PWD 모델에 기반한 효율적인 조정검사점 기법*

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

Efficient Coordinated Checkpointing Scheme based on PWD Model

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

요 약

본 논문에서는 PWD 모델을 기반하였을 경우 검사점들에 대한 새로운 일관성 조건이 필요함을 보이고, PWD 모델에 적합한 조정검사점 기법을 제안하고자 한다. 제안된 조정검사점 기법은 전체 프로세스가 일관된 검사점집합을 구성할 때 기존의 일관성 조건을 따르는 것이 아니라 PWD 모델에 적합한 새로운 일관성 조건을 따른다. 또한 각 프로세스의 수행상태를 비결정적 사건으로 인해서 발생하는 상태구간으로 구분하여 이전의 검사점 이후에 변화된 상태구간에 새로운 의존성을 생성하는 프로세스만 검사점을 취한다. 제안된 기법은 PWD 모델에 기반한 시스템에서 기존의 조정검사점 기법이 보이는 불필요한 오버헤드를 없애고, 결함발생시 시스템의 제한된 복귀를 보장한다.

1. 서론

분산시스템에는 임의의 프로세스에게 결함(failure)이 발생하면 프로세스 의존성(dependency)으로 인해 전체 응용프로그램이 원하는 결과를 산출해내지 못하는 잠재적 위험성(potential risk)이 존재한다[1]. 이를 해결하기 위해 프로세스가 자신의 상태정보를 안정된 저장소(stable storage)에 저장하고, 결함발생시 이를 이용하여 회복가능한 상태에서 응용프로그램을 재수행하는 복귀회복(rollback-recovery)기법은 분산시스템에 결함포용(fault-tolerance)을 제공해주는 기법으로서 많은 연구가 있어왔다[5].

복귀회복기법은 크게 검사점(checkpoint)만을 사용하는 검사점기반 회복기법과 프로세스에서 발생하는 비결정적(non-determi nistic) 사건들을 로그(log)하여 이용하는 메시지 로깅기법으로 구분 된다[2, 5]. 일반적으로 메시지 로깅기법에는 독립검사점 기법을 사용하는 것이 더 좋다는 관련연구들이 있었지만 최근의 연구결과들은 조정검사점 기법과 메시지 로깅기법의 결합이 수행능력(performance)과 단순성(simplicity)에 있어서 뛰어나다는 것을 보이고 있다[2]. 그러나 관련연구 [2]는 PWD 모델을 기반하지 않은 시스템에서 제안된 조정검사점 기법을 그대로 사용함으로써 PWD 모델에 대한 아무런 고려 없이 검사점을 취해왔다. 또한 관련연구 [4, 6]들은 PWD 모델 특성에 맞추어 메시지 로깅기법을 변형시켰지만, 이 역시 불필요한 검사점을 추가로 취하는 오버헤드는 제거하지 못했다.

본 논문에서는 PWD 모델을 기반하였을 경우 검사점들에 대한 새로운 일관성 조건(consistency condition)이 필요함을 보이고, PWD 모델에 적합한 조정검사점 기법을 제안하고자 한다.

2. 시스템 모델과 동기

시스템은 응용프로그램을 수행하는 프로세스와 프로세스들간의 통신채널(communication channel)로 구성된다[5]. 통신채널은 신뢰성 있다고 가정하고, FIFO 기반의 메시지 전달 순서(delivery order)를 보장한다[5]. 각 프로세스의 수행은 PWD 모델을 따른다. 프로세스에서 발생하는 모든 비결정적 사건은 메시지의 전달(delivery)로 제한한다. PWD 모델을 기반한 경우 각 프로세스의 수행은 초기상태(initial state)와 수신된 메시지의 전달순서(delivery order)에 의해서 결정된다[2]. 또한 프로세스는 결함발생시 자신의 수행을 정지하고[5], 시스템은 프로세스의 결함을 자동으로 인지하며 안정된 저장소에 있는 정보를 이용하여 결함이 발생한 프로세스를 회복시킨다.

기존 조정검사점 기법의 가장 중요한 고려사항은 각 프로세스가 고아메시지(orphan message)를 생성하지 않으면서 검사점을 취하는 것이다. 고아 메시지란 수신 프로세스에서는 메시지를 전달한 사건이 기록되어 있는 반면에 송신 프로세스에서는 메시지를 송신한 사건이 기록되지 않은 메시지를 의미한다[1].

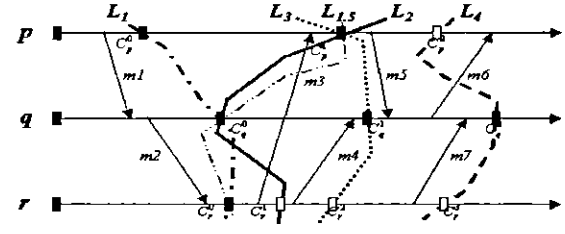

그림 1 PWD 모델을 기반하지 않는 시스템에서의 일관된 검사점집합

그림 1을 보면 라인 L_1에서 프로세스 p, q, r은 일관적 검사점 C_p^0, C_q^0, C_r^0를 유지하고 있다. 이후 프로세스 p가 새로운 검사점 C_p^1을 취함으로 인해 검사점들의 집합은 라인 L_1에서 $L_{1.5}$로 발전한다. 그러나 프로세스 r에서 송신된 메시지 $m3$의 전달사건이 프로세스 p의 검사점 C_p^1에는 기록되어 있는 반면, 프로세스 r의 검사점 C_r^0에는 메시지 $m3$의 송신사건이 기록되어 있지 않기 때문에 메시지 $m3$는 고아 메시지가 되고, 결국 라인 $L_{1.5}$는 비일관적인 검사점집합이 된다. 메시지 $m3$가 고아메시지가 되지 않게 하기 위해서 프로세스 r은 새로운 검사점 C_r^1을 취함으로써 검사점들의 집합은 라인 $L_{1.5}$에서 L_2로 발전되고, 라인 L_2는 고아 메시지를 만들지 않으므로 일관된 검사점집합이 된다. 이후 라인 L_3와 L_4 역시 각각 C_r^2 그리고 C_p^2와 C_r^3를 취해야만 일관된 검사점집합이 된다.

그러나 각 프로세스의 수행이 PWD 모델을 기반 할 경우 라인 L_2에서 취해지는 검사점 C_r^1과 라인 L_3에서 취해지는 C_r^2 그리고 라인 L_4에서 취해지는 검사점 C_p^2과 C_r^3는 불필요한 검사점이 된다. 이는 PWD 모델을 기반한 시스템에서는 비결정적 사건으로 인해서 발생하는 상태구간에서의 송신사건들이 결정적으로 재수행될 수 있기

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

때문이다.

3. 제안된 조정검사점 기법

3.1 검사점의 일관성 조건

PWD 모델을 기반한 시스템에서는 검사점의 일관성 조건이 PWD 모델을 기반하지 않은 시스템에서의 조건과 달라진다. 그림 2에서 라인 L_1은 일관된 검사점집합으로 구성되지만, 두 번째 라인 PwL_2는 PWD 모델을 기반하지 않는 시스템에서는 메시지 $m5$와 $m7$이 고아메시지가 되기 때문에 일관된 검사점집합을 구성할 수 없다.

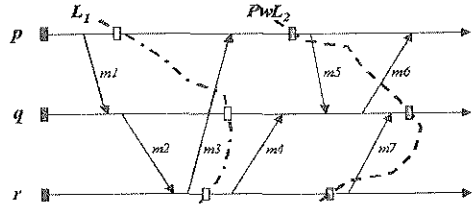

그림 2 일관된 검사점집합의 구성

그러나 PWD 모델을 기반하는 시스템에서는 라인 PwL_2 역시 일관된 검사점집합을 구성한다. 왜냐하면 프로세스 p는 메시지 $m5$을 송신한 상태구간을 생성한 메시지 $m3$에 대한 결정자를 자신의 검사점에 기록하고 있기 때문에 메시지 $m5$의 전송을 포함한 상태구간 s^{1}_{p}를 결정적으로 재수행 할 수 있기 때문이다. 이러한 조건은 프로세스 r도 마찬가지로 메시지 $m2$에 대한 결정자가 검사점 안에 포함되어 있으므로 메시지 $m7$에 대한 송신사건을 포함한 상태구간 s^{1}_{q}을 결정적으로 재수행할 수 있기 때문에 결국에는 프로세스 q가 유지하고 있는 비결정적 사건 메시지 $m5$, $m7$에 대한 전달을 결정적으로 보장해 준다. PWD 모델을 기반하는 시스템에서의 검사점들간의 일관성 조건을 정형적으로 나타내면 다음과 같다.

C1 $\forall\ p, q \quad deliver(m) \in C_p \Rightarrow Det(ss_q(m)) \in C_q$

iff C_p *is consistent with* C_q

여기서 $ss_q(m)$는 메시지 m을 송신한 프로세스 q의 상태구간이며 $Det(ss_q(m))$는 상태구간 $ss_q(m)$를 생성시킨 메시지의 결정자이다. 결국 PWD 모델을 기반한 시스템에서 일관된 검사점집합 $\sum_{pwd}(C_{p_i})$은 다음과 같이 나타낼 수 있다. 단, 각 프로세스 p_i에서 하나의 검사점 C_{p_i}만이 유효하다고 가정한다.

Dfn 1 $\sum_{pwd}(C_{p_i}) = \forall i \bigcup_{i \in Id} C_{p_i}$

(C_{p_i} *satisfy* C1, *Id: a set of process identification number*)

3.2 자료구조

N개의 프로세스로 구성된 시스템에서 각 프로세스는 소문자 p로 표기하며, 프로세스의 식별자는 p의 아래첨자로 나타낸다. 프로세스 p_i의 k번째 상태구간은 $s^{k}_{p_i}$로 표기한다. 프로세스 p_i가 정상수행시 회복을 위해 유지하는 자료구조는 다음과 같다.

· $DeV^{k}_{p_i}$: 프로세스 p_i는 자신의 k번째 상태구간과 의존성을 가지는 다른 프로세스의 상태구간을 판별하기 위하여 각 구성성분이 해당 프로세스의 상태구간 숫자를 나타내는 인과적 의존벡터 $DeV^{k}_{p_i}$를 유지한다[3].

· ε_{p_i}: 프로세스 p_i는 현재 자신이 유지하고 있는 검사점이 속해있는 상태구간 숫자를 가리키는 변수 ε_{p_i}를 유지한다.

· $CP_{count}(p_i)$: 각 프로세스 p_i는 자신이 취한 검사점이 시스템 차원에서 몇 번째로 취해진 일관된 검사점집합의 원소인지를 나타내기 위한 변수 $CP_{count}(p_i)$를 유지한다.

3.3 조정검사점 알고리즘

제안하는 검사점 알고리즘은 기존연구 [1]의 방법을 따르면서 앞에서 제시한 C1의 조건을 만족시키는 검사점을 취함으로써 전체 시스템 차원에서 취해지는 검사점의 수를 줄인다. 각 프로세스는 자신이 보내는 모든 메시지에 자신의 상태구간에서 유지하고 있는 의존벡터를 첨부한다. 또한 메시지를 받고 처리하기 이전에 메시지에 첨부된 의존벡터와 자신의 의존벡터간에 갱신을 하고 메시지 로그에 상태정보로 유지한다.

① 조정자 프로세스 p_i

검사점을 취하고자 하는 프로세스 p_i는 최근의 검사점 이후부터 현재의 상태구간까지를 기록하여 임시검사점을 취한다. 그리고 ε_{p_i}가 가리키는 상태구간과 임시검사점이 취해진 상태구간 $s^{\delta}_{p_i}$의 의존벡터를 비교하여 검사점을 취해야만 하는 프로세스를 결정한다.

$$forced\_checkpoint(p_j) = \begin{cases} p_j & if\ (DeV^{\varepsilon_{p_i}}_{p_i})_j < (DeV^{\delta}_{p_i})_j \\ \text{"No"} & otherwise \end{cases} \quad (j \neq i)$$

이후 프로세스 p_i는 최근의 검사점 이후에 새로운 의존성을 생성한 프로세스 p_j에게 임시검사점을 취하라는 요구메시지에 자신이 유지하고 있는 변수 $CP_{count}(p_i)$의 값을 하나 증가시켜 첨부하여 보낸다. 조정자 프로세스는 요구메시지를 보낸 모든 프로세스로부터 확인메시지(*ack-message*)를 받을 때까지 대기한 후에 각 프로세스에게서 확인메시지를 받으면 임시검사점을 고정검사점으로 전환하고 검사점이 취해진 상태구간을 가리키는 변수 ε_{p_i}를 현재 고정검사점이 취해진 상태구간을 가리키도록 갱신한다. 또한 변수 $CP_{count}(p_i)$의 값을 하나 증가시킨다. 이후 각 프로세스가 취한 임시검사점을 고정검사점으로 전환하라는 결정메시지(*decision-message*)에 증가된 변수 $CP_{count}(p_i)$를 첨부하여 전체 프로세스에게 브로드캐스팅(broadcasting)한다. 만약에 조정자 프로세스가 요구메시지를 보낸 모든 프로세스에게서 확인메시지를 받지 못했다면 자신의 임시검사점을 취소하고 증가된 $CP_{count}(p_i)$의 값을 하나 감소시킨다. 그리고 변화된 $CP_{count}(p_i)$를 첨부하여 요구메시지를 보낸 프로세스에게만 새로이 취한 임시검사점을 취소하라는 결정메시지를 보낸다.

Coordinator process p_i

Do

take a tentative checkpoint in state interval $s^{\delta}_{p_i}$;

Var($CP_{count}(p_i)$)=Var($CP_{count}(p_i)$) + 1;

compare($DeV(s^{\varepsilon_{p_i}}_{p_i})$, $DeV(s^{\delta}_{p_i})$);

$forced\_checkpoint(p_j) = \begin{cases} p_j & if\ (DeV^{\varepsilon_{p_i}}_{p_i})_j < (DeV^{\delta}_{p_i})_j \\ \text{"No"} & otherwise \end{cases} \quad (j \neq i)$;

for ($forced\_checkpoint(p_j)$!= "No")

  send *request-message*(p_j, $CP_{count}(p_i)$);

wait(*ack-message* from $\forall$ $forced\_checkpoint(p_j)$);

if receive *ack-message*($\forall$ $forced\_checkpoint(p_j)$) **then**

  make tentative checkpoint permanent;

  Var($CP_{count}(p_i)$)=Var($CP_{count}(p_i)$) + 1;

  $\varepsilon_{p_i} = s^{\delta}_{p_i}$;

  broadcast *decision-message*(permanent-decision, $CP_{count}(p_i)$);

fi;

else *undo* tentative checkpoint;

  Var($Ch_{count}(p_i)$)=Var($Ch_{count}(p_i)$)-1;

  send *decision-message*($\forall$ $forced\_checkpoint(p_j)$, discard-decision, $CP_{count}(p_i)$);

oD;

Non-coordinator process p_i

Upon receipt of *request-message*(p_k, $CP_{count}(p_i)$);

Do

if $CP_{count}(p_i) \geq CP_{count}(p_j)$ **then**

  if $CP_{count}(p_i) > CP_{count}(p_j)$ **then**

    $CP_{count}(p_j) = CP_{count}(p_i)$;

    if $\varepsilon_{p_i} = s^{\delta}_{p_i}$ **then**

      send *ack-message*(p_k, ack);

    fi;

    else

      take a tentative checkpoint in state interval $s^{\delta}_{p_i}$;

      compare($DeV(s^{\varepsilon_{p_i}}_{p_i})$, $DeV(s^{\delta}_{p_i})$);

      $forced\_checkpoint(p_r) = \begin{cases} p_r & if\ (DeV^{\varepsilon_{p_i}}_{p_i})_r < (DeV^{\delta}_{p_i})_r \\ \text{"No"} & otherwise \end{cases} \quad (r \neq j)$;

      for ($forced\_checkpoint(p_r)$!= "No")

        send *request-message*(p_r, $CP_{count}(p_i)$);

      wait(*ack-message* from $\forall$ $forced\_checkpoint(p_r)$);

      if receive *ack-message*($\forall$ $forced\_checkpoint(p_r)$) **then**

        send *ack-message*(p_k, ack);

      fi;

  fi;

  else if $CP_{count}(p_i) = CP_{count}(p_j)$ **then**

    send *ack-message*(p_k, ack);

  fi;

  wait *decision-message* from process p_k or p_i;

  if receive *decision-message*(p_i, permanent-decision, $CP_{count}(p_i)$) **then**

    $CP_{count}(p_j) = CP_{count}(p_i)$;

    make tentative checkpoint permanent;

    $\varepsilon_{p_i} = s^{\delta}_{p_i}$;

  fi;

```
else if receive decision-message( p_k, discard-decision, CP_count(p_i)) then
        undo tentative checkpoint;
        CP_count(p_j)=CP_count(p_i);
        if ¬1p_r
                send decision-message( ∀p_r, discard-decision, CP_count(p_i));
        fi;
    fi;
else send ack-message( p_k, Non_ack);
oD;

Non_participated process p_x
Do
Upon receipt of decision-message( p_i, permanent-decision, CP_count(p_i));
CP_count(p_x)=CP_count(p_i);
oD;
```

그림 3 조정검사점 알고리즘

② 비조정자 프로세스 p_j

요구메시지를 받은 각 프로세스 p_j는 자신의 검사점이 속해있는 상태구간 ε_{p_j}와 현재 수행중인 상태구간 $s^{\delta}_{p_j}$을 비교하여 수행을 결정한다.

- $\varepsilon_{p_j} < s^{\delta}_{p_j}$: 만약에 최근의 검사점이 취해진 상태구간 ε_{p_j}이후에 상태구간 전이를 이루어 새로운 상태구간 $s^{\delta}_{p_j}$을 수행하고 있다면 현재의 상태구간에서 임시검사점을 취하고 $CP_{count}(p_j)$의 값을 $CP_{count}(p_i)$의 값으로 갱신한다. 이후 ε_{p_j}가 가리키는 검사점 이후에 새로운 의존성을 생성하는 프로세스들에게 검사점을 취하라는 요구메시지를 보낸다. 프로세스 p_j는 자신과 의존성이 존재하는 프로세스들에게서 확인메시지를 받고 나서야 조정자 프로세스에게 확인메시지를 보낸다.
- $\varepsilon_{p_j} = s^{\delta}_{p_j}$: 최근의 검사점이 취해진 상태구간 ε_{p_j}와 현재의 상태구간 $s^{\delta}_{p_j}$가 같다면 프로세스 p_j는 검사점 이후에 상태구간 전이를 하지 않았기 때문에 $CP_{count}(p_j)$를 $CP_{count}(p_i)$로 갱신하고 즉시 요구메시지를 보낸 프로세스에게 확인메시지를 보낸다.

이후 조정자 프로세스 p_i로부터 임시검사점을 고정검사점으로 전환하라는 결정메시지를 수신하면 다음과 같이 수행한다.

- $\varepsilon_{p_j} < s^{\delta}_{p_j}$: 결정메시지를 받은 프로세스 p_j는 현재 취한 임시검사점을 고정검사점으로 전환하고 ε_{p_j}를 갱신한다. 또한 고정메시지에 첨부된 $CP_{count}(p_i)$로 자신의 $CP_{count}(p_j)$를 갱신한다.
- $\varepsilon_{p_j} = s^{\delta}_{p_j}$: 단지 첨부된 $CP_{count}(p_i)$로 자신의 $CP_{count}(p_j)$를 갱신한다.

그렇지 않고 임시검사점을 취소하라는 결정메시지를 수신하면 다음과 같이 수행한다.

- $\varepsilon_{p_j} < s^{\delta}_{p_j}$: 자신의 임시검사점을 취소시키고 첨부된 $CP_{count}(p_i)$로 자신의 $CP_{count}(p_j)$를 갱신한다. 그리고 자신이 요구메시지를 보냈던 프로세스에게 결정메시지를 송신한다.
- $\varepsilon_{p_j} = s^{\delta}_{p_j}$: 단지 첨부된 $CP_{count}(p_i)$로 자신의 $CP_{count}(p_j)$를 갱신한다.

③ 검사점 알고리즘에 참여하지 않은 프로세스 p_x

검사점 알고리즘에 참여하지 않은 프로세스 p_x가 조정자 프로세스에게서 결정메시지를 받는 경우는 알고리즘이 정상적으로 종료했다는 것을 의미한다. 이는 일관적인 검사점집합이 발전하였음을 의미하기 때문에 수신된 결정메시지에 첨부된 $CP_{count}(p_i)$로 자신이 유지하고 있는 $CP_{count}(p_x)$를 갱신한다.

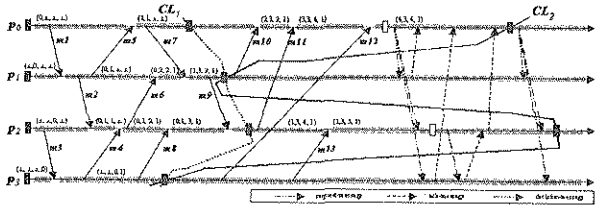

그림 4 제안된 조정검사점 알고리즘의 예

그림 3은 제안된 검사점 알고리즘을 표현한 것이다. 만약에 한 프로세스가 검사점 알고리즘을 조정자 프로세스로서 수행하고 있는 중에 다른 프로세스에게서 검사점 알고리즘 참여를 요청하는 메시지를 받거나, 또는 비조정자 프로세스가 알고리즘 종료에 대한 확인 없이 동시에 두 개의 알고리즘 참여 메시지를 받는 경우는 시스템 시작시 부여된 프로세스 우선순위에 따라 우선순위가 높은 프로세스에서 시작된 알고리즘을 수행한다. 그림 4는 제안된 조정검사점 알고리즘의 실제 사례를 보여준다.

3.4 정당성 증명

보조정리 1 *임의의 프로세스 p_i가 검사점 알고리즘에 참여하여 새로운 검사점을 생성하였을 경우 이 검사점은 일관적인 검사점이다.*

증명: 지면관계상 생략. ■

보조정리 2 *임의의 프로세스 p_i가 검사점 알고리즘에 참여하여 기존의 검사점을 그대로 유지하는 경우 이 검사점은 일관적인 검사점이다.*

증명: 지면관계상 생략. ■

보조정리 3 *임의의 프로세스 p_i가 알고리즘에 참여하지 않고 현재의 검사점을 그대로 유지하고 있다면 이 검사점은 일관적인 검사점이다.*

증명: 지면관계상 생략. ■

정리 1 *임의의 프로세스 p_i가 유지하고 있는 검사점 C_{p_i}는 일관된 검사점집합 $\sum_{p_i \in \Gamma}(C_{p_i})$의 원소이다.*

증명: 지면관계상 생략. ■

3.5 결함발생시 회복

결함이 발생한 프로세스는 시스템에 의해서 최근의 검사점이 존재하는 상태구간까지 회복한다. 이후 전체 프로세스에게 결함발생 사실을 알리는 복귀메시지(*rollback-message*)를 브로드캐스팅한다. 검사점 이후에 재수행이 가능한 상태구간을 결정하는 것은 사용되는 메시지 로깅기법에 따라 달라진다.

낙관적 메시지 로깅기법을 사용시는 재수행을 시작하기 이전에 각 프로세스간에 일관된 전역상태(consistently global state)를 결정하는 과정이 선행되어 한다[5]. 그러나 비관적 메시지 로깅기법을 사용시는 프로세스간의 어떠한 동기화과정 없이 결함발생 직전의 상태까지 재수행을 할 수 있다[5].

4. 결론 및 향후 연구과제

본 논문에서는 프로세스가 최근의 검사점 이후에 상태구간 전이를 수행하지 않은 상태에서는 추가로 검사점을 취하지 않고도 다른 프로세스의 검사점과 일관성 조건을 만족한다는 사실을 보였다. 또한 이러한 조건을 기반으로 하였을 경우 기존연구에서 취해지는 검사점의 횟수보다 적은 검사점으로 구성되는 새로운 조정검사점 기법을 제안하였다. 새로운 기법에서는 기존연구에서 보이는 불필요한 오버헤드를 제거하였으며, 결함발생시 각 프로세스의 제한된 복귀를 보장한다. 그러나 본 논문에서 제안한 조정검사점 기법은 검사점의 횟수를 줄이는 만큼 결함발생 이후에 각 프로세스가 재수행해야 하는 작업의 양은 많아질 수밖에 없다. 앞으로는 검사점의 횟수를 줄임으로써 얻어지는 수행능력 향상과 재수행해야 하는 작업량의 증가로 감수해야하는 오버헤드에 대한 정확한 성능평가가 이루어질 것이다.

참고문헌

[1] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *IEEE Trans. on Sortware Engineering*, Vol. SE-13, No. 1, pp. 23-31, Jan. 1987.

[2] E. N. Elnozahy and W. Zwaenepoel, "On the use and implementation of message logging", *In Proceedidngs of the 24th International Symposium on Fault-Tolerant Computing(FTCS-24)*, pp. 298-307, Jun 1994.

[3] Ravi Prakash, Mukesh Singhal, "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 7, No. 10, pp 1035-1048, Oct. 1996.

[4] Nitin H. Vaidya, "Staggered Consistent Checkpointing," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 10, No. 7, pp. 694-702, July 1999.

[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] J. Fowler and W. Zwaenepoel, "Causal Distributed Breakpoints", *Proc. Int'l Conf. on Distributed Computing Systems*, pp. 134-141, May 1990.