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

정 광식, 황 승희, 유 헌창, 황 종선
{cks,shhwang}@disys.korea.ac.kr, yuhc@comedu.korea.ac.kr, hwang@disys.korea.ac.kr

Garbage Collection of Message Log without Additional Message on Causal Message Logging Protocol

Kwang-Sik Chung , Seung-Hee Hwang, Heon-Chang Yu, Jong-Sun Hwang
Dept. of Computer Science & Engineering, Korea University
* Dept. of Computer Science Education, Korea University

요 약 이 논문은 인과적 메시지 로깅 기법에서 결함 포용을 목적으로 완전 저장 장치(stable storage)에 저장되는 메시지 로그와 메시지 순서 로그의 쓰레기 처리 기법을 제안한다. 메시지 로그와 메시지 순서 로그는 메시지 순서 정보에 부가되는 검사점 정보를 기반으로 쓰레기 처리되어질 수 있으며, 이를 위해 메시지 로그와 메시지 순서 로그의 쓰레기 처리 조건을 제시한다. 제시된 조건을 기반으로 한 메시지 로그와 메시지 순서 로그의 쓰레기 처리는 송수신 메시지에 부가된 정보를 이용하므로 제안된 알고리즘은 '지연 쓰레기 처리 현상(lazy garbage collection)'를 발생시킨다. 하지만 '지연 쓰레기 처리 현상'은 분산 시스템의 일관성을 위배하지 않으며, 쓰레기 처리를 위한 부가적인 메시지 교환을 필요로 하지 않는다.

1. 서론

네트워크 기술과 분산 시스템의 발달로 인해 하나의 작업이 여러 프로세스에서 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 포용 기법은 동기적 검사점을 기반으로 하는 기법과 비결정적 사건의 로그와 비동기적 검사점을 기반으로 한 기법으로 나뉠 수 있다. 메시지 로깅 기법은 메시지 로그와 비결정적 사건에 대한 정확한 순서를 저장함으로써, 결함이 발생한 프로세스는 결함 발생 이전의 상태를 재생성할 수 있다. 이러한 메시지 로그 기법은 어떻게 메시지 로그를 취하느냐에 따라, 비관적 메시지 로깅 기법(pessimistic message logging protocol), 낙관적 메시지 로깅 기법(optimistic message logging protocol), 인과적 메시지 로깅 기법(causal message logging protocol)으로 구별할 수 있다. 비관적 메시지 로깅 기법에서는 각 프로세스가 비결정적 사건이 완전 저장 장치(stable storage)에 저장되는 동안, 작업 수행(failure-free operation)을 멈춘다. 낙관적 메시지 로깅 기법은 정상 작업 성능의 비관적 메시지 로깅 기법에 비해 향상시켰지만, 고아 메시지 발생과 회복 비용, 쓰레기 처리 비용 그리고 외부 결과 출력 비용면에서 약점을 가지고 있다. 인과적 메시지 로깅 기법은 비결정적 사건의 로그를 완전 저장 장치에 저장하지 않고, 여러 프로세스의 불완전 저장 장치(volatile storage)에 저장함으로써 결함 발생에도 불구하고 메시지 로그를 유지한다. 또한 완전 저장 장치에 비동기적으로 접근함으로써 낙관적 메시지 로깅 기법의 장점을 유지한다. 그러나 결함 회복, 쓰레기 처리 및 통신 대역폭 면에서 위의 두 기법에 비해 상대적으로 큰 비용을 필요로 한다[2,3].

기존의 인과적 메시지 로깅 기법에서는 메시지 로그의 쓰레기 처리를 위해 부가적인 메시지 교환을 통해 쓰레기 처리되어야할 메시지 로그를 결정하였다. 이 논문에서는 인과적 메시지 로깅 기법에서 메시지 로그의 쓰레기 처리를 위해 필요한 조건을 새롭게 정의하며, 이를 기반으로 논문[1]에서 제안된 참여 그래프(antecedence graph)를 새롭게 정의하여 쓰레기 처리 비용을 향상시켰다. 이 논문의 2장에서는 메시지 로그의 쓰레기 처리를 위한 조건을 정의한다. 3장에서는 기존의 인과적 메시지 로깅 기법에서 수행되었던 메시지 로그의 쓰레기 처리 기법의 문제점과 연구 수행 방법에 대해 언급한다. 4장에서는 이 논문에서 새롭게 제안되는 메시지 로그의 쓰레기 처리 기법을 논문[1]에서 제안된 기법과 비교하여 제안한다. 5장에서는 결론과 향후 연구 과제에 대해 언급한다.

2. 관련 연구

인과적 메시지 로깅 기법을 사용하는 Manetho에서 프로세스 p는 상태 간격(state interval) σ_i^p이전에 보내진 메시지 로그의 쓰레기 처리를 결정할 수 있다. 즉, Manetho에서 프로세스는 마지막 검사점이전으로 복귀(rollback)하지 않으므로, 프로세스 p가 프로세스 p에게 메시지를 보낼 때, q가 실제로 그 메시지를 받았지만 아직 검사점을 취하지 않았다면, p는 q에게 검사점을 취할 것을 요청한다. 모든 프로세스들이 이 검사점 요청을 받은 후에, 프로세스 p는 안전하게 σ_i^p이전에 보내진 모든 메시지를 지울 수 있다. 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 순서 로그의 쓰레기 처리를 위해 강제적 검사점을 요구하고, 메시지 로그의 쓰레기 처리를 위해 부가적인 메시지의 교환을 필요로 한다. 이러한 비용을 줄이기 위해 이 논문에서는 메시지 로그를 세분화하여 정의하였으며, 이를 쓰레기 처리하기위한 조건을 제안하였다. 또한 메시지 순서 정보에 검사점 정보를 부과하여 메시지 순서 정보(참여 그래프)를 새롭게 구성하고, 이 정보를 기반으로 메시지 로그와 메시지 순서 로그를 쓰레기 처리하는 조건을 제안한다.

3. 시스템 모델

분산 시스템 N은 n개의 프로세스로 구성된다. N의 수행 단위를 ρ 라 하며, ρ 동안 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[2].

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

$Depend(m)$은 m이 수신 프로세스에 반영된 후 발생한 송신 메시지 m'이 반영된 프로세스들과 메시지 m이 반영된 수신 프로세스

의 합집합을 의미하며, $Depend(m)$에 포함되는 프로세스는 메시지 m에 의존한다.

메시지는 결함 포용 정보 관리의 관점에서 두 가지로 분류되어 관리된다. 먼저 메시지에 의해 전달되는 응용 프로그램의 메시지 내용이다. 이것은 $contents_m$으로 나타낸다. 두 번째로 수신 프로세스의 상태 전이에 영향을 미치며, 메시지의 송수신 순서를 나타낸다. 이것은 $\#_m$으로 나타내며, 각각 다음과 같이 정의된다.

[정의 1] 메시지 정보와 메시지 순서 정보

메시지 순서 정보 : $\langle m.source,\ m.ssn,\ m.rsn \rangle$
$\Rightarrow$ *determinant of event deliver* $_{m.dest}(m)$
$\Rightarrow \#\, m$
메시지 정보 : $\langle m.data,\ m.ssn,\ m.rsn \rangle$
$\Rightarrow$*contents of message m*
$\Rightarrow contents_m$

효율적인 결함 포용 정보 관리를 위해 필요한 메시지 관련 정보를 다음과 같이 정의한다.

$$Fault-T(m) \stackrel{\text{def}}{=} \{contents_m\} \bigcup \{Depend(m)\}$$

집합 N에서 결함이 발생한 프로세스의 집합 C 중 고아 프로세스 p가 다음과 같이 정의된다.

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

$Log(\#_m)$은 $\#_m$의 복사본을 불완전 저장 장치에 가지고 있는 프로세스들의 집합이며, 메시지 순서 로그라 한다. $Log(contents_m)$은 $contents_m$의 복사본을 불완전 저장 장치에 가지고 있는 프로세스들의 집합이며, 메시지 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.

다음은 인과적 메시지 로깅 기법에 의해 프로세스가 고아 메시지가 되지 않는 조건이며, 이 조건을 만족하는 프로세스로 구성된 시스템은 결국 일관성을 유지한다.

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

$\#_m$의 복사본을 완전 저장 장치에 저장한 프로세스의 집합을 $stable(\#_m)$이라고 한다. f개까지의 결함 발생을 포용하기 위해 다음과 같은 조건을 만족하면, 인과적 메시지 로깅 기법은 전체 시스템의 일관성을 유지시킬 수 있다.

또한 결함 발생에 대한 회복 프로토콜의 성능을 향상시키기 위해 송신된 메시지의 내용이 저장되는 조건은 다음과 같다.

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

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

다음은 $\#_m$의 쓰레기 처리 조건이다.

$$\exists m, \exists P_k : (deliver_{m.dest}(m) \bigwedge Chpk_{P_k}(\#_m)) \Rightarrow garbage(\#_m) \quad (1)$$

다음은 $contents_m$의 쓰레기 처리 조건이다. 수신 프로세스의 검사점이 취해졌을 때 메시지의 송수신 순서 정보에 표시된 검사점 정보를 기반으로, 검사점 이전의 메시지 로그는 쓰레기 처리되어 질 수 있다.

$$\exists m : (deliver_{m.dest}(m) \bigwedge Chpk_{m.dest}(\#_m)) \Rightarrow garbage_{m.source}(contents_m) \quad (2)$$

$Chpk_{m.dest}(\#_m)$는 프로세스 $m.dest$가 메시지 m의 송수신 순서 정보를 포함하는 검사점을 취하는 사건이며, $garbage_{m.source}(contents_m)$는 프로세스 $m.source$가 메시지 m의 데이터를 제거시키는 사건이다.

메시지 순서 정보의 쓰레기 처리를 만족하는 집합은 메시지 로그의 쓰레기 처리를 만족하는 집합을 포함하며, $garbage(\#_m) \supset garbage_{m.source}(contents_m)$이다. 따라서 메시지 순서 로그는 메시지 로그의 쓰레기 처리에 의존한다.

4. 결함 포용 정보의 쓰레기 처리

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 로그와 메시지 송수신 순서 정보인 참여 그래프의 쓰레기 처리를 위해 부가적인 메시지의 전달을 필요로 한다. 쓰레기 처리를 위한 부가적 메시지의 발생을 줄이기 위해 메시지 순서 정보에 메시지 송수신 순서 외에 검사점 발생 사건도 프로세스의 상태 전이 사건으로 기록한다. 검사점 발생 사건이 부가된 메시지 송수신 순서 정보는 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.

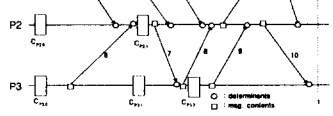
[정의 1] MAG(Modified Antecedence Graph)

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

다음은 송신 프로세스 p_j에 대한 수신 프로세스 p_i에서의 쓰레기 처리 알고리즘이다.

```
Big_MAG ← MAG_{p_i} ∪ MAG_{p_i(m)};
while(p_k ∈ Depend(m))
  Big_MAG ← garbage_collection(Big_MAG, MAG_{p_k(ckpt)});
MAG_{p_i} ← Big_MAG;

Big_MAG ← MAG_{p_i} ∪ MAG_{p_j(m)};
while(p_k ∈ Depend(m)) ∧ ( e_{k,i} ∉ Big_MAG))
stable(contents_m)
      ← garbage_collection (stable(contents_m), contents_m);
*k, i ∈ p_k's and  p_k's state interval
```



<그림 1> 시스템 작업 수행도

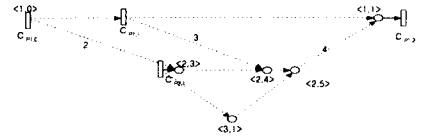
4.2 메시지 순서 로그 쓰레기 처리

Manetho의 경우 참여 그래프의 쓰레기 처리를 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 그래프내에 검사점 사건을 기록하여

부가적인 검사점 인덱스 메시지 교환을 방지한다.

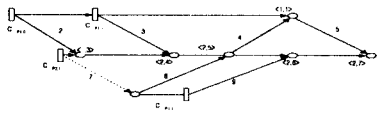
메시지 순서 정보는 어느 프로세스라도 할 수 있다. 따라서 위에서 제시된 조건에 만족하는 모든 메시지 순서 정보는 쓰레기 처리되어 질 수 있다.

<그림 1>에서 다른 프로세스의 결함 포용을 위해 존재하는 메시지 로그 중, 시점 t에서 메시지 1, 2, 3, 4, 6, 7, 8에 대한 로그는 더 이상 필요하지 않다.



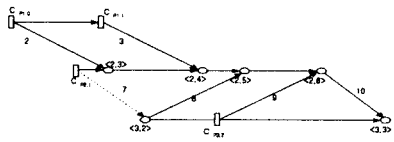
<그림 2> P_1의 메시지 순서 정보 쓰레기 처리

프로세스 P_1은 <그림 2>에서처럼 참여 그래프의 표시된 검사점 $Cp_{2,1}$로부터 메시지 1에 대한 로그는 더 이상 필요치 않다는 것을 알 수 있다. 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 3> P_2의 메시지 순서 쓰레기 처리

시점 t에서 메시지 2, 3, 4, 7, 8의 메시지 순서 로그는 더 이상 필요없다. 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 4> P_3의 메시지 순서 쓰레기 처리

프로세스 P_3의 메시지 6에 대한 메시지 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 로그 쓰레기 처리

Manetho의 경우 메시지 로그의 쓰레기 처리를 위해 다른 프로세스로부터 검사점 확인(Ack)을 받아야하며, 만일 검사점이 취해지지 않았을 경우, 다른 프로세스의 강제적 검사점을 필요로 한다. 하지만 쓰레기 처리는 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 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 1>에서 다른 프로세스의 결함 포용을 위해 존재하는 메시지 로그 중, 시점 t에서 메시지 1, 4, 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 2>에서 시점 t의 프로세스 P_1은 자신이 P_2로부터 받은 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 3>에서 프로세스 P_2의 경우, 프로세스 P_1의 결함 포용을 위한 로그인 메시지 4의 메시지 로그에 대한 로그는 더 이상 필요치 않다는 사실을 알 수가 없다. 그것은 프로세스 P_1의 검사점 $C_{P1,2}$가 취해진 이후에 프로세스 P_1로 부터 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 4>에서 프로세스 P_3의 메시지 6에 대한 메시지 로그는 프로세스 P_2가 검사점 $C_{P2,1}$을 취했다는 것을 안 이후에 삭제될 수 있다.

4.4 쓰레기 처리 조건

4.1절에서 제안된 쓰레기 처리 알고리즘은 식 (1),(2)를 만족한다면, 결함의 발생에도 불구하고 시스템의 일관된 상태를 재생성할 수 있다. [정리 1]과 [정리 2]는 식 (1)과 (2)로부터 생성된 조건이다.

[정리 1] $Chpk(\#_m)$을 만족하는 어떤 프로세스 P_k가 존재하고, 결함 발생시 이 정보를 가지고 있는 모든 프로세스의 $garbage(\#_m)$은 전체 시스템의 일관성을 유지해 준다.

[정리 2] $Chpk_{dest_m}(\#_m)$을 만족하고, 이 정보를 가지고 있는 프로세스 $m.source$의 $garbage(contents_m)$은 전체 시스템의 일관성을 유지해 준다.

5. 결론 및 향후 연구과제

이 논문에서는 인과적 메시지 로깅 기법에서 메시지 로그와 메시지 순서 로그의 쓰레기 처리를 위한 조건을 제안하였다. 메시지 로깅 정보를 메시지 로그와 메시지 순서 로그로 나누어 정의하였으며, 이를 기반으로 메시지 로그의 쓰레기 처리가 메시지 순서 로그의 쓰레기 처리를 포함한다는 것을 보였다. 또한 쓰레기 처리 조건을 만족하는 정보를 유지하기 위해 Manetho[1]에서 제안된 참여 그래프에 검사점 정보를 기록하여 관리함으로써, 부가적인 메시지의 송수신없이 메시지 로그와 메시지 순서 로그의 쓰레기 처리가 가능하도록 하였다.

이 논문에서 제안된 메시지 로그와 메시지 순서 로그의 쓰레기 처리 기법은 Manetho의 쓰레기 처리 기법에 비해 부가적인 메시지를 필요로 하지 않는다. 하지만, 메시지의 송수신 사건이 발생될 때까지 쓰레기 처리가 늦어지는 지연 쓰레기 처리 현상이 발생한다.

현재 지연 쓰레기 처리를 줄이기 위한 기법과 메시지 로그와 메시지 순서 로그의 쓰레기 처리 알고리즘의 성능 실험이 진행되고 있다.

6. 참고 문헌

[1] 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.

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

[3] 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.

[4] 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.