인과적 메시지 로깅 기법에서 부가적 메시지 교환없는 메시지 로그 쓰레기 처리 기법

(A Garbage Collection of Message Logs without Additional Messages on Causal Message Logging Protocol)

정 광 식 † 유 헌 창 †† 백 맹 순 † 손 진 곤 ††† 황 종 선 ††††

(Kwang Sik Chung) (Heon-Chang Yu) (Maeng-Soon Baik) (Jin Gon Shon) (Jong-Sun Hwang)

요 약 이 논문은 인과적 메시지 로깅 기법에서 결함 포용을 목적으로 안전 저장 장치(stable storage)에 저장되는 메시지 내용 로그와 메시지 순서 로그의 쓰레기 처리 기법을 제안한다. 기존의 인과적 메시지 로깅을 기반으로 한 메시지 로그 쓰레기 처리 기법은 메시지 순서 로그의 쓰레기만을 처리하였다. 메시지 내용 로그는 불필요한 복귀를 피하기 위해 유지해야 하며, 메시지내용 로그의 쓰레기 처리를 위해서는 부가적인 메시지를 필요로 하였다.

이 논문은 인과적 메시지 로깅 기법에서 메시지 내용 로그와 메시지 순서 로그가 쓰레기 처리되어지기 위해 필요한 조건을 새롭게 정의하며, 이 정의를 기반으로 한 메시지 내용 로그와 메시지 순서 로그의 쓰레기 처리 알고리즘을 제안한다. 제시된 조건을 기반으로 한 메시지 내용 로그와 메시지 순서 로그의 쓰레기 처리는 송수신 메시지에 부가된 *MAG(Modified Antecednce Graph)*를 이용하므로 쓰레기 처리를 위한 부가적인 메시지를 발생시키지 않는다. 하지만 제안된 기법은 일반 메시지가 송수신되기 전까지 쓰레기 처리가 지연되는 '지연 쓰레기 처리 현상(lazy garbage collection)'을 발생시킨다. 하지만 '지연 쓰래기 처리 현상'은 분산 시스템의 일관성을 위배하지 않으며, 쓰레기 처리를 위한 부가적인 메시지 교환을 필요로 하지 않는다.

Abstract This paper presents a garbage collection protocol for message content logs and message determinant logs which are saved on the stable storage for fault tolerance. The previous works of garbage collections in causal message logging protocol try to solve the garbage collection of message determinant log. In order to avoid the sympathetic rollback, the process has to maintain the message content logs. But the previous works, in order to solve the garbage collection of message content logs, they needed additional messages.

We propose new definitions for garbage collections conditions for message determinant logs and message content logs and the garbage collection algorithm based on the definitions. Since the proposed algorithm uses *MAG(Modified Antecednce Graph)*, the additional messages for garbage collection is not needed. The proposed garbage collection algorithm makes 'the lazy garbage collection effect', because relying on the piggybacked checkpoint information in send/receive message. But 'the lazy garbage collection effect' does not break the consistency of the whole systems and does not need additional messages for the garbage collections.

† 비 회 원 : 고려대학교 컴퓨터학과
cks@disys.korea.ac.kr
msbak@disys.korea.ac.kr
†† 정 회 원 : 고려대학교 컴퓨터교육과 교수
yuhc@disys.korea.ac.kr
††† 종신회원 : 한국방송통신대학교 컴퓨터과학과 교수
†††† 종신회원 : 고려대학교 컴퓨터학과 교수
hwang@disys.korea.ac.kr
논문접수 : 2000년 11월 27일
심사완료 : 2001년 5월 10일

1. 서 론

분산 컴퓨팅 시스템의 발달로 인해 하나의 작업이 여러 프로세스에서 분산되어 수행되고 있다. 분산 작업의 수행은 결함 발생 확률을 높이며, 이에 따라 결함 발생을 포용해 줄 수 있는 여러 가지 결함 포용 기법이 개발되고 있다. 결함 포용을 위한 대표적인 기법으로 검사점 기법과 메시지 로깅 기법이 있으며, 결함 포용 기법들은 결함 포용을 위해 검사점과 메시지 로깅 정보를 저장한

다. 이와 같은 결함 포용을 위한 정보의 유지는 어느 시점을 지나면 차지하는 메모리의 양을 증가시키게 되고, 저장된 상태 정보 중 불필요한 정보의 처리, 즉 쓰레기 처리(garbage collection)를 필요로 한다. 검사점 기법은 결함 포용을 위해 프로세스의 상태 정보를 저장하며, 쓰레기 처리의 대상은 검사점이다[3,4]. 메시지 로깅 기법은 비관적(pessimistic), 낙관적(optimistic), 인과적 메시지 로깅 기법(causal)으로 나뉘고, 결함 포용을 위해서 메시지 내용 로그와 비결정적 사건에 대한 정확한 순서 정보를 저장하였다.

인과적 메시지 로깅 기법은 비결정적 사건의 순서 로그를 안전 저장 장치(stable storage)에 저장하지 않고, 여러 프로세스의 불안전 저장 장치(volatile storage)에 저장함으로써 메시지 순서 로그를 유지하며, 안전 저장 장치에 비동기적으로 저장함으로써 낙관적 메시지 로깅 기법의 장점을 유지한다[5,6]. 기존의 인과적 메시지 로깅 기법에서 쓰레기 처리의 주요 대상은 메시지 순서 로그였다. 이것은 인과적 메시지 로깅 기법에서 결함 회복이 메시지 순서 로그를 기반으로 수행되었기 때문이다. 하지만 결함 회복 수행의 성능 향상을 위해 대부분의 메시지 로깅 기법에서는 메시지 내용 로그를 유지한다[6,7,8]. 쓰레기 처리 대상을 메시지 순서 로그만으로 정하였기 때문에, 메시지 내용 로그의 쓰레기 처리를 위해 부가적인 메시지 교환을 필요로 했다[7].

이 논문에서는 인과적 메시지 로깅 기법에서 메시지 로그를 메시지 내용 로그와 메시지 순서 로그로 분류하여 정의하였다. 이를 기반으로 논문[7]에서 제안된 참여 그래프(antecedence graph)를 새롭게 정의한 *MAG* (Modified Antecedence Graph)를 이용하여 쓰레기 처리를 위한 부가적 메시지 송수신을 제거하였다. 이 논문의 2장에서는 기존의 인과적 메시지 로깅 기법에서 수행되었던 메시지 순서 로그와 메시지 내용 로그의 쓰레기 처리 기법의 문제점과 연구 수행 방법에 대해 언급한다. 3장에서는 메시지 내용 로그와 메시지 순서 로그의 쓰레기 정보 탐색을 위한 조건을 정의한다. 4장에서는 이 논문에서 새롭게 제안되는 메시지 내용 로그와 메시지 순서 로그의 쓰레기 처리 알고리즘과 정당성의 증명을 보인다. 5장에서는 기존의 기법과 이 논문에서 제안된 *MAG*를 이용한 쓰레기 처리 기법을 실험을 통해 비교 분석한다. 6장에서는 연구의 결론과 향후 연구 과제에 대해 언급한다.

2. 관련 연구

결함 포용 기법에서 저장 장치는 불안정 저장 장치와 안정 저장 장치를 가정한다. 쓰레기 처리 기법이란 결함 포용 정보가 안정 저장 장치로부터 삭제되어지는 시기를 결정하는 것이다. 기존의 결함 포용 기법에서는 일관된 전역 검사점(consistent global checkpoints)의 구성에 기반한 쓰레기 처리 기법을 사용하거나[3,4], 부가적인 메시지의 교환을 포함하는 쓰레기 처리 기법을 제안하였다[7,11].

일관된 전역 검사점(consistent global checkpoints)의 구성을 이용한 쓰레기 처리 기법은 전역 검사점이 완료되는 시점에서, 전역 검사점의 구성에 참여한 프로세스들이 안전 저장 장치에 있는 가장 최근의 검사점을 제외한 모든 결함 포용 정보를 삭제한다[3,4,10]. 일관된 전역 검사점의 구성은 가장 최근의 검사점 이전에 발생한 결함 포용 정보를 필요로 하지 않으므로 쓰레기 처리의 정당성이 보장된다.

하지만 인과적 메시지 로깅 기법에서는 일관된 전역 검사점의 구성이 보장되지 않는다. 인과적 메시지 로깅에 기반한 쓰레기 처리 기법은 일관된 검사점 기법의 쓰레기 처리에 비해 훨씬 복잡하다[8]. 즉, 검사점 시점에 관련된 정보를 부가적으로 교환해야 하는 부담을 가지고 있다[8]. 인과적 메시지 로깅 기법을 사용하는 Manetho[7]에서 프로세스 p는 검사점 $ckpt_i$에 의해 결정되는 상태 간격(state interval) σ_i^p이전에 보내진 메시지 순서 로그의 쓰레기 처리를 결정할 수 있다. 즉, Manetho에서 프로세스는 마지막 검사점이전으로 복귀(rollback)하지 않는다. 따라서 프로세스 p가 프로세스 q에게 메시지를 보낼 때, q가 실제로 그 메시지를 받았지만 아직 검사점을 취하지 않았다면, p는 q에게 검사점을 취할 것을 요청한다. 검사점 요청을 받은 모든 프로세스들이 검사점을 취한다면, 프로세스 p는 안전하게 σ_i^p이전에 보내진 모든 메시지 내용 로그를 쓰레기 처리할 수 있다. 또한, 각 프로세스는 메시지 로그를 쓰레기 처리하기 전에, 다른 프로세스의 상태 간격 인덱스(state interval index)를 포함한 리스트를 관리한다. 그리고 메시지 순서 정보(참여 그래프 ; antecedence graph)에 대해서는 현재 검사점보다 이전의 검사점에 대응되는 참여 그래프의 노드들을 지울 수 있다. 즉, Manetho에서는 이전의 검사점에 대응되는 참여 그래프의 노드 정보는 복구에 더 이상 필요하지 않게 된다. 참여 그래프의 쓰레기 처리는 프로세스들이 자신의 가장 최근의 검사점 인덱스를 상태 간격 정보와 함께 교환함으로서 가능하다. 결국 메시지 내용 로그의 쓰레기 처리를 위해 강제적 검사점을 요구하고, 검사점 요구를 위해

부가적인 메시지의 교환을 필요로 한다.

이러한 비용을 줄이기 위해 이 논문에서는 메시지 로그를 새롭게 정의하였으며, 이를 기반으로 메시지 로그의 쓰레기 처리 조건을 제안하였다. 또한 이를 구현하기 위한 자료 구조로 메시지 순서 정보에 검사점 정보를 부과한 개선된 메시지 순서 정보를 정의하고, 이 정보를 기반으로 메시지 내용 로그와 메시지 순서 로그를 쓰레기 처리 조건 알고리즘을 제안한다.

3. 시스템 모델

분산 시스템 N은 n개의 프로세스로 구성된다. N의 수행 단위를 ρ라 하며, ρ동안 전달된 메시지는 m로 정의한다. 메시지 m의 송신자 프로세스는 $m.source$ 의 식별자를 가지며, $m.ssn$은 송신자 프로세스에 의해 메시지에 부여된 메시지 송신 식별자이다. $deliver_{m.dest}(m)$는 메시지 수신 프로세스에 의해 메시지의 수신 사건이 프로세스의 상태에 반영되어 분산 시스템의 전체 상태가 전이하였음을 나타낸다[3].

$$Depend(m) \stackrel{\text{def}}{=} \left\{ j \in N \mid \begin{array}{c} \bigvee ((j = m.dest) \\ \bigwedge j \text{ has delivered } m) \\ \bigvee (\exists m' : (\ deliver_{m.dest}(m) \\ \rightarrow \ deliver_j(m')) \end{array} \right\}$$

$Depend(m)$은 m이 수신 프로세스에 반영된 후 발생한 송신 메시지 m'이 반영된 프로세스들과 메시지 m이 반영된 수신 프로세스의 합집합을 의미하며, $Depend(m)$에 포함되는 프로세스는 메시지 m에 의존한다.

메시지에 대한 결함 포용 정보는 결함 정보를 이용하는 복귀 회복 기법의 관점에서 두 가지로 분류되어 관리된다. 먼저 메시지에 의해 전달되는 응용 프로그램의 메시지 내용이다. 이것은 $content_m$으로 나타낸다. 두 번째로 송수신 프로세스들 사이의 메시지의 송수신 순서를 나타낸다. 이것은 $\#_m$으로 나타내며, 각각 다음과 같이 정의된다.

[정의 1] 메시지 내용 정보와 메시지 순서 정보는 다음과 같이 정의된다.

메시지 순서 정보 $\#_m$는 $\langle m.source, m.ssn, m.rsn \rangle$로 구성되며, 메시지 내용 정보 $content_m$는 $\langle m.data, m.ssn, m.rsn \rangle$로 구성된다.

효율적인 결함 포용 정보 관리를 위해 필요한 메시지 m에 관련된 결함 포용 정보(FT_m)를 다음과 같이 정의한다.

$$FT_m \stackrel{\text{def}}{=} \{content_m\} \bigcup \{Depend(m)\}$$

분산 시스템 전체 집합 N에서 결함이 발생한 결함 프로세스의 집합 C에서 고아 프로세스 p가 다음과 같이 정의된다.

$$orphan process p of C \stackrel{\text{def}}{=} \left(\begin{array}{l} \bigwedge (p \in N - C) \\ \bigwedge (\exists m : ((p \in Depend(m)) \\ \bigwedge (Log(\#_m) \subseteq C))) \end{array} \right)$$

$Log(\#_m)$은 $\#_m$의 복사본을 불안전 저장 장치에 가지고 있는 프로세스들의 집합이며, 메시지 순서 로그라 한다. $Log(content_m)$은 $content_m$의 복사본을 불안전 저장 장치에 가지고 있는 프로세스들의 집합이며, 메시지 내용 로그라 한다. $N-C$의 집합에 속하는 프로세스 p가 $Depend(m)$의 원소이고, $Log(\#_m)$이 C의 부분집합을 만족하는 프로세스 p를 고아 프로세스로 정의한다. 위의 고아 프로세스 정의에 의해서 고아 프로세스가 되지 않기 위한 조건은 다음과 같다

$$\forall m : ((Log(\#_m) \subseteq C) \Rightarrow (Depend(m) \subseteq C))$$

메시지 m의 $\#_m$을 로그하고 있는 프로세스들의 집합이 결함이 발생한 프로세스의 집합 C에 포함된다면, 메시지 m에 의존하고 있는 모든 프로세스 집합 역시 C의 부분집합이 된다

$$\forall m : (Depend(m) \subseteq Log(\#_m))$$

따라서, 메시지 m에 의존하고 있는 프로세스 집합은 m을 로그하고 있는 집합의 부분집합이 된다. 위의 두 조건을 만족하는 프로세스는 고아 프로세스가 아니다.

[조건 1]은 인과적 메시지 로깅 기법에 의해 프로세스가 고아 메시지를 발생시키지 않는 조건이며, 이 조건을 만족하는 프로세스로 구성된 시스템은 결함 발생과 관계없이 일관성을 유지한다.

[조건 1] 시스템의 일관성 유지를 위한 조건 : 인과적 메시지 로깅 기법에서 시점 t까지 발생된 모든 메시지가 다음의 조건을 만족한다면, 시점 t에서, 결함 발생에 관계없이 전체 분산 시스템은 일관성을 유지한다.

$$\forall m : \Box(((\mid Log(\#_m) \mid \leq f) \Rightarrow ((Depend(m) \subseteq (Log(\#_m) \wedge \neg stable(\#_m)))) \wedge \Diamond(Depend(m) = (Log(\#_m) \wedge \neg stable(\#_m)))))$$

[증명] : [조건 1]은 메시지 로깅을 위한 분산 알고리즘의 안정성 조건(safety property)이다. 따라서 [조건 1]은 전체 시스템의 시작 상태에서부터 종결 상태까지 항상(always; $\Box$) 만족되어야 한다. 즉, 결함이 발생하더라도 전체 시스템의 일관성을 유지하면서, $\#_m$에 대한 로그가 f개 이하 존재하는 것을 항상 만족하기 위해서는, 메시지 m을 인과적으로 전달한 프로세스의 집합이 $\#_m$을 로깅한 프로세스의 집합과 $\#_m$이 안전 저장 장치에 저장되지 않은 프로세스의 집합의 교집합에 포함

되고, 결국(eventually; ◇) 두 집합이 같아야한다. 만일 인과적 메시지 전달에 관련된 프로세스들이 제한된 한 시점까지 모두 로깅을 했다면, f개의 결함 발생에 의한 복귀(rollback)는 존재하지 않는다. ■

$\#_m$의 복사본을 안전 저장 장치에 저장한 프로세스의 집합을 $stable(\#_m)$이라고 한다. f개까지의 결함 발생을 가정한다면, [조건 1]을 만족하는 인과적 메시지 로깅 기법은 전체 시스템의 일관성을 유지시킬 수 있다.

또한 결함 프로세스에서의 손실 메시지로 인해 발생되는 불필요한 복귀(sympathetic rollback)를 없애기 위해서, 송신자 기반의 메시지 내용 로그가 필요하다[9]. 따라서 인과적 메시지 로깅 기법에서는 메시지 내용에 대한 로깅 조건이 필요하며, 메시지 내용의 로깅 조건은 [조건 2]와 같다.

[조건 2] 불필요한 복귀의 회피 조건 : 인과적 메시지 로깅 기법에서 발생되는 송신 메시지에 대한 로그가 시점 t에서 다음의 조건을 만족한다면, 결함 프로세스의 복귀로 발생되는 손실 메시지로 인한 불필요한 복귀는 발생되지 않는다.

$$\forall m : \Box(\neg stable(\#_m) \Rightarrow ((Log(content_m) \subseteq C) \Rightarrow \Diamond(Depend(m) \subseteq C)))$$

[증명] : [조건 2]는 불필요한 복귀를 발생시키지 않기 위한 분산 알고리즘의 안정성 조건(safety property)이다. 따라서 [조건 2]는 전체 시스템의 시작 상태에서부터 종결 상태까지 항상(always) 만족되어야 한다. 즉, 메시지 m의 $content_m$을 로깅한 프로세스 집합이 결함 프로세스 집합에 포함되고, 결국 메시지 m을 인과적으로 전달한 프로세스들의 집합이 결함 프로세스 집합에 포함된다면, 메시지 m의 $\#_m$을 안전 저장 장치에 저장한 프로세스는 항상 존재하지 않는다. 따라서 메시지 m의 $content_m$으로 인한 불필요한 복귀는 발생되지 않는다. ■

전체 시스템의 정상 작업 수행 중에 쓰레기 처리되어야 하는 정보는 메시지의 송수신 순서인 $\#_m$과 메시지의 내용인 $content_m$이 있다. $\#_m$과 $content_m$이 쓰레기 처리되는 조건은 회복 기법에서 필요한 정보를 제외한 불필요한 정보를 최대한 많이 찾아주어야 한다.

[조건 3]은 $\#_m$의 쓰레기 처리 조건이다.

[조건 3] 메시지 내용 로그의 쓰레기 처리 조건 : 인과적 메시지 로깅 기법에서 발생되는 송신 메시지에 대한 로그가 시점 t 에서 다음의 조건을 만족한다면, 메시지 내용 로그는 쓰레기 처리되어 질 수 있다.

$$\exists m, \exists P_k : (deliver_{m.dest}(m) \wedge Chpt_{P_k}(\#_m) \Rightarrow garbage(\#_m))$$

[조건 4]는 $content_m$의 쓰레기 처리 조건이다. 수신 프로세스의 검사점이 취해졌을 때 메시지의 송수신 순서 정보에 표시된 검사점 정보를 기반으로, 검사점 이전의 메시지 내용 로그는 쓰레기 처리되어 질 수 있다.

[조건 4] 메시지 순서 로그의 쓰레기 처리 조건 : 인과적 메시지 로깅 기법에서 발생되는 송신 메시지에 대한 로그가 시점 t에서 다음의 조건을 만족한다면, 메시지 내용 로그는 쓰레기 처리되어 질 수 있다.

$$\exists m : (deliver_{m.dest}(m) \wedge Chpt_{m.dest}(\#_m) \Rightarrow garbage_{m.source}(content_m))$$

$Chpt_{m.dest}(\#_m)$는 프로세스 $m.dest$가 메시지 m의 송수신 순서 정보를 포함하는 검사점을 취하는 사건이며, $garbage_{m.source}(content_m)$는 프로세스 $m.source$가 메시지 m의 내용 로그를 쓰레기 처리하는 사건이다.

4. 결함 포용 정보의 쓰레기 처리

4장에서는 앞서 제안된 쓰레기 처리 조건이 만족되었는지를 결정하기 위해 이 논문에서는 검사점과 메시지 순서 정보를 이용하였다. 따라서 검사점과 메시지 순서 정보를 함께 전달할 수 있는 자료구조로서 Manetho에서 사용하는 참여 그래프를 수정한 *MAG*와 *MAG*를 이용한 쓰레기 처리 알고리즘을 설명한다.

4.1 쓰레기 처리 알고리즘

인과적 메시지 로깅 기법을 사용하는 Manetho 시스템은 결함 회복 성능을 향상시키기 위해 송신된 메시지의 내용을 송신자 기반의 낙관적 메시지 로깅 기법으로 관리한다. 하지만 메시지 내용 로그와 메시지 송수신 순서 정보인 참여 그래프의 쓰레기 처리를 위해 부가적인 메시지의 전달을 필요로 한다. 하지만 이 논문에서는 쓰레기 처리를 위한 부가적 메시지의 발생을 줄이기 위해 메시지 순서 정보에 메시지 송수신 순서외에 검사점 발생 사건도 프로세스의 상태 전이 사건으로 기록한다. 검사점 발생 사건이 부가된 메시지 송수신 순서 정보는 그래프 형태로 메시지에 부가되어 전달되며, 프로세스들 사이의 부가적인 메시지 없이 프로세스의 메시지 송수신 순서 정보와 메시지 내용 로그의 쓰레기 처리를 가능하게 한다. 검사점 발생 사건이 부가된 메시지 송수신 순서 정보는 다음과 같이 정의된다.

[정의 2] *MAG* (*Modified Antecedence Graph*)

i) $MAG = (V, E)$
ii) $V(MAG) = \{\langle i, j\rangle \cup C_{p_{i,k}} \mid i$: *process id*, j: *event number*, k: *checkpoint number*$\}$
iii) $E(MAG) = \langle V_{p_{i,k}}, V_{p_{i,l}}\rangle$
iv) $\langle V_{p_{i,k}}, V_{p_{j,l}}\rangle \neq \langle V_{p_{j,i}}, V_{p_{i,k}}\rangle$

*MAG*는 점(vertices)과 선(edges)으로 구성되며, 점은 사건의 발생을 의미한다. *MAG*에 포함되는 사건은 메시지 수신 사건, 메시지 송신 사건, 검사점 발생 사건으로 구성된다. 간선은 사건과 사건을 연결하며, 사건사이의 발생 순서를 표현한다.

그림 1은 [조건 3]과 [조건 4]를 만족시키며, [정의 2]에서 정의된 *MAG*를 사용하는 쓰레기 처리 알고리즘이다. 프로세스 P_j는 프로세스 P_i에게 메시지 m을 보내며, 자신이 가지고 있는 *MAG*를 부가하여 보낸다.

```
P_j : sending process
P_i : receiving process
procedure Garbage_content_M
    {
      MAG ← P_i's MAG ∪ m_{P_j}.MAG;
      while( (P_k ∈ MAG) ∧ ( m^{P_k}_{P_i} ∉ MAG))
          stable_content_M_log
              = garbage(stable_content_M_log, content_{m^{P_k}_{P_i}} );
    }
procedure Garbage_#_M
    {
      MAG ← P_i's MAG ∪m_{P_i}.MAG;
      while( P_k ∈ MAG)
          MAG = garbage(MAG, MAG depended with
                P_k before P_k's Chpt);
      P_i's MAG = MAG;
    }
procedure garbage(Source Information, Target Information)
    {
      garbage_information = Target Information;
      remove garbage_information on stable storage;
    }
```

그림 1 메시지 내용 로그 및 순서 로그의 쓰레기 처리 알고리즘

그림 1의 알고리즘에서 프로세스 P_i가 메시지 m을 받았을 때, P_i는 메시지 내용 로그와 메시지 순서 로그에 대한 쓰레기 처리를 할 수 있다. 먼저 메시지 내용 로그를 쓰레기 처리하기 위해 P_i는 메시지 m에 포함된 *MAG*와 P_i가 현재 유지하고 있는 *MAG*의 합집합을 만든다. *MAG*의 합집합을 이용하여 [조건 4]를 만족시키는 메시지 내용 로그를 선택하여 쓰레기 정보(Target Information)로 정의한다. 쓰레기 정보로 정의된 결함 포용 정보는 garbage 함수에서 안전 저장 장치로부터 제거된다. [조건 3]를 만족하는 메시지 순서 로그 정보는 쓰레기 정보로 정의된다. 쓰레기 정보로 정의된 메시지 순서 로그 정보는 garbage 함수에서 안전 저장 장치로부터 제거된다.

4.2 메시지 순서 로그의 쓰레기 탐지

Manetho의 경우 참여 그래프의 쓰레기 탐지를 위해 비정기적으로 가장 최근의 검사점의 상태 간격 인덱스(state interval index)를 교환한다. 하지만 이 기법은 부가적인 메시지의 발생을 필요로 한다. 이를 해결하기 위해 참여 그래프내의 검사점 사건을 기록하여 부가적인 검사점 인덱스 메시지 교환을 방지한다.

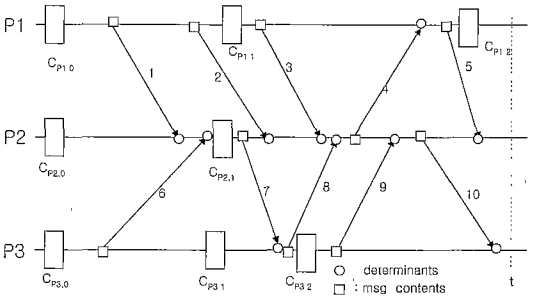


그림 2 시스템 작업 수행도

메시지 순서 정보는 어느 프로세스라도 할 수 있다. 따라서 위에서 제시된 조건에 만족하는 모든 메시지 순서 정보는 쓰레기 탐지 및 처리되어 질 수 있다.

그림 2에서 다른 프로세스의 결함 포용을 위해 존재하는 메시지 내용 로그 중, 시점 t 에서 메시지 1, 2, 3, 4, 6, 7, 8에 대한 순서 로그는 더 이상 필요하지 않다. 따라서 시점 t 이후의 메시지 송신사건에서 부가되는 메시지 순서 로그는 없다.

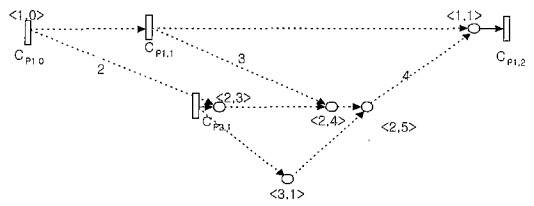


그림 3 P_1의 메시지 순서 로그 쓰레기 처리

그림 3에서 프로세스 P_1은 메시지 4를 수신한 이후에, 참여 그래프에 표시된 검사점 $C_{P2,1}$로부터 메시지 1에 대한 순서 로그는 더 이상 필요치 않다는 것을 알 수 있다.

그림 4에서 프로세스 P_2에서 송신된 메시지에는 메시지 1과 메시지 6의 순서 정보가 제거된다. 또한 검사점 $C_{P1,2}$에 의해 메시지 2, 3, 4, 7, 8에 대한 메시지 순서 로그는 삭제되어질 수 있다.

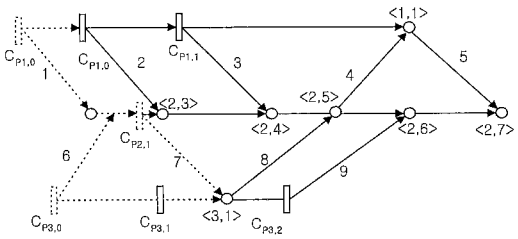


그림 4 P_2의 메시지 순서 로그 쓰레기 처리

시점 t에서 메시지 2, 3, 4, 7, 8의 메시지 순서 로그는 더 이상 필요 없다. 하지만, 프로세스 P_1의 검사점 $C_{P1,2}$가 취해진 이후에 프로세스 P_1로부터 메시지를 수신한 적이 없기 때문에 프로세스 P_2는 메시지 2, 3, 4, 8의 메시지 순서 로그를 쓰레기 처리 할 수 없다. 그 후 P_3으로부터 메시지 9를 받은 후, P_3이 검사점 $C_{P3,2}$를 취했다는 것을 알게되고, 이를 통해 메시지 7에 대한 메시지 순서 로그를 삭제할 수 있다.

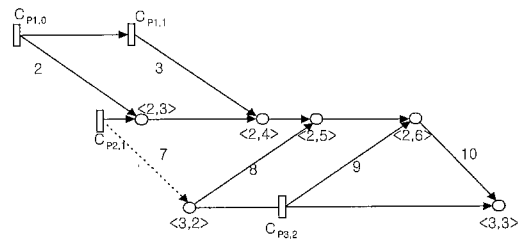


그림 5 P_3의 메시지 순서 로그 쓰레기 처리

그림 5에서 프로세스 P_3의 메시지 6에 대한 메시지 순서 로그는 프로세스 P_2로부터 메시지 7을 받은 후, 검사점 $C_{P2,1}$을 취했다는 정보를 얻는다. 그 후 프로세스 P_3은 메시지 1, 6에 대한 메시지 순서 정보를 쓰레기 처리 할 수 있고, $C_{P3,2}$ 이후에 메시지 7에 대한 메시지 순서 로그를 삭제할 수 있다.

4.3 메시지 내용 로그 쓰레기 탐지

Manetho의 경우 메시지 내용 로그의 쓰레기 정보 탐지를 위해 다른 프로세스로부터 검사점 확인(Ack)을 받아야하며, 만일 검사점이 취해지지 않았을 경우, 다른 프로세스의 강제적 검사점을 필요로 한다. 하지만 쓰레기 탐지는 시간적으로 늦게 처리되어도 전체 시스템의 일관성에 영향을 주지 않는다. 따라서 참여 그래프에 검사점의 발생도 하나의 사건으로 기록하여, 메시지의 부가적 정보로 보낸다. 메시지를 수신한 프로세스는 참여 그래프에 기록된 검사점을 기반으로 자신이 가지고 있는 송신 메시지 내용 로그를 쓰레기 처리할 수 있다. 프로세스들은 자신이 송신한 메시지에 대한 기록을 보존하며, 만일 쓰레기 처리가 되어 있지 않다면 자신의 참여 그래프 기록에서 삭제하지 않는다. 메시지 내용 정보는 그 메시지를 송신한 프로세스에서만 쓰레기 처리되어 질 수 있다.

그림 2에서 다른 프로세스의 결함 포용을 위해 존재하는 메시지 내용 로그 중, 시점 t에서 메시지 1, 6, 7에 대한 로그는 더 이상 필요하지 않다. 하지만 쓰레기 처리가 가능한 메시지를 결정하기 위한 정보가 필요하고, 이 정보는 위에서 제안된 알고리즘과 검사점 정보가 부가된 참여 그래프를 이용하여 얻어진다.

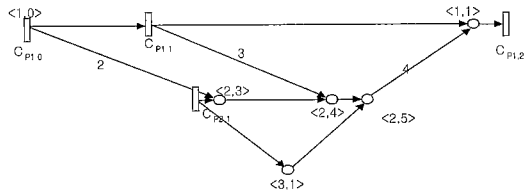


그림 6 P_1의 메시지 내용 로그 쓰레기 처리

그림 6에서 시점 t의 프로세스 P_1은 자신이 P_2로부터 받은 있는 참여 그래프에 메시지 1의 수신 사건이 발생되어 있지 않으므로, 메시지 1의 수신 사건 이후에 검사점이 취해졌다는 것을 알 수 있다. 따라서 메시지 1에 대한 로그는 더 이상 필요치 않다는 것을 알 수 있다.

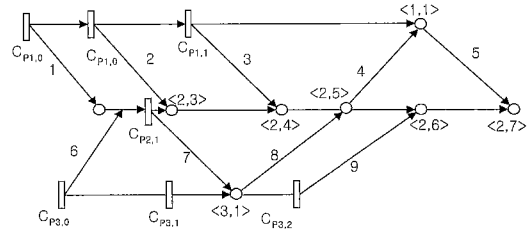


그림 7 P_2의 메시지 순서 쓰레기 처리

그림 7에서 프로세스 P_2의 경우, 프로세스 P_1의 결함 포용을 위한 로그인 메시지 4의 메시지 내용 로그에 대한 로그는 더 이상 필요치 않다는 사실을 알 수가 없다. 그것은 프로세스 P_1의 검사점 $C_{P1,2}$가 취해진 이후에 프로세스 P_1으로부터 메시지를 수신한 적이 없기

때문이다. 하지만 P_3으로부터 메시지 8을 받은 후에, P_3이 검사점 $C_{P3,2}$를 취했다는 것을 알게되고, 이를 통해 메시지 7에 대한 로그를 삭제할 수 있다.

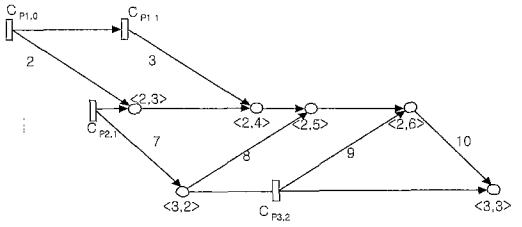


그림 8 P_3의 메시지 순서 쓰레기 처리

그림 8에서 프로세스 P_3의 메시지 6에 대한 메시지 내용 로그는 프로세스 P_2가 검사점 $C_{P2,1}$을 취했다는 것을 안 이후에 삭제될 수 있다.

4.4 쓰레기 정보 탐지 조건의 정당성

4.1절에서 제안된 쓰레기 처리 알고리즘이 [조건 3]과 [조건 4]를 만족하는 메시지 로그에 대해 쓰레기 처리를 수행한다면, 결함의 발생에도 불구하고 시스템의 일관된 상태는 재생성될 수 있다. 이를 증명하기 위해 [정리 1]과 [정리 2]의 쓰레기 탐지 조건에 기반한 쓰레기 처리 알고리즘은 결함이 발생하더라도, 전체 시스템의 일관성을 유지한다는 것을 보인다. 즉, 이 논문에서 제안된 쓰레기 처리 알고리즘으로 인한 부가적 복귀(additional rollback)는 없다는 것을 보인다.

[정리 1] $Chpt_{P_k}(\#_m)$을 만족하는 어떤 프로세스 P_k가 존재하고, 결함 발생시 이 정보를 가지고 있는 모든 프로세스가 [조건 3]의 $garbage(\#_m)$을 만족한다면, 전체 시스템의 일관성은 유지된다.

[증명] : 임의의 프로세스 P_k는 임의의 메시지 m의 메시지 순서 로그를 검사점 $Chpt_{P_k}(\#_m)$과 함께 안전 저장 장치에 보관한다. 따라서 메시지 m에 의존하는 임의의 프로세스에서 발생한 결함에 대해, P_k는 검사점 $Chpt_{P_k}(\#_m)$로 복귀하여, 메시지 m의 메시지 순서 정보를 결함 프로세스에게 보내줄 수 있다. P_k로부터 메시지 순서 정보를 받은 결함 프로세스는 결함 발생 전과 동일한 메시지 순서를 재생성할 수 있다.

[조건 1]에서 $stable(\#_m)$이 참값을 가지게 되면, $Depend(m)$의 원소인 어떤 프로세스에서 결함이 발생하더라도 메시지 m에 의해 발생되는 고아 메시지는 존재하지 않는다. 따라서 MAG에 의해 전파되는 $\#_m$ 정보는 더 이상 불필요하다. 즉, 메시지 m에 대한 $Chpt(\#_m)$ 정보를 갖게 되는 모든 프로세스는 MAG에서 메시지 m에 대한 $\#_m$를 삭제할 수 있다. ■

[정리 1]에 의해 $\#_m$가 프로세스 P_k에서 안전 저장된다면, 프로세스 P_k이외의 프로세스에서 $\#_m$의 유지는 불필요해진다. P_k는 자신이 $\#_m$의 유일한 정보 관리자임을 선출 알고리즘(election algorithm)을 통해 보장받을 수 있으며, 동시에 다른 프로세스에서의 $garbage(\#_m)$이 수행될 수 있다.

[정리 2] $Chpt_{dest_m}(\#_m)$을 만족하고, 이 정보를 가지고 있는 프로세스 $m.source$이 [조건 4]의 $garbage_{m.source}(content_m)$를 만족하면, 전체 시스템의 일관성은 결함이 발생해도 유지된다.

[증명] : 임의의 메시지 m의 메시지 순서 로그에 대해 검사점을 취한 프로세스 $m.dest$은 결함이 발생하더라도 $Chpt_{dest_m}(\#_m)$이후로 복귀하지 않는다. 이것은 $\#_m$의 정보가 $dest_m$에 의해 유지되고 있기 때문이다. 따라서 $m.source$ 에서 $content_m$은 더 이상 필요없게 된다.

즉, [조건 2]에서 $m.dest$에서 검사점을 취하게 되면, $stable(\#_m)$이 참값을 가지게 된다. 따라서 메시지 m에 의존하는 프로세스는 결함이 발생하더라도 $m.dest$에 의해 재생성되어 질 수 있다. 이것은 검사점을 취한 프로세스 $m.dest$는 검사점 이후로 복귀하지 않고, 프로세스 $m.source$에서 유지되고 있는 $content_m$ 정보가 더 이상 필요하지 않다는 것을 의미한다.

결과적으로 $garbage_{m.source}(content_m)$은 결함 발생에도 불구하고 전체 시스템의 일관성을 보장하며, 메시지 내용 로그의 쓰레기 처리로 인한 무의미한 복귀를 발생시키지 않는다. ■

[정리 3]에 의해 메시지 순서 로그의 쓰레기 정보 탐지는 메시지 내용 로그의 쓰레기 정보 탐지에 의존한다는 것이 증명된다. 따라서 메시지 내용 로그의 쓰레기 처리가 완료될 때까지 메시지 순서 로그의 쓰레기 처리는 보류되어야 한다.

[정리 3] 메시지 순서 정보의 쓰레기 처리를 만족하는 집합은 메시지 내용 로그의 쓰레기 처리를 만족하는 집합을 포함하며, $garbage(\#_m) \supset garbage_{m.source}(content_m)$이다.

[증명] : i) *m.dest*에 의해 검사점이 취해졌을 경우

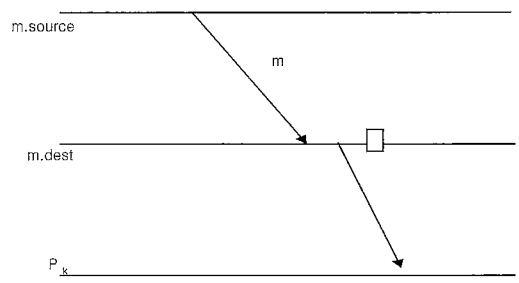


그림 8 시스템 작업 수행도(i)

*m.dest*에서의 검사점은 #_m의 내용을 저장한다면, 결함 발생 후에도 *m.dest*는 검사점 이후로 복귀하지 않는다. 이것은 [정리 2]에서 증명되었다. 따라서 *m.source*에서 유지되는 $content_m$은 더 이상 유지할 필요가 없다. 또한 $\#_m$은 *m.dest*이외의 다른 프로세스에서 유지될 필요가 없다. 즉, $garbage(\#_m) \supset garbage_{m.source}(content_m)$이 성립한다.

ii) *Depend*(*m*)에 속하면서 *m.dest*가 아닌 임의의 프로세스 P_k에 의해 검사점이 취해졌을 경우

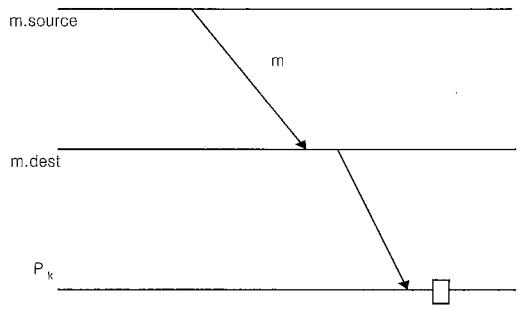


그림 9 시스템 작업 수행도(ii)

P_k에서의 검사점은 $\#_m$의 내용을 저장한다. 하지만 *m.dest*에서 유지되는 $content_m$은 *m.dest*은 결함 발생으로 메시지 *m*의 수신 사건 이후로 복귀할 수 있다. 따라서 $\#_m$은 [조건 2]에 의해 *m.source*에서 유지되어야 하며, $garbage_{m.source}(content_m) = \emptyset$이다. $\#_m$은 P_k의 검사점에서 저장되어 유지되므로 P_k이외의 다른 프로세스에서 중복 유지될 필요가 없다. 즉, $garbage(\#_m) \supset garbage_{m.source}(content_m)$이 성립한다. ■

[정리 3]에 의해 메시지 순서 로그의 쓰레기 처리는 메시지 내용 로그의 쓰레기 처리에 의존한다. 따라서 메시지 순서 로그는 메시지 내용 로그가 안전한 상태에서 쓰레기 처리가 이루어질 수 있을 때까지 보류되어야 한다.

5. 실 험

4장에서 제안된 *MAG*를 이용한 쓰레기 처리 알고리즘이 여러 가지 환경 요소에서 어떻게 반응하는가와 다른 논문에서 제안된 쓰레기 처리 기법과의 성능을 비교 분석한다. 실험을 위해 UCLA의 병렬 컴퓨팅 연구실에서 개발된 PARSEC(PARallel Simulation Environment for Complex systems) 시뮬레이터를 사용했다. PARSEC은 이산 사건 시뮬레이션 언어로써 C-언어에 기초한다. 또한 PARSEC은 프리프로세싱과 컴파일을 위해 C 컴파일러 혹은 GCC 컴파일러를 필요로 하며, 이 실험을 위해 Visual C++ 6.0 컴파일러를 사용했다. 실험을 위한 환경 변수로는 메시지의 수, 메시지 로그의 크기를 선택하였다.

실험을 위한 가정은 다음과 같다.

- 메시지의 발생은 정규 분포를 따른다.
- 모든 메시지의 크기는 동일하다고 가정한다.
- 메시지를 송수신하는 프로세스의 수는 10이며, 각 프로세스는 메시지 로그를 위한 제한된 저장 장치를 갖는다고 가정한다.

정규 분포를 따르는 메시지는 한번의 실험에서 10,000개에서부터 100,000개까지 발생하도록 하였다. 실험에서 메시지를 송수신하는 프로세스의 수를 10개로 고정시킨 이유는 프로세스의 수의 변화가 결국 메시지의 수의 변화와 같기 때문이다. 메시지 로그를 위한 저장 장치의 크기는 메시지의 저장 가능 수로 하였으며, 저장 가능한 메시지의 수는 5부터 30까지로 하였다. 실험의 결과는 쓰레기 처리를 하는 횟수이다. 송수신 메시지 수의 변화에 따라 메시지 로그의 쓰레기 처리가 어떤 경향을 갖는지를 보여준다. 그리고 두 번째 실험에서는 메시지 로그의 크기를 제한함으로써 제안된 쓰레기 처리 기법과 기존의 기법사이의 경향을 비교한다. 실험에서 프로토콜 A는 manetho에서 사용된 쓰레기 처리 기법이며, 프로토콜 B는 이 논문에서 제안된 기법이다.

송신 메시지 수의 변화에 대한 쓰레기 처리 횟수의 변화는 그림 10과 같다. 송신 메시지의 수가 증가함에 따라 메시지 로그에 저장되는 메시지의 수는 증가한다. 따라서 프로토콜 A와 프로토콜 B 모두 증가하는 경향을 보인다. 하지만 프로토콜 B의 경우, 송신 메시지에

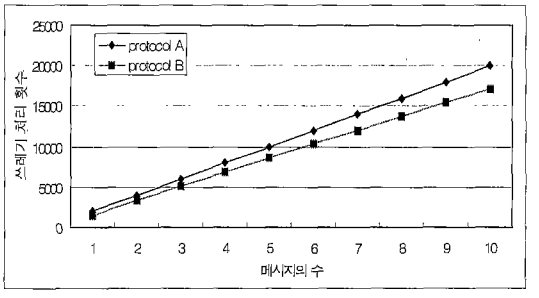


그림 10 송수신 메시지의 수-쓰레기 처리 횟수

대한 메시지 로그의 쓰레기 처리가 두 번 이루어진다. 즉, 메시지를 수신하였을 때, 관련 프로세스로부터 부가되어온 검사점과 송신 메시지 정보를 이용하여 자발적인 쓰레기 처리가 가능하다. 또한 메시지 로그를 위해 할당된 저장 장치가 모두 소모되었을 때, 강압적 쓰레기 처리가 수행된다. 따라서 프로토콜 A에 비해 전체 프로세스의 수행 기간동안 발생되는 강압적 쓰레기 처리 횟수는 프로토콜 B가 적다. 강압적 쓰레기 처리의 경우, 부가적인 메시지를 필요로 하기 때문에 프로세스 성능을 저하시킨다.

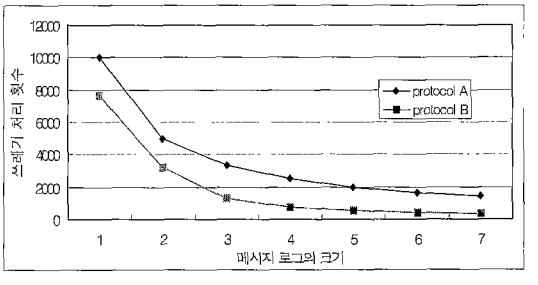


그림 11 메시지 로깅 저장 장치-쓰레기 처리 횟수

메시지 로그에 할당된 저장 장치의 크기에 따른 쓰레기 처리 횟수의 실험 결과는 그림 11과 같다. 그림 11의 실험은 송신 메시지의 수가 50,000개로 고정된 상태에서 각 프로세스의 메시지 로그에 할당된 저장 장치의 크기를 변화시킨 결과이다. 저장 장치의 크기가 5라는 것은 송신 메시지의 로그를 5개까지 저장할 수 있다는 것을 의미한다. 이 실험에서는 송신 메시지를 최대 35개까지 저장할 수 있는 저장 장치까지 가정하였다.

그림 12는 50,000개의 메시지가 발생되며, 메시지 로그의 크기는 10인 경우에 프로토콜 B의 검사점 간격에 따른 쓰레기 처리 횟수의 변화를 보여준다. 프로토콜 B는

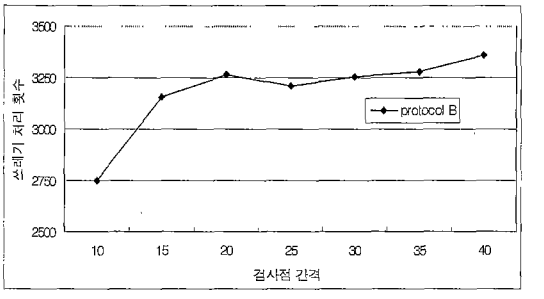


그림 12 검사점 간격에 대한 프로토콜 B의 쓰레기 처리 횟수 변화

검사점 이전의 메시지에 대한 자발적 쓰레기 처리를 수행한다. 따라서 검사점 간격이 커질수록 자발적 쓰레기 처리 발생 빈도는 줄어든다. 검사점 간격이 증가할수록 프로토콜 B가 강제적 쓰레기 처리 횟수를 증가시키지 않기 위해서는 메시지 로그의 크기를 증가시켜야 한다.

6. 결론 및 향후 연구과제

이 논문에서는 인과적 메시지 로깅 기법에서 메시지 내용 로그와 메시지 순서 로그의 쓰레기 처리를 위한 조건을 제안하였다. 기존의 기법에서 결함 포용을 위해 유지하던 메시지 로깅 정보를 메시지 내용 로그와 메시지 순서 로그로 나누어 정의하였으며, 이를 기반으로 메시지 순서 로그의 쓰레기 처리가 메시지 내용 로그의 쓰레기 처리에 의존한다는 것을 증명하였다. 또한 쓰레기 처리 조건을 만족하는 메시지를 찾아내기 위한 정보를 유지하기 위해 Manetho에서 제안된 참여 그래프에 검사점 정보를 기록하여 관리하는 자료 구조로 *MAG*를 제안하였다. *MAG*를 이용하여 부가적인 메시지의 교환없이 메시지 내용 로그와 메시지 순서 로그의 쓰레기 처리가 가능한 기법을 제안하였다. 제안된 기법은 강제적인 쓰레기 처리 횟수를 줄였으며, 전체 시스템의 성능을 향상시켰다.

제안된 기법은 메시지의 송수신 사건이 발생될 때까지의 쓰레기 처리 조건만을 기반으로 한 쓰레기 처리가 이루어진다. 이것은 '지연 쓰레기 처리 현상(lazy garbage collection)'을 발생시켰다. 지연 쓰레기 지연 현상은 전체 시스템의 일관성을 깨뜨리지는 않지만, 송수신 메시지에 부가되는 정보의 양을 증가시킨다. 따라서 현재 지연 쓰레기 처리 현상을 줄이기 위한 최적화 기법에 관한 연구가 진행되고 있다.

참 고 문 헌

[1] R. Koo, S. Toueg, "Checkpoint and Rollback-Recovery for Distributed Systems," *IEEE Trans. S. E.*, Vol. 13, pp.23-31, Jan. 1987.

[2] Sean W. Smith, et al, "Completely Asynchronous Optimistic Recovery with Minimal Rollbacks," *Proc. of IEEE FTCS-25,* 1995, pp361-370.

[3] Sylvain R. Y. Louboutin, Vinny Cahill, "On Thorough Garbage Collection in distributed Systems," *IEEE*, pp. 576-581, 1998.

[4] M. V. Sreenivas, Subhash Bhalla, "Garbage Collection in Message Passing Distributed Systems," *IEEE,* pp. 213-218, 1995.

[5] Lorenzo Alvisi, Keith Marzullo. "Message Logging: Optimistic, Causal and Optimal," *In Pro IEEE Int. Conf. Distributed Computing Systems,* pages 229-236, March 1995.

[6] Lorenzo Alvisi, Bruce Hoppe, Keith Marzullo. "Nonblocking and orphan-free message logging protocols," *In Proceedings of 23rd Fault-Tolerant Computing Symposium,* pages 145-154, June 1993.

[7] E. L. Elnozahy, W. Zwanepoel. "Manetho: Transparent rollback-recovery with low overhead," limited rollback and fast output commit. *IEEE Transactions on Computers,* 41(5):526-531, March 1992.

[8] Mootaz Elnozahy, Lorenzo Alvisi, Yi-Min Wang, David B. Johnson. "A Survey of Rollback-Recovery Protocols in Message-Passing Systems," *Technical Report CMU-CS-96-181. Department of Computer Science. Carnegie Mellon University.* Sept. 1996.

[9] Jian Xu, Robert H. B. Netzer, Milon Mackey, "Sender-based Message Logging for Reducing Rollback Propagation," *IEEE*, pp.602-609, 1995.

[10] D. Manivannan, Mukesh Singhal, "A Low-Overhead Recovery Technique Using Quasi-Synchronous Checkpointing," *Proceedings of the 16th ICDCS*, pp100-107. 1996.

[11] D. B. Johnson, W. Zwaenepoel, "Sender-based Message Logging," *In Digest of papers:17 Annual International Symposium on Fault-Tolerant Computing*, 14-19, IEEE Computer Society, June 1987.

정 광 식

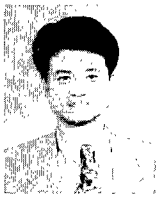
1993년 고려대학교 컴퓨터학과 학사. 1995년 고려대학교 컴퓨터학과 석사. 1995년 ~ 현재 고려대학교 컴퓨터학과 박사과정. 관심분야는 분산시스템, 이동 컴퓨팅 시스템, 결함포용시스템

유 헌 창

1989년 고려대학교 이과대학 컴퓨터학과 졸업. 1991년 고려대학교 대학원 컴퓨터학과 졸업(이학석사). 1994년 고려대학교 대학원 컴퓨터학과 졸업(이학박사). 1995년 ~ 1997년 서경대학교 이공대학 컴퓨터공학과 조교수. 1998년 ~ 현재 고려대학교 사범대학 컴퓨터교육과 조교수. 관심분야는 분산 시스템, 이동 컴퓨팅 시스템, 결함 포용 시스템, 웹기반교육

황 종 선

1978년 Univ. of Georgia, Statistics and Computer Science 박사. 1978년 South Carolina Lander 주립대학교 조교수. 1981년 한국표준연구소 전자계산실 실장. 1995년 한국정보과학회 회장. 1982년 ~ 현재 고려대학교 컴퓨터학과 교수. 1996년 ~ 현재 고려대학교 컴퓨터과학기술대학원 원장. 관심분야는 알고리즘, 분산시스템, 데이타베이스 등



손 진 곤

1984년 고려대학교 이과대학 수학과졸업. 1988년 고려대학교 대학원 전산학 석사학위취득. 1991년 고려대학교 대학원 전산학 박사학위취득. 1991년 ~ 현재 한국 방송통신대학교 컴퓨터과학과 교수. 관심분야는 분산시스템, 컴퓨터통신망, 원격교육, 컴퓨터 보안, Petri nets 등

백 맹 순

1998년 고려대학교 수학교육과 졸업. 1999년 ~ 현재 고려대학교 컴퓨터학과 석사과정. 관심분야는 분산시스템, 결함 포용, 이동에이전트