인과적 메시지 전달 순서를 기반한 낙관적 메시지 로깅 기법을 위한 효율적 회복 알고리즘*

백맹순° 김기범 안진호 황종선
고 려 대 학 교 컴 퓨 터 학 과 분 산 시 스 템 연 구 실
{msbak, kibom, jhahn, hwang}@disys.korea.ac.kr

An Efficient Recovey Algorithm for Optimistic Message Logging Based on Causal Order of Message Delivery

MaengSoon Baik° Kibom Kim Jinho Ahn ChongSun Hwang
Dept. of Computer Science & Engineering, Korea University

요 약

급격한 통신의 발달은 분산시스템의 메시지 전달 환경을 FIFO 뿐만 아니라, 인과적 순서로도 가능케 한다. 낙관적 메시지 로깅 기법은 프로세스의 결함을 저비용으로 회복할 수 있다. 기존의 낙관적 메시지 로깅기법은 인과적 메시지 전달 순서를 보장해주는 환경에 적용되었을 때, 회복과정시 각 프로세스가 전체 프로세스에게 회복메시지를 브로드캐스팅함으로써 불필요한 오버헤드를 가진다. 그러나 제안하는 알고리즘은 회복과정을 시작하고 종료하는 시점에서 조정자 프로세스만 전체 프로세스에게 회복메시지를 브로드캐스팅함으로써 이러한 오버헤드를 줄일 수 있다. 즉, 조정자 프로세스가 아닌 프로세스는 정상수행시 유지하고 있는 상태정보에 기반하여 자신과 의존성이 존재하는 프로세스에게만 회복메시지를 선택적으로 멀티캐스팅함으로써 회복과정시 필요한 메시지의 수를 줄이고 전체적인 회복비용을 감소시킨다.

1. 서론

분산시스템은 통신 네트워크에 의해서 서로 연결되고, 이를 통해 메시지들을 주고 받는 프로세스들의 집합체이다. 각 프로세스는 분산 응용프로그램의 하나의 수행 주체가 된다. 그러나 장시간(long time) 수행되어야 하는 응용프로그램의 경우 프로세스들 중에서 하나라도 결함(failure)이 발생하게 되면 전체 응용프로그램이 원하는 결과를 산출해 내지 못하는 잠재적 위험성이 존재하게 된다. 그러므로 각 프로세스가 자신의 상태정보를 주기적으로 저장하고, 프로세스의 결함 발생시 이를 이용하여 응용프로그램을 재수행(replaying)하는 복귀회복(rollback recovery) 기법은 분산시스템에 결함-포용성(fault-tolerance)을 제공해 주는 매력적인 기법이다[1, 5]. 복귀회복기법에서 각 프로세스는 이후에 발생할 지 모르는 결함에 대비하여 자신의 상태 정보를 안정된 저장소(stable storage)에 저장하게 되고, 이 때 저장된 상태 정보를 검사점(checkpoint)이라 한다[5]. 그러나 검사점만을 사용하는 복귀회복 기법은, 결함 발생시 일관된 검사점들의 집합으로 복귀하여 재수행하기 때문에 검사점 이후의 상실되는 작업의 양은 무시할 수 없다. 그래서 상실되는 작업의 양을 줄이기 위해 검사점기법에 메시지로깅(message logging)기법을 결합한다. 이처럼 분산시스템의 잠재적 결함발생 가능성을 대비하기 위해 검사점과 메시지 로깅을 결합한 결함 포용 기법은 널리 사용되고 있다[1, 2, 3, 5].

그러나 최근의 통신환경의 발전은 기존의 시스템에서는 보장해주지 못한 인과적 메시지 전달 순서를 가능하게 하였으며, 이를 기반으로 하는 응용프로그램이 등장하게 되었다[4, 6].

우리는 정상수행시의 오버헤드가 가장 적은 낙관적 메시지 로깅 기법에 대한 연구를 수행한 결과 기존의 동기적 회복기법을 사용하는 낙관적 메시지 로깅 기법들을 FIFO보다 강력한 인과적 메시지 전달 순서(causal message delivery)를 보장해주는 시스템에 적용하였을 경우 회복 과정시 불필요한 오버헤드(overhead)가 발생함을 발견했다. 이에 본 논문에서는 인과적 메시지 전달 순서를 보장해주는 시스템에 적합한 낙관적 메시지 로깅 기법에서의 동기적 회복기법을 제안한다. 기존의 FIFO 기반의 기법들은 회복과정에서 각 프로세스가 전체프로세스에게 회복메시지를 브로드캐스팅하는 기법을 사용하였다[1, 3]. 그러나 제안하는 회복알고리즘은 회복과정에서 조정자 프로세스를 제외한 프로세스는 자신에게 의존성이 존재하는 프로세스에게만 회복메시지를 선택적으로 멀티캐스팅(multicasting)한다. 이러한 방법은 상대적으로 비용이 많이 드는 브로드캐스팅을 시작과 끝에만 수행하게 함으로써 전체 회복비용을 줄일 수 있다.

* 이 연구는 정보통신연구진흥원 대학기초연구지원사업(과제번호:2000-024-01)인 "Mobile IP 환경에서 이동 호스트를 위한 최적의 회복 프로토콜의 설계"의 지원으로 수행되었음.

2. 시스템 모델

시스템은 응용프로그램을 수행하는 프로세스와 프로세스들간의 통신채널로 구성되어진다. 각 프로세스는 독립적으로 자신의 검사점을 취하며 검사점 이후에 입력되는 메시지들은 일단 각 프로세스의 휘발성 메모리에 로깅된다. 그리고 나서 프로세스는 가장 적합한 시점에 입력 메시지와 입력 메시지로 인해 발생한 상태구간의 정보를 안정된 저장소에 저장한다. 각 프로세스는 정상 수행시 자신이 취한 검사점들간의 일관성 유지를 위한 책임은 없다. 프로세스들간의 통신채널은 인과적 메시지 전달 순서를 보장해 준다. 인과적 메시지 전달 순서를 정형적으로 표현하면 다음과 같다[3, 6].

$$\forall p, q, r, m, m' : send_p(m) \rightarrow send_q(m') \Rightarrow deliver_r(m) \rightarrow deliver_r(m')$$

여기서 $send_p(m)$은 프로세스 p가 프로세스 r에게 메시지 m을 보낸다는 것을 의미하며 $deliver_r(m)$은 프로세스 r이 메시지 m을 응용 프로그램에게 전달한다는 의미이다. →는 Lamport가 정의한 사건들간의 전후관계(happened-before)를 나타낸다[3]. 또한 시스템은 임의의 프로세스에게 결함이 발생하면 자동으로 이를 탐지하고 결함이 발생한 프로세스를 안정된 저장소에 있는 정보를 이용하여 복구시킨다. 복구된 프로세스는 조정자(coordinator) 역할을 하게되며 다른 프로세스에게 결함이 발생하였음을 알리게 된다. 결함이 발생하였음을 인식한 프로세스들은 자신의 수행을 중지시키고 현재까지 휘발성 메모리에 있는 내용을 안정된 저장소에 저장하고 나서 제안된 회복 알고리즘을 이용하여 프로세스들간의 일관된 상태를 찾는 과정을 수행한다. 시스템내의 모든 프로세스들은 일관된 상태가 찾아지면 그 지점까지 복귀하여 이후의 작업을 수행하게 된다. 이때 회복 알고리즘을 이용하여 일관된

상태를 찾는 과정에서의 결함발생 가능성은 배제한다. 만약에 회복 과정에서 결함이 발생하게 되면 회복 알고리즘을 처음부터 다시 수행하면 된다.

3. 제안된 회복 알고리즘

3.1 동기

기존의 낙관적 메시지 로깅 기법에서의 동기적 회복기법들을 인과적 메시지 전달 순서를 기반으로 하는 시스템에 그대로 적용하면 그림 1 (a)와 같은 회복과정을 보인다.

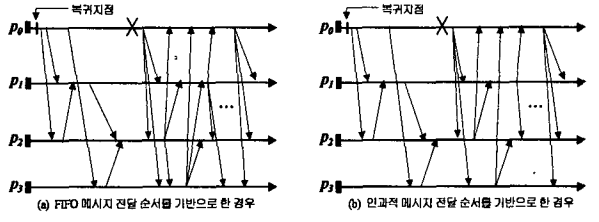

그림 1. 낙관적 메시지 로깅의 회복과정

결함이 발생하였음을 알리는 메시지를 받은 각 프로세스 p_1, p_2, p_3는 자신의 회복지점을 설정하고 다시 전체 프로세스에게 브로드캐스팅을 하게된다. 그리고 다른 프로세스가 브로드캐스팅한 메시지로 인해 회복지점이 변경될 때 역시 전체 프로세스에게 자신의 회복지점을 브로드캐스팅하게 된다.

그러나 인과적 메시지 전달 순서를 보장해주는 시스템에서는 그림 1 (a)에서처럼 각 프로세스가 전체 프로세스에게 브로드캐스팅을 할 필요가 없다. 회복과정에서의 브로트캐스팅은 조정자 프로세스에 의해 시작과 종료시 두 번으로 충분하다. 나머지 수행과정에서의 메시지 전송은 조정자 프로세스로 인해 고아상태(orphan state)가 발생한 프로세스가 자신의 상태와 의존성이 존재하는 프로세스에게만 선택적으로 멀티캐스팅하면 된다. 그림 1 (b)에서 보면 프로세스 p_1은 정상수행시 자신과 의존성이 있는 프로세스 p_2에게만 자신의 현재 회복지점을 알리는 메시지를 보내게 된다. 마찬가지로 프로세스 p_2와 p_3도 정상수행시 자신과 의존성이 있는 프로세스 p_1과 p_2에게만 자신의 현재 회복지점을 알리게 된다. 이러한 방법은 비용이 많이 드는 브로드캐스팅을 두 번으로 줄임으로써 전체 회복비용을 현저하게 감소시킬 수 있다.

3.2 자료구조

각 프로세스는 소문자 p로 표기하며, 프로세스의 식별자는 p의 아래첨자로 나타낸다. 또한 프로세스 p_i의 k번째 상태 구간은 p_i^k로 표기한다. 프로세스 p_i가 결함 발생시 회복을 위해 유지해야 하는 자료구조는 다음과 같다.

- DeV_i: 프로세스 p_i는 다른 프로세스와 의존성을 나타내는 의존벡터 DeV_i를 유지한다. 이 의존벡터를 통해 각 프로세스는 자신의 상태구간이 다른 프로세스의 어느 상태구간과 의존하고 있는지 여부를 파악할 수 있다. 의존벡터 DeV_i^k는 프로세스가 메시지를 받아 새로운 상태구간 k을 수행할때마다 갱신되는 의존벡터이다.
- $SeSp_i^k$: 프로세스 p_i는 자신의 k번째 상태동안 메시지를 보낸 프로세스들의 집합을 유지한다.
- DeM: 결함이 발생하여 회복알고리즘의 적용시 조정자 프로세스 p_{cor}은 의존행렬 DeM을 유지하고 각 프로세스의 임시 복귀지점이 일관된 상태구간 집합을 구성하는지 여부를 검사한다. 이때 의존행렬의 각 행들은 조정자 프로세스를 포함한 각 프로세스들이 유지하고 있는 $TdRp_i$에서의 의존벡터 DeV_i가 되며 각 프로세스에서 새로운 DeV_i^k이 올때마다 DeM은 갱신된다.
- $TdRp_i$: 회복과정시 프로세스 p_i는 자신이 복귀해야 하는 상태구간 지점을 나타내는 포인터 변수 $TdRp_i$를 유지하고 다른 프로세스와의 의존성 여부에 따라 이를 계속 갱신시켜 간다.

3.3 정상 수행시

프로세스 p_i는 자신의 상태 구간 k에서 메시지 m을 생성하여 프로세스 p_j에게 보낼때 자신의 프로세스 식별자 i와 상태구간 숫자 k를 같이 보낸다. 그리고 자신의 $SeSp_i^k$에 프로세스 p_j를 첨가한다. 프로세스 p_j는 메시지 m을 일단 자신의 휘발성 메모리에 저장하고 기존의 의존 벡터 DeV_j^l를 DeV_j^{l+1}로 갱신한다. 의존벡터 DeV_j^l의 갱신과정은 다음과 같다. 먼저 프로세스 p_j가 현재 유지하고 있는 벡터 DeV_j^l에서의 i번째 원소와 메시지 m에서 첨부된 프로세스 p_i의 상태구간 숫자 k를 비교한다. k가 자신의 i번째 원소보다 크다면 의존벡터의 i번째 원소를 k로 대체하고 j번째 원소인 l을 하나 증가시켜 새로운 의존벡터 DeV_j^{l+1}을 얻게된다. 그렇지 않고 k가 자신의 의존벡터 DeV_j^l의 i번째 원소와 같다면 의존벡터의 l번째 숫자를 하나 증가시켜서 DeV_j^{l+1}를 얻게 된다. 입력되는 k가 자신의 DeV_j^l의 i번째 원소보다 작은 경우는 인과적 메시지 전달 순서를 위반하는 것이므로 시스템에서 발생하지 않는다. 이후 프로세스 p_j는 새로운 상태구간 $1+k$를 수행하게 된다.

3.4 회복시 알고리즘

회복 알고리즘은 2단계로 수행된다. 첫 번째 단계에서는 각 프로세스가 협력하여 일관성 있는 각각의 회복지점을 결정하게된다. 그러나 실제로 이 과정에서 복귀하는 것은 아니며 단지 전체 프로세스가 복귀지점에 대한 협의와 결정만을 하게 된다. 첫 번째 단계가 완전히 수행된 후 두 번째 단계가 수행되는데 이 단계에서 각 프로세스는 복귀 지점까지 복귀하게 되고 응용 프로그램을 재수행하게 된다. 복귀 지점이 결정되면 각 프로세스의 복귀와 재 수행은 독립적으로 진행된다.

(1) 복귀지점에 대한 협의와 결정

① 조정자 프로세스

결함이 발생한 프로세스 p_{cor}은 시스템에 의하여 복구된 이후 조정자 프로세스로 설정되며 자신의 안정된 저장소에 있는 정보를 이용하여 최대한 복귀할 수 있는 지점 $TdRp_{cor}$를 선택한다. 그리고 모든 프로세스에게 결함이 발생하였음과 현재의 $TdRp_{cor}$의 상태구간숫자 s_{cor}를 갖는 메시지 $m_{start-recovery}$을 브로드캐스팅한다. 브로드캐스팅이 끝난 p_{cor}은 전체 프로세스로부터 의존벡터 DeV를 받아서 자신의 의존벡터 DeV_{cor}^s와 각 프로세스의 DeV가 하나의 행으로 구성된 DeM을 유지한다. 이후 새로운 DeV_i^k가 도착할 때나 자신의 $TdRp_{cor}$가 변화하여 해당 DeV_{cor}^s를 갱신할때마다 새로운 DeV_i^k와 $DeV_{cor}^{s'}$로 DeM을 갱신한다. 조정자 프로세스 p_{cor}은 DeM의 생성과 갱신시에 일관성 검사를 수행한다. 일관성 검사는 해당 DeM이 다음 조건을 만족하는지를 판별하는 것이다.

$\forall i,j \;\; DeM_{i,i} \geq DeM_{j,i} \Rightarrow DeM \text{ is consistent}$

만약에 각 프로세스의 상태들간에 일관성이 발견되면 복귀 과정을 종료시키고 그때 DeM에서 유지하고 있는 각 DeV_i^k의 해당 프로세스의 상태 구간 숫자 k를 프로세스들에게 보내고, 자신의 회복지점을 나타내는 $TdRp_{cor}$가 다른 프로세스와 일관된 상태임을 최종 결정한다.

```
[Coordinator algorithm]
begin
  TdRp_cor ← the latest event number s_cor of p_cor from stable storage;
  broadcast  an start-recovery message(id( p_cor ), s_cor ) to all process;
  wait for an recovery message(id, DeV) from all process;
  create  DeM  from first recovery message from all process and  DeV_cor^s ;
  compute  DeM ;
  while ( ∃i,j  DeM_{i,i} < DeM_{j,i} ) {
        wait for an recovery message (id, DeV) from process;
        upgrade( DeM );
        compute  DeM ; }
  find recovery line from  DeM ;
  send terminate message to all process;
end;
```

그림 2. 조정자 프로세스의 회복 알고리즘

② 비조정자 프로세스

결함이 발생하였음과 조정자 프로세스가 유지하고 있는 상태구간의 숫자 s_{corr}를 받은 각 프로세스 p_i는 자신의 수행을 종료하고 휘발성 메모리에 있는 입력 메시지와 상태정보들을 안정된 저장소에 저장한다. 이후 조정자 프로세스의 상태구간숫자를 참조하여 조정자 프로세스의 고아 상태와 의존성이 없도록 $TdRp_i$를 설정한다. 만약 $TdRp_i$가 k번째 상태구간이라면, 조정자 프로세스에게 현재 자신의 $TdRp_i$가 가리키고 있는 상태구간에서의 DeV_i^k를 보낸다. 그리고 나서 상태구간 k 이후에 메시지를 보냈던 프로세스의 집합 $SeSp_i^k$를 이용하여 해당 프로세스에게 자신의 상태구간 숫자를 전송해준다. 이후에 다른 프로세스들로부터 상태구간숫자를 받음으로 인해서 고아상태가 발생하지 않도록 $TdRp_i$가 갱신되었을 때는 해당 상태구간 k'에서의 $DeV_i^{k'}$를 다시 조정자 프로세스에게 보낸다. 이러한 과정을 조정자 프로세스로부터 회복과정의 종료를 알리는 메시지가 도착할 때까지 계속한다. 조정자 프로세스로부터 회복과정의 종료를 알리는 메시지와 자신의 상태구간 숫자를 받으면 다른 프로세스와 일관된 상태구간을 최종 결정하고 회복 알고리즘을 종료한다.

```
[Non-coordinator algorithm]
begin
  receive start-recovery message(id( p_cor), s_cor) from coordinator process;
  save the latest state information to stable storage;
  TdRp_i ← the latest stable event number from stable storage and
  start-recvery message(id( p_cor), s_cor) from coordinator process;
  send an recovery message(id, DeV) to coordinator process p_cor;
  while( ! receive terminate message( p_i^k) from coordinator process p_cor)
    {
      wait receive message(id( p_j), s_TdRp_j) from neiborhood process;
      TdRp_i ← the latest stable event number from stable storage;
      send an recovery message(id, DeV) to coordinator process p_cor;
      send an recovery message(id, s_TdRp_i) to dependent process;
    }
  find rollback point from terminate message;
end;
```

그림 3 비조정자 프로세스의 회복 알고리즘

(2) 복귀지점으로의 복귀와 재수행

복귀지점을 발견한 각 프로세스는 복귀지점까지 복귀하고 다시 응용 프로그램을 재수행한다. 각 프로세스의 복귀와 재수행은 독립적으로 이루어진다.

만약에, 복수개의 프로세스가 결함이 발생하면 일단 결함이 발생한 프로세스들은 자신의 안정된 상태까지 복구되고 각 프로세스의 고유한 id에 부여된 우선순위를 통해서 조정자 프로세스를 선출한다. 이후의 과정은 단일한 프로세스에서 결함이 발생한 경우와 동일하다.

4. 정당성

보조정리 1. 프로세스 p_i에서 $TdRp_i^j < TdRp_i^k$ $(j \neq k)$인 $TdRp_i^j$가 존재한다면 프로세스 p_i는 고아상태구간 s_k를 가지게 된다.

증명: 프로세스 p_i가 복귀지점 설정의 과정에서 두 개의 $TdRp_i$를 가지게 된다는 것은 처음 설정한 $TdRp_i^k$에서의 상태구간이 고아 메시지에 의존하고 있다는 것을 가리킨다. 그러므로 프로세스 p_i가 $TdRp_i^j < TdRp_i^k$ $(j \neq k)$인 $TdRp_i^j$을 가진다면 이전의 $TdRp_i^k$이 나타내는 상태구간은 고아 상태가 된다는 것을 의미한다. ■

보조정리 2. 조정자 프로세스 p_{cor}이 같은 프로세스 p_i로부터 $DeV_i^j \rightarrow DeV_i^k$의 순서로 두 개의 의존벡터를 받았다면 항상 DeV_i^j는 다른 프로세스의 고아상태에 의존한 상태이다.

증명: 프로세스 p_i는 조정자 프로세스나 다른 프로세스의 안정된 상태구간 숫자를 받아서 자신의 상태구간이 안정된지 여부를 검사한다. 만약에 자신의 상태구간이 다른 프로세스의 고아상태에 의존하게 되면 자신의 $TdRp_i$를 가장 마지막의 안정된 상태구간의 숫자를 가리키게 하고 이 상태구간에서의 DeV_i^k를 조정자 프로세스에게 보내게 된다. 그러므로 조정자 프로세스가 동일한 프로세스 p_i로부터 두 개의 의존벡터 DeV_i^j와 DeV_i^k를 받았다면 이 중에 먼저 도착한 의존벡터 DeV_i^j는 다른 프로세스의 고아상태구간에 의존하고 있다는 것을 나타낸다.■

정리 1. 조정자 프로세스 p_{cor}에서 유지하고 있는 의존행렬 DeM에서 $\forall i,j \ \ DeM_{i,i} \geq DeM_{j,i}$ 조건을 만족할 때 전체 프로세스들은 일관성있는 복귀 지점을 결정할 수 있다.

증명: 정리 1의 증명은 대우를 사용하여 보일 수 있다. 먼저 전체 프로세스들이 서로 일관성이 존재하지 않는다고 가정한다면 프로세스들이 유지하고 있는 상태들 중의 최소한 하나는 다른 프로세스의 고아 상태에 의존하게 된다. 이를 각 프로세스가 해당 상태구간에서 의존하고 있는 의존벡터 DeV_i^k의 관계로 살펴보면 프로세스 p_i의 의존벡터 DeV_i^k와 다른 프로세스 p_j의 의존벡터 DeV_i^l에서 프로세스 p_i의 현재 상태구간은 p_j의 상태구간 숫자보다 더 큰 l에 의존하고 있다. 이를 의존행렬로 구현하면 p_j의 상태구간 숫자는 최소한 p_i에서 유지하고 있는 의존벡터의 DeV_i^k의 j번째 성분보다 작게 된다. 이는 주어진 조건 $\forall i,j \ \ DeM_{i,i} \geq DeM_{j,i}$을 만족시키지 못한다. ■

5. 관련연구

분산시스템에서의 낙관적 메시지 로깅 기법들 중에 결함이 발생하였을 때 동기적으로 이를 회복하려는 연구는 D. B. Johnson[1]과 S. Venkatesan[3]에 의해서 이루어져 왔다. 그러나 이 두 연구 모두다 FIFO 메시지 전달 순서를 기반으로 하여 진행되어 왔기에 인과적 메시지 전달 순서를 보장해주는 시스템에서는 불필요한 오버헤드를 가지고 있다. D. B. Johnson의 경우 복귀 지점을 산정하는데 있어서 각 프로세스가 유지하고 있는 의존벡터를 전체 프로세스들에게 브로드캐스팅한다. 또한 S. Venkatesan의 경우 각 프로세스는 자신이 받은 메시지와 보낸 메시지의 총 숫자를 비교하여 복귀지점을 선정한 다음 전체 프로세스에게 브로드캐스팅하게된다. 이러한 기법들은 공통적으로 각 프로세스가 전체 프로세스에게 브로드캐스팅하는 방법을 선택하는데 이는 전체 회복시간을 지연시키는 원인이 된다. 제안된 회복기법은 브로드캐스팅을 단 두 번으로 줄임으로써 회복비용을 줄이게 된다.

6. 결론 및 향후 연구

통신기술의 발전은 FIFO보다 더 강력한 인과적 메시지 전달 순서를 보장해주게 되었다. 낙관적 메시지 로깅 기법에서의 동기적 회복기법을 제공하는 이전의 회복 알고리즘들은 회복과정에서 각 프로세스가 자신과 의존성이 없는 프로세스라 할지라도 전체 프로세스에게 회복메시지를 브로드캐스팅해야 한다. 그러나 이러한 방법은 FIFO보다 강력한 인과적 메시지 전달 순서를 보장해주는 시스템에서는 불필요한 오버헤드가 된다. 제안된 회복알고리즘은 회복과정에서 비조정자 프로세스가 브로드캐스팅이 아닌 자신과 의존성이 존재하는 프로세스들에게만 회복메시지를 선택적으로 멀티캐스팅을 함으로써 조정자 프로세스에 의한 두 번의 브로드캐스팅만으로 전체 회복기법을 종료할 수 있다. 앞으로 제안된 알고리즘과 기존의 알고지즘에 대한 정상수행시와 회복시 정확한 성능평가가 이루어져야 할 것이다.

7. 참고문헌

[1] D. B. Johnson, "Recovery in distributed systems using optimistic message logging and checkpointing," *Journal of Algorithms*, Vol. 11, No. 3, pp. 462-491, Sept. 1990.

[2] L. Alvisi and K. Marzullo, "Message logging: Pessimistic, optimistic, causal and optimal," *In IEEE Transactions on Software Engineering*, Vol. 24, No. 2, pp. 149-159, Feb. 1998.

[3] S. Venkatesan, "Optimistic crash recovery without changing application messages." *In IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 3, pp. 263-271, Mar. 1997.

[4] A. Acharya and B. R. Badrinath, "Recording distributed snapshots based on causal order of message delivery," *In Information Processing Letters*, pp. 317-321, Dec. 1992.

[5] E.N. Elnozahy, D.B. Johnson and Y.M. Wang, "A Survey of Rollback-Recovery Protocols in Message Passing Systems," *CMU Tecnical Report CMU-CS-99-148*, June. 1999.

[6] S. Alagar and S. Venkatesan, "Causal Ordering in Distributed Mobile Systems," *IEEE Transactions on Computers*, Vol. 46, No. 3, pp. 353-361, March. 1997.