션}; \qquad \text{(식-3)}$$

본 논문에서 기술하는 2-PC 프로토콜은 일반적인 2-PC 프로토콜과 동일하며 다만 조정자 트랜잭션이 참여자 트랜잭션들의 수행 내용을 투표하는 과정에서 GCPN을 결정하는 작업만이 추가되었다. 본 논문에서 사용되는 2-PC 프로토콜 중에서 트랜잭션 T_i의 GCPN을 결정하는 알고리즘을 다음과 같이 기술하였다.

알고리즘 1:GCPN 결정 알고리즘

(step 1) for each T_{ij}

(step 1.1) send (T_{ij}, "Start 2-PC");

(step 1.2) receive ($LCPN_j(T_{ij})$, "State of T_{ij}");
(step 1.3) if ("State of T_{ij}" == ABORT)
global state of T_i = ABORT;
break;
endif
(step 1.4) global state of T_i = COMMIT;
(step 1.5) $GCPN(T_i) = \max(LCPN_j(T_{ij}))$;
endfor
(step 2) for each T_{ij}
send (T_{ij}, "global state of T_i", $GCPN(T_i)$);
endfor

알고리즘 1의 (step 1.1)에서는 조정자 트랜잭션 T_{i0}가 일정 기간 후에 2-PC을 시작하기위한 메시지를 각 참여자 트랜잭션 T_{ij}에게 전송한다. (step 1.2)에서는 각 참여자 트랜잭션 T_{ij}가 자신의 실행 상태(완료 또는 취소)와 $LCPN_j(T_{ij})$을 함께 T_{i0}에게 전송한다. 이와 같은 send, receive 프리미티브의 실제 구현 방법은 시스템에 따라 다를 수 있으며, 본 논문에서는 간략한 표기 방법으로 기술하였다.

(step 1.4)에서 조정자 트랜잭션 T_{i0}는 참여자 트랜잭션들의 상태에 따라서 전역적인 완료 또는 취소 결정을 하고, 참여자 트랜잭션 중에 하나라도 취소메시지를 보내면 트랜잭션 T_i는 즉시 취소 상태로 전환된다 (step 1.3). (step 1.5)에서 참여자 트랜잭션들의 $LCPN_j(T_{ij})$중에서 가장 큰 값을 현재 트랜잭션 T_i의 GCPN(T_i)로 한다. (step 2)는 트랜잭션 T_i의 조정자 트랜잭션은 트랜잭션의 전역적인 상태(완료 또는 취소)와 GCPN을 각 참여자 트랜잭션에게 전송한다.

본 논문에서 사용되는 로그 메시지는 (그림 3)과 같은 형태를 갖고, 로그 메시지의 상태와 GCPN은 관련된 트랜잭션의 상태와 트랜잭션의 GCPN과 일치한다. 시스템 내의 임의의 사이트에서 $LCPN_x$가 n이고, n 번째 완료 검사점 $CCP_x(n)$까지 설정되어 있는 경우 연산 로그들의 형태는 다음과 같은 4가지 유형으로 분류할 수 있다.

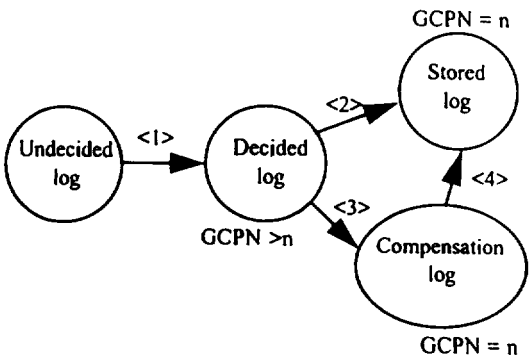
(1) 미 결정 로그(undecided log): 연산 로그의 상태와 GCPN이 아직 결정되지 않은 상태의 로그.
(2) 결정 로그(decided log): 로그의 상태와 GCPN이 결정되어 있고, 로그의 GCPN이 $LCPN_x$보다 큰 경우의 로그.
(3) 저장 로그(stored log): 로그의 상태와 GCPN이 결정되어 있고, 로그의 GCPN이 $LCPN_x$와 같은 로그로서 $CCP_x(n)$에 이미 저장되어져 있는 로그.
(4) 보상 로그(compensation log): 로그의 상태와 GCPN이 결정되어 있고, 로그의 GCPN이 $LCPN_x$와 같은 로그로서 $CCP_x(n)$에 저장되지 못한 로그.

LSN	Tid	Rid	Status	GCPN	Operation	Current Value

LSN : Log Sequence Number
Tid : Transaction Identification
Rid : Resource Identification
Status : Status of Transaction
GCPN : Global Checkpoint Number

(그림 3) 로그 메시지 형식
(Fig. 3) Log Message Format

(그림 4)는 사이트 x에서 $LCPN_x = n$이고, $CURRENT\_CP_x = CCP_x(n)$ 인 경우에 로그들의 상태와 상태 전이도이다. 일반적으로 시스템의 로그들은 2-PC 프로토콜에 의해 로그의 상태와 GCPN이 결정되기 전까지 미 결정 로그 상태로 존재하다가 2-PC 프로토콜이 완료되는 순간 로그의 상태와 GCPN이 결정되게 된다. 이러한 경우에 로그의 GCPN과 $LCPN_x$와의 값에 따라 (그림 4)와 같이 구분된다.

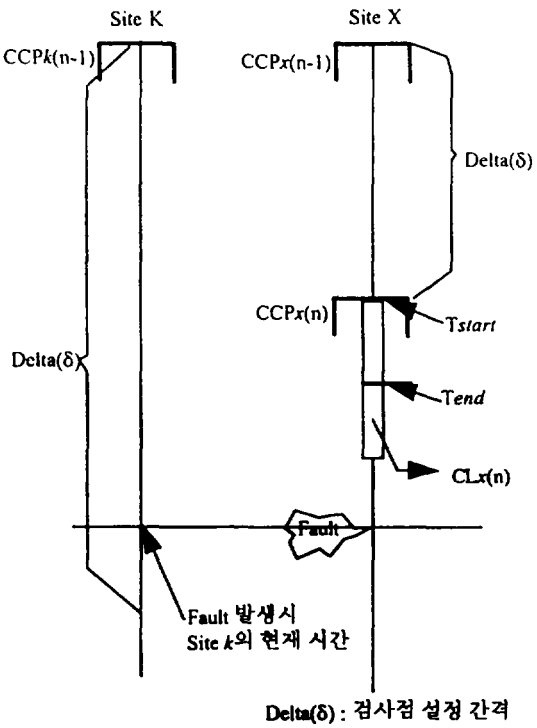


GCPN : GCPN of log
LCPNx = n, Current CPx = CCPx(n)

(그림 4) 로그의 상태 천이도
(Fig. 4) State Transition of Log

일반적으로 분산 시스템에서는 모든 사이트의 시간을 일치시키는 전역적인 클럭이 존재하기 어렵고, 메시지 전송 지연(delay) 등의 원인으로 인하여 임의의 사이트 x에서 새로운 CCP를 설정하고자 하는 시점에 로그의 상태와 로그의 GCPN이 아직 결정되지 않은 연산 로그들이 존재할 수 있다.

즉, (그림 5)에서와 같이 사이트 x에서 n 번째 새로운 완료 검사점 CCPx(n)을 설정하고자 하는 시점, T_{start},에서 아직 로그의 상태와 GCPN이 결정되지 못한 로그들이 있을 수 있다. 또한 n 번째 새로운 검사점 작업이 완료되어 현재 지역 검사점 번호가 $LCPN_x = n$ 으로 되어진 시점, T_{end}, 이 후에도 메시지 전송상의 지연 때문에 (그림 5)의 빗줄친 부분에서도 연산 로그의 GCPN이 n이 되는 로그들이 존재 할 수 있다. 본 논문에서는 이러한 로그를 보상 로그(CL:Compensation Log)라 정의하고, n 번째 완료 검사점 $CCP_x(n)$에 저장되지 못한 보상 로그들을 $CL_x(n)$으로 표기하겠다



(그림 5) 검사점 설정 및 복구 방법
(Fig. 5) Checkpoint and Recovery

4. 검사점 설정 알고리즘

본 논문의 기술을 간략히 하기 위하여 분산 시스템의 각 사이트에서 설정되는 LCPN 값의 차는 (식-1)에서와 같이 1을 초과하지 않을 만큼 검사점 설정 간격 δ가 충분히 크다고 가정한다. 또한 (가정 3)과 같이 신뢰성 있는 네크워크를 사용하여 네트워크 자체의 고장이나 메시지의 손실도 없다고 가정하였다.

본 논문에서 제안하는 알고리즘은 분산 시스템의 각 사이트에서 δ 간격으로 비동기적으로 완료 검사점을 설정하도록 한다. n+1 번째 새로운 완료 검사점을 설정하는 작업은 $CCP_x(n)$ 이후 검사점 설정 간격 동안 발생된 연산 로그들을 새로운 $CCP_x(n+1)$에 저장하는 작업으로 정의한다. 새로이 설정되는 $CCP_x(n+1)$에 저장될 연산 로그는 로그의 상태가 완료 상태이고 로그의 GCPN이 새로이 설정되는 $CCP_x(n+1)$의 $LCPN_x = n+1$과 같은 연산 로그들이 된다.

검사점은 주로 연산 로그를 중심으로 하여 로그의 상태와 GCPN을 이용하여 로그의 내용을 안정 저장 장치에 적용하는 과정으로 이루어 진다. $LCPN_x = n$인 사이트 x에서 (n+1) 번째 완료 검사점 $CCP_x(n+1)$을 설정하는 알고리즘은 다음과 같다.

알고리즘 2: 검사점 설정 알고리즘

```
(step 1) increment LCPN_x;
(step 2) for each log after CCP_x(LCPN_x - 1)
(step 3)    if (status of log == COMMIT && GCPN
            of log == LCPN_x)
                apply log to CCP_x(LCPN_x);
                delete log from log disk;
(step 4)    else if (status of log == COMMIT &&
            GCPN of log < LCPN_x)
                store log to CL_x(LCPN_x - 1);
(step 5)    else if (status of log == COMMIT &&
            GCPN of log > LCPN_x)
                mark log to deceided log;
(step 6)    else if (status of log == ABORT)
                delete log from log disk;
            endif
(step 7)    mark log to undecided log;
            endfor
```

시스템에서 고장이 발생 하는 경우 복구를 위해 임의의 사이트 x에서 $CCP_x(n)$ 이 후에 발생된 연산들은 기존의 로그 관리자에 의해 유지되고 있고, 고장이 발생한 경우에도 최근의 완료 검사점 이후에 발생된 연산 로그는 고장 관리자에 의해 안정 기억 장치에 저장됨을 가정한다. $LCPN_x = n$ 인 임의의 사이트 x에서 고장이 발생된 경우에 고장 사이트에서는 가장 최근의 완료 검사점 $CCP_x(n)$과 고장 관리자에 의해 저장된 연산 로그들를 이용하여 알고리즘 3과 같이 복구 작업을 한다.

이 경우에 고장 사이트에 존재하는 연산 로그들과 관련된 사이트들도 재 수행되어야 한다. 알고리즘 3의 (step 4)에 의해 재 수행해야 할 사이트 식별명을 구하고, 고장 사이트와 관련되어 재 수행 되어야 할 사이트 집합을 $REDO_x(n)$과 같이 표기하였다. 알고리즘 3의 (step 5), (step 6)에 의해 고장 사이트와 관련된 사이트들이 재 수행 하도록 하는 메시지를 송신(send)한다. 이와 같은 경우에 같은 사이트로 반복적 또는 이행적(transitive)으로 재 수행 메시지를 보냄으로써 시스템내에 복구 작업이 반복되는 순환적 재시작(cyclic restart) 현상이 발생될 수 있다. 따라서 알고리즘 3의 (step 5)와 같이 이미 재 수행 메시지를 송신한 사이트에는 다시 메시지를 보내지 않도록 하여 시스템에서 순환적 재시작 현상이 발생하지 않도록 하였다.

또한 제안하는 알고리즘은 각 사이트에서 δ 시간 주기로 완료 검사점을 설정하고, 일반적으로 분산 시스템에서는 시간 지연이 발생 할 수 있기 때문에 (그림 5)와 같이 고장 사이트와 관련된 임의의 사이트 k에서는 아직 완료 검사점 $CCP_k(n)$이 설정되지 않은 경우가 발생할 수 있다. 본 논문에서는 이러한 사이트에서도 현재 완료 검사점 $CURRENT_CP_k = CCP_k(n-1)$까지 복귀(rollback)하도록 하였다. 그러나 알고리즘 4의 (step 4)에서와 같이 $CCP_x(n-1)$ 이후에 발생된 연산 로그 중에서 연산 로그의 $GCPN \geq n$인 로그들을 알고리즘 3과 같은 방법으로 재 수행 하여 복구 하고, 연산 로그의 $GCPN < n$이 되는 로그들은 알고리즘 4의 (step 4.3)의 과정을 생략하고 (step 4.8)의 단순 재수행 작업(simple redo)만을 실행하여 복구하였다. 본 논문에서는 이와 같은 방법으로 $LCPN_x = n$ 인 사이트에서 고장이 발생한 경우에 고장 사이트와 관련된 모든 사이트에서 n 번째 완료 검사점까지만 복귀하여 복구할 수 있도록 함으로써 시스템 전역적으로 n 번째 검사점 이전으로 복귀되는 도미노 현상을 제거하였다.

본 논문에서는 복구 알고리즘을 고장 사이트에서의 복구 알고리즘과 고장 사이트와 관련된 사이트에서의 복구 알고리즘으로 나누어 기술하였다. $LCPN_x = n$인 사이트 x에서 발생된 고장을 복구하기 위한 알고리즘은 다음과 같다.

알고리즘 3: 고장 사이트(단, $LCPN_x = n$)에서의 복구 알고리즘

```
(step 1) REDO_x(n) = φ; Rollback to CCP_x(n);
(step 2) for each log after CCP_x(n)
(step 3)     if (GCPN of log < n) then
                 fail("Inconsistency Checkpoint");
             endif
(step 4)     k = site identification from Tid of this log;
(step 5)     if ( REDO_x(n) ∩ k ! = φ) then
(step 6)         send( x, k, n ); /* x, k is site id,
                 n is LCPN_x */
                 REDO_x(n) = REDO_x(n) ∩ k;
             endif
(step 7)     redo this log;
         endfor
```

고장 사이트와 관련된 사이트에서의 복구 알고리즘은 다음과 같다. 이 경우에도 같은 원인의 고장임에도 불구하고 고장 사이트와 관련된 사이트에서 복구 요청이 반복적 또는 이행적으로 발생되는 순환적 재시작(cyclic restart) 현상을 방지하기 위해 Site_id_list라는 변수를 사용하였다. 이 변수는 초기에 널 값을 같는다고 가정한다.

알고리즘 4: 고장 사이트와 관련된 사이트 k에서 복구 알고리즘

```
(step 1) receive(x, n);
(step 2) if (x ∈ Site_id_list) then
             ignore this redo message;
             go to step 1;
         else
             Site_id_list = Site_id_list ∪ x;
```

```
        endif
(step 3)Site_id_list = Site_id_list ∪ x;
(step 4)if (LCPN_k < n) then
(step 4.1)   Rollback to CCP_k(n-1);
(step 4.2)   for each log after CCP_k(n-1)
(step 4.3)     if (GCPN of log ≥ n)
(step 4.4)       i=site identification from Tid of this log;
(step 4.5)       if (Site_id_list ∩ i != φ) then
(step 4.6)         send( k, i, n );/*i, k is site id,
                   n is LCPN_x */
                   Site_id_list = Site_id_list ∪ i;
                 endif
               endif
(step 4.7)     if (GCPN of log < n-1) then
                 fail("Inconsistency Checkpoint");
               endif
(step 4.8)     redo this log;
             endfor
        else
(step 5)     Rollback to CCP_x(n);
(step 5.1)   for each log after CCP_x(n)
(step 5.2)     if (GCPN of log < n) then
                 fail("Inconsistency Checkpoint");
               endif
(step 5.2)     i=site identification from Tid of this log;
(step 5.3)     if (Site_id_list ∩ i != φ) then
(step 5.4)       send(k, i, n);/*i, k is site id,
                 n is LCPN_x */
                 Site_id_list = Site_id_list ∪ i;
               endif
(step 5.5)     redo this log;
             endfor
        endif
```

5. 정확성 증명

본 논문에서 제안하는 검사점 설정 알고리즘이 DTP 시스템에서 일관성을 보장할 수 있음을 증명하기 위해 다음과 같은 용어를 정의하고, 일반적으로 관찰 되어지는 다음과 같은 성질(property)들을 이용하였다.

[정의 1] $LCPN_x = n$ 인 사이트 x에서 존재 할 수 있는 연산 로그들은 다음과 같은 4가지 집합(set) 중에 한 곳에 속하게 된다.

(1) $UL_x(n)$: n 번째 완료 검사점 설정 이 후 연산 로그의 상태와 GCPN이 아직 결정되지 않은 상태의 미결정 로그(undecided log)들의 집합.

(2) $DL_x(n)$: n 번째 완료 검사점 설정 이 후 연산 로그의 상태와 GCPN이 결정되어 있고, 로그의 GCPN이 $LCPN_x$보다 큰 경우인 결정 로그(decided log)들의 집합.

(3) $SL_x(n)$: n 번째 완료 검사점 설정 이 후 연산 로그의 상태와 GCPN이 결정되어 있고, 로그의 GCPN이 $LCPN_x$와 같은 경우의 로그로서 $CCP_x(n)$에 이미 저장되어져 있는 저장 로그(stored log)들의 집합.

(4) $CL_x(n)$: n 번째 완료 검사점 설정 이 후 연산 로그의 상태와 GCPN이 결정되어 있고, 로그의 GCPN이 $LCPN_x$와 같은 경우의 로그로서 $CCP_x(n)$에 저장 되지 못한 보상 로그(compensation log)들의 집합.

(성질 1) 임의의 트랜잭션 T_i의 모든 참여자 트랜잭션 T_{ij}들은 2-PC 프로토콜과 GCPN 결정 알고리즘에 의해 전역적으로 동일한 상태(완료 또는 취소)를 가지고, 동일한 GCPN을 가지게 된다.

(성질 2) 임의의 사이트의 $LCPN_x = n$이면 다른 모든 사이트의 $LCPN_k = n$ or $(n-1)$이 되거나 또는 $LCPN_k = n$ or $(n+1)$이 된다.

(증명) 검사점 설정 간격 δ가 (식-1)을 만족 한다는 가정에 의해 명백함(증명끝).

(성질 3) 시스템 내의 임의의 사이트 x 에서 $LCPN_x$가 n이고, n 번째 완료 검사점 $CCP_x(n)$까지 설정되어 있는 시점에 사이트 x에 존재 하는 모든 로그의 집합 $SET(LOG) = UL_x(n) \cup DL_x(n) \cup SL_x(n) \cup CL_x(n)$이고, $UL_x(n) \cap DL_x(n) \cap SL_x(n) \cap CL_x(n) = \phi$ 이다.

[정리 1] 임의의 트랜잭션 T_i의 $GCPN(T_i) = n$인 경우 T_{ix}들이 연산 내용은 $CCP_x(n)$ 또는 $CL_x(n)$에 저장된다(단, x는 T_i의 모든 참여자 트랜잭션의 사이트 명).

(증명) 임의의 트랜잭션 T_i의 모든 참여자 트랜잭션 T_{ij}는 (성질 1)에 의해 동일한 상태와 GCPN을 같는다. 따라서 (식-3)에 의해서 $GCPN(T_i) = n$ 이기 위해서는 $LCPN(T_{ix}) = n$ 또는 $n-1$이 된다. 따라서 트랜잭션 T_i와 관련된 사이트 x의 $LCPN_x = n-1$ 또는 $n-2$

가 된다.

(i) $LCPN_x = n-2$ 인 경우: 사이트 x에서는 n−1 번째 완료 검사점 $CCP_x(n-1)$을 설정하는 시점에서 알고리즘 2의 (step 5)에 의해 $CCP_x(n-1)$에 저장되지 않고 다음 완료 검사점인 $CCP_x(n)$을 설정하는 시점에 저장된다.

(ii) $LCPN_x = n-1$ 인 경우: 사이트 x에서는 n 번째 완료 검사점 $CCP_x(n)$을 설정하는 시점에서 T_{ix}가 수행한 연산 내용의 상태와 GCPN이 결정된 경우에는 알고리즘 2의 (step 3)에 의해 $CCP_x(n)$에 저장된다. 또한, $LCPN_x = n-1$ 인 경우에서 n 번째 완료 검사점 $CCP_x(n)$을 설정하는 시점에 T_{ix}가 수행한 연산 내용의 상태와 GCPN이 결정되지 않은 경우에는 n+1번째 완료 검사점 $CCP_x(n+1)$을 설정하는 시점에 알고리즘 2의 (step 4)와 (성질 3)에 의해 $CL_x(n)$에 저장된다(증명끝).

[정리 2] $LCPN_x = n$인 사이트 x에서 검사점 설정 간격 δ 동안에 존재하는 연산 로그들의 GCPN은 n, n+1, n+2 가 될 수 있다.

(증명)(i) 연산 로그의 GCPN < n 인 경우: (성질 2)에 의해 $LCPN_x = n$ 이므로 시스템 내의 다른 사이트의 $LCPN_k$의 최소 값은 n−1이 된다. 따라서 임의의 사이트 k에서 $LCPN_k(T_{ik}) < n$ 이 되는 T_i 가 존재할 수 없으므로 GCPN 결정 알고리즘에 의해 GCPN < n되는 연산 로그는 존재할 수 없다.

(ii) 연산 로그의 GCPN = n 인 경우: $CCP_x(n)$에 저장되어야 하지만 메시지 전송상의 시간 지연등의 원인으로 저장되지 못한 로그로써 n+1 번째 완료 검사점 $CCP_x(n+1)$을 설정하는 시점에 $CL_x(n)$에 저장되게 된다.

(iii) 연산 로그의 GCPN = n+1 또는 n+2 인 경우: 사이트 x의 $LCPN_x = n$인 경우에는 (성질 2)에 의해 다른 사이트의 $LCPN_k = n$ 또는 n+1이므로 임의의 연산 로그의 GCPN은 n+1 또는 n+2가 될 수 있다. 이와 같은 연산 로그들은 n+1번째 완료 검사점 $CCP_x(n+1)$에 저장되거나 또는 n+2 번째 완료 검사점 $CCP_x(n+2)$에 저장되게 된다.

(iv) 연산 로그의 GCPN > n+2 인 경우: 사이트 x의 $LCPN_x = n$인 경우에는 (성질 2)에 의해 다른 사이트의 $LCPN_k$의 최대값은 n+1 이므로 GCPN > n+2 인 경우는 존재할 수 없다(증명끝).

위의 [정리 2]에 의해 알고리즘 3의 (step 3)과 알고리즘 4의 (step 4.7), (step 5.2)에서 n 번째 완료 검사점 이후에 존재하는 연산 로그의 일관성이 위배되는 경우을 검사하여 완료 검사점이 일관성을 유지하도록 하였다.

[정리 3] $LCPN_x = n$ 인 임의의 사이트에서 고장(failure)이 발생된 경우에 복구 알고리즘은 고장 사이트와 관련된 모든 사이트 k에서 $CURRENT\_CP_k$ 또는 $CCP_k(n)$과 연산 로그를 이용하여 복구되어 진다.

(증명) $LCPN_k = n$인 경우는 (성질 3)에 의해 $CCP_k(n)$과 $CCP_k(n)$이후의 연산 로그들을 이용하여 복구할 수 있다. $LCPN_x = n$ 이므로 시스템 내의 다른 사이트들의 $LCPN_k$는 (성질 2)에 의해 n 또는 n−1 이거나 n 또는 n+1이 된다. 또한 (성질 2)에 의해 아래의 경우 i, ii는 동시에 존재할 수 없다.

(i) $LCPN_k = n-1$인 경우: [정리 2]에 의해 사이트 x에는 GCPN < n이 되는 연산 로그들은 존재할 수 없으므로 GCPN < n 인 연산 로그들은 실행 내용 상의 오류는 없고 데이타 일관성에도 문제가 없다. 따라서 GCPN < n 되는 연산 로그들은 원칙적으로 재 수행 할 필요가 없다. 다만, 분산 시스템에 존재하는 시간 지연 때문에 극히 희박한 확률이지만 (그림 5)와 같은 경우가 발생하여 $LCPN_k = n-1$인 사이트가 존재 할 수 있다. $LCPN_k = n-1$이므로 $CURRENT_CP_k = CCP_k(n-1)$이 된다. 따라서 $CURRENT\_CP_k$ 와 연산 로그를 이용하여 복구되어 진다.

(ii) $LCPN_k = n+1$인 경우: $LCPN_k = n+1$ 이므로 $CURRENT_CP_k = CCP_k(n+1)$이 된다. 따라서 이미 $CCP_k(n)$은 설정되어 있으므로 $CCP_k(n)$을 이용하여 복구되어 진다(증명끝).

[정리 4] $LCPN_x = n$ 인 임의의 사이트에서 고장(failure)이 발생된 경우에 복구 알고리즘은 고장 사이트와 관련된 모든 사이트 k에서 n 번째 완료 검사점 이전까지 복귀되지 않음을 보장한다.

(증명) [정리 3]에서와 같이 $LCPN_k = n$ 또는 n+1인 경우는 명백하다. $LCPN_k = n-1$인 경우는 알고리즘 4의 (step 4.3)에서 연산 로그의 GCPN < n되는 연산 로

그들은 다른 사이트로 재 수행 시키는 메시지를 전달하지 않도록 함으로써 다른 사이트에서 n 번째 완료 검사점 이전으로 복귀되지 않음을 보장한다(증명끝).

[정리 5] 제안하는 복구 알고리즘은 순환적 재시작 현상을 방지한다.

(증명) 순환적 재시작 현상은 고장 사이트와 관련된 임의의 사이트에서 재 수행 요구를 반복적 또는 이행적으로 수신함으로써 발생된다. 알고리즘 3의 (step 5)와 알고리즘 4의 (step 4.5), (step 5.3)에서 한번 재 수행 요구 메시지를 보낸 사이트에는 다시 메시지를 보내지 않도록 하였다. 또한 알고리즘 4의 (step 2)에 의해 재 수행 메시지를 한번 수신한 사이트에서 보낸 메시지는 무시하도록 함으로써 순환적 재시작 현상을 방지할 수 있다(증명끝).

6. 성능 분석

본 논문에서 제안하는 검사점 설정 알고리즘의 성능을 기존의 검사점 설정 알고리즘과 비교하기 위하여 (표 1)과 같은 성능 요소들을 고려하였다. 각 성능 요소들에 대한 정의는 다음과 같다.

[정의 2] 통신 비용, 간섭 비용, 저장 장치 비용.

(1) 통신 비용(communication cost): 검사점을 설정하기 위해 분산 시스템의 각 사이트의 검사점 관리자 사이에 주고 받은 메시지 수.

(2) 간섭 비용(interference cost): 검사점을 설정하는 동안 수행 중인 트랜잭션들이 실행하지 못함으로 받는 영향의 정도로 CPU시간으로 표시한다.

(3) 저장 장치 비용(stroage cost): 로그 관리자에 의해 만들어지는 연산 로그와 검사점에서 저장되는 내용을 저장하기 위한 메모리 또는 디스크 사용 공간을 표시한다.

일반적인 알고리즘의 중요한 성능 평가 요소는 알고리즘의 시간 복잡도(time complexity)이지만, 분산 처리 시스템에서 적용되는 분산 알고리즘의 성능을 좌우하는 가장 중요한 요소중의 하나는 시스템내에 요구되는 메시지 수이다. 따라서 n개의 사이트로 구성된 분산 시스템에서 동기적인 검사점 기법은 검사점을 설정하기 위해 최소 2*n개의 메시지를 필요로 한다. 비동기적인 방법들은 각 사이트에서 독자적으로 검사점을 설정하기 때문에 검사점을 설정하는 시점에서의 통신 비용은 없다. 본 논문에서 제안하는 검사점 설정 알고리즘도 기존의 2-PC 프로토콜을 사용하여 결정되기 때문에 검사점을 설정하기 위한 별도의 메시지는 사용하지 않으므로 [정의 2]에 의해 본 알고리즘에서 소요되는 통신 비용은 없다.

그러나 비동기적 기법에서는 검사점을 설정하는 시점에서 통신 비용의 부담은 없지만 고장이 발생한 경우 일관성이 보장된 검사점을 찾아서 복구해야 하기 때문에 복구 알고리즘이 어렵고 복잡하게 된다. 그러나 동기적 기법에서는 검사점을 설정하는 시점에 통신 비용을 부담하여 일관성을 유지하도록 검사점을 설정하였으므로 고장이 발생한 경우 복구 알고리즘은 상대적으로 간단하게 된다. 제안하는 알고리즘도 [정리 3]에서 설명하였듯이 $LCPN_x = n$ 인 임의의 사이트에서 고장(failure)이 발생된 경우에 복구 알고리즘은 고장 사이트와 관련된 모든 사이트 k에서 $CURRENT\_CP_k$ 또는 $CCP_k(n)$과 연산 로그를 이용하여 간단하게 복구할 수 있다.

또한 동기적 기법에서는 일관성이 보장되는 검사점을 설정하는 동안 다른 트랜잭션이 실행될 수 없으므로 간섭 비용이 심각하게 된다. 비동기적 기법들은 검사점이 다른 사이트와 독립적으로 설정되기 때문에 검사점을 설정하는 동안 간섭 비용을 최소화 할 수 있다. 제안하는 알고리즘도 비동기적인 기법들과 유사하게 각 사이트에서 독립적으로 검사점을 설정하고, 기존의 2-PC 프로토콜을 이용하여 검사점에 일관성을 보장하기 때문에 일관된 검사점을 설정하기 위한 별도의 간섭 비용 없이 알고리즘을 운영할 수 있다. 단, 기존의 방법과 같이 각 사이트에서 검사점이 실행되는 동안에 검사점의 내용을 안정 기억 장치에 데이타를 저장하기 위한 작업등은 필요하지만 이러한 시간은 별도의 하드웨어를 사용하여 구현되어질 수 있다.

본 논문에서 제안하는 기법은 일관성이 보장되는 검사점을 설정할 수 있기 때문에 임의의 사이트 x에서 가장 최근의 완료 검사점 $CCP_x(n)$에 해당하는 내용만을 안정 저장 장치에 기억시키면 된다. 다만, [정리 3]의 (ii)와 같은 경우가 발생될 수 있기 때문에 최

근의 완료 검사점 $CCP_x(n)$과 직전 완료 검사점 $CCP_x(n-1)$의 내용만을 저장한다. 따라서 제안하는 알고리즘은 시스템내의 각 사이트에서 두개의 완료 검사점 내용을 저장할 수 있는 저장 장치 비용을 요구함으로 기존의 검사점 설정 알고리즘에 비해 최소한의 기억 장치 용량만을 요구한다.

또한 기존의 동기적 또는 비동기적인 검사점 설정 방법에서는 순환적 재 시작 현상에 관한 언급이 없거나 해결 가능성을 불분명하게 기술하고 있으나 제안하는 기법에서는 순환적 재시작 현상을 제거 할 수 있다(∵ [정리 5]). 그리고 비동기적인 검사점 설정 기법에서는 도미노 현상이 발생할 수 있지만, 제안 하는 기법에서는 도미노 현상을 제거할 수 있다(∵ [정리 4]). 이상의 내용을 정리하면 〈표 1〉과 같다.

7. 결 론

일반적으로 분산 처리 시스템에서 일관성이 보장되는 검사점의 설정은 일관성을 보장하기 위해 주고받는 메시지 때문에 발생되는 통신 비용과 이러한 일관된 시점을 결정하는 동안에 트랜잭션들이 더 이상 활동(active) 상태가 될 수 없는 간섭 현상을 유발하게 된다. 또한 검사점의 내용을 저장하기 위한 저장 공간도 시간에 따라 증가하게 된다.

본 논문에서 제안하는 검사점 설정 알고리즘은 트랜잭션 실행에 원자성을 보장하는 기존의 2-PC 프로토콜을 이용하여 쉽고, 효과적으로 일관성을 보장하는 검사점을 설정하도록 하였다. 제안하는 검사점 설정 알고리즘은 2-PC 프로토콜에서 사용되는 메시지를 이용하여 일관성이 보장되는 검사점을 설정하기 때문에 추가의 메시지 부담이 없다. 또한 활동 중인 트랜잭션의 간섭 효과를 최소화 함으로써 분산 트랜잭션들의 실 시간성을 유지 할 수 있도록 하였다. 검사점을 저장하기 위한 기억 공간도 임의의 사이트에서 현재 완료 검사점과 직전 완료 검사점의 두 내용만을 저장하게 하여 저장 장치 비용을 최소화 하였다.

그리고 제안하는 복구 알고리즘은 [정리 3], [정리 4]에서 설명하였듯이 $LCPN_x = n$인 사이트에서 발생된 고장으로 인하여 n 번째 완료 검사점까지만 복귀하여 오류를 복구하도록 하였다. 따라서 제안하는 복구 알고리즘은 도미노 현상을 제거할 수 있고, [정리 5]와 같이 순환적 재시작 현상을 제거할 수 있다. 앞으로의 연구 과제로는 제안하는 검사점 설정 및 복구 알고리즘을 실제 분산 트랜잭션 처리 시스템에 구현하기 위한 연구와 구현을 통한 성능 분석등이 있을 수 있다.

참 고 문 헌

[1] S. H. Son and A. K. Agrawala, "Distributed Checkpointing for Globally Consistent States of Databases", IEEE Tr. on SE, Vol. 15, No. 10, Pages 1157-1167, Oct. 1989.

[2] Hong Va Leong, Divyakant Agrawal, "Using Message Semantics to Reduce Rollback in Optimistic Message Logging Recovery Schemes", 1063-6927/94, 1994 IEEE, Pages 227-234, 1994.

[3] Dhiraj K. Pradhan, Nitin H. Vaidya, "Roll-Forward and Rollback Recovery: Performance-Reliability Trade-off", 0363-8928/94, 1994 IEEE, Pages 186-195, 1994.

[4] V. F. Nicola and J. M. V. Spanje, "Comparative Analysis of Different Models of Checkpointing and Recovery", IEEE Tr. on SE, Vol. 16, Pages 807-821, Aug. 1990.

[5] A. Reuter, "A Fast Transaction-Oriented Logging Scheme for Undo Recovery", IEEE Tr. on SE, Vol. SE-6, Pages 348-356, July 1980.

[6] R. Koo and S. Toueg, "Checkpointing and Rollback-recovery for Distributed Systems", IEEE Tr. on SE, Vol. SE-13, No. 1, Pages 23-31, Jan. 1987.

[7] D. B. Johnson and W. Zwaenepoel, "Recovery in Distributed Systems", Journal of Algorithms, Vol. 11, No. 3, Pages 462-491, Sep. 1990.

[8] K. M. Chandy and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems", ACM Tr. on Computer Systems, Vol. 3, No. 1, Pages 63-75, Feb. 1985.

[9] L. Lamport, "Time, Clocks and the Ordering of Events in Distributed Systems", CACM, Vol. 21, No. 7, Pages 558-565, July 1978.

[10] Nicholas S. Bowen, Dhiraj K. Pradhan, "Virtual

Checkpoints: Architecture and Performance", IEEE Tr. on Computers, Vol 41, No. 5, Pages 516-525, May 1992.

[11] Eyal Zimran, "The Two_Phase Commit Performance of the DECdtm Services," Proceedings 11th Symposium on Reliable Distributed Systems, Oct. 5-9 1992, pages 29-38.

[12] Jim Gray, Andreas Reuter, "Transaction Processing: Concept and Techniques," Morgan Kaufmann Publishers, 1993.

[13] Bill Graybook, "OLTP: Online Transaction Processing Systems," Wiley Professional Company, 1992.

박 윤 용

1983년 숭전대학교 계산통계학과 졸업(학사)
1985년 서울대학교 대학원 계산통계학과(이학 석사)
1994년 서울대학교 대학원 계산통계학과(이학 박사)
1985년~1992년 한국전자통신연구소 연구원
1992년~현재 선문대학교 전자계산학과 조교수
관심분야: 분산처리운영체제, 멀티미디어시스템, 고장 감래 시스템

전 성 익

1985년 중앙대학교 전자계산학과 졸업(학사)
1987년 중앙대학교 대학원 전자계산학과(이학 석사)
1992년 정보처리기술사
1987년~현재 한국전자통신연구소 선임연구원
관심분야: 분산처리운영체제, 고장 감래 시스템

조 주 현

1978년 서울대학교 전자공학과 졸업(학사)
1984년 한국과학기술원 전자계산학과(이학 석사)
1993년 한국과학기술원 전자계산학과(박사 수료)
1978년~현재 한국전자통신연구소 책임연구원(실시간 OS 연구실장)
관심분야: 분산처리운영체제, 고장 감래 시스템, 객체지향시스템