Rollback과 Roll-forward 기법을 사용한 TMR 고장의 시간여분 복구 정책

윤 재 영† · 김 학 배††

요 약

날로 더욱 복잡해지는 제어컴퓨터의 고신뢰성 확보를 위해 시간여분(time redundancy)의 일종인 rollback과 roll-forward 기법 등을 TMR 구조에 적절하게 혼용하는 두가지 방법을 제안한다. Rollback과 roll-forward 기법은 약간의 추가시간만을 사용하여 재구성(reconfiguration) 없이도 일시적인 결함(fault)에 의해 발생한 TMR 고장(failure)의 회복을 위해 상호보완적으로 사용될 수 있다. 본 논문은 시스템의 고장상태확률의 유도를 통해 전체 작업의 평균실행시간이 최소가 되는 최적의 checkpoint 간격벡터를 유도한다. 수치예제를 통해 제안된 방법들이 다른 고장회복 기법을 적용했을 때와 비교하여 매우 효과적임을 보이고, 다양한 환경하에서 여러가지 최적의 checkpoint 간격벡터를 유도하였으며, 시뮬레이션을 통한 정량적 결과로써 그 우수성을 비교하고 검증하였다.

A Time-Redundant Recovery Policy of TMR Failures Using Rollback and Roll-forward

Jae-Young Yoon† · Hag-Bae Kim††

ABSTRACT

In the paper we propose two recovery methods by adopting a rollback and/or roll-forward technique(s) to recover TMR failures in a TMR (structured) system that is the simplest spatial redundancy. This technique is apparently effective to recovering TMR failures primarily caused by transient faults. The proposed policies carry out few reconfigurations at the cost of (minimal) time-overhead needed for those time-redundant schemes. The optimal checkpoint-interval vectors are derived for both methods through the likelihoods of all (possible) states of the system as well as the total execution-time. Consequently, the effectiveness of our proposed policies is validated through certain numerical examples and simulations.

1. 서 론

TMR(Triple Modular Redundancy)은 공간여분을 사용한 가장 간단한 구조를 지니면서도 대표적인 고장 포용 기법중의 하나로 널리 활용되고 있다[1]. TMR 시스템은 세개의 동일한 모듈에서 실행결과를 받아 이들의 값을 서로 비교하여 세개중에 두개 이상의 결과가 같으면 이를 출력으로 발생시키는 보팅(voting) 작동에 기초한다. TMR의 고장난 모듈은 자체 고장이 거의 없는 단순한 조합논리회로를 사용하여 구성된 보터(voter)의 작동에 의해 용이하게 검출될 수 있다. 이러한 보터는 간단한 비교검출기를 사용한 형태에서 자체

※ 본 논문은 정보통신연구관리단의 1997~2000년 대학기초연구지원사업에 의해 지원되었음.
† 준 회 원 : 연세대학교 대학원 기계전자공학부
†† 정 회 원 : 연세대학교 기계전자공학부 전기전공 교수
논문접수 : 1998년 6월 25일, 심사완료 : 1998년 10월 14일

검진과 고장에 안전한 TSCC(Totally Self Checking Circuit)[2]까지 확장될 수 있다. 만약 TMR 시스템의 두개이상의 고장난 모듈에서 동일한 값의 잘못된 결과를 발생시킬 경우에 TMR 고장은 보팅 작동에 의해 검출될 수는 없다. 그러나, 게이트레벨이 아닌 함수레벨에서 두 모듈이상의 잘못된 결과가 동일한 값을 발생할 확률은 극히 미세하므로 이러한 경우는 무시할 수 있다. 이러한 보팅 기법을 사용할 때 임의의 한 모듈의 결과가 잘못된 값일지라도 나머지 두개의 모듈의 값이 옳은 값이라면 잘못된 결과를 가지는 모듈의 영향은 무시될 수 있다. 그러므로 하나의 에러는 보팅 작동에 의해 차폐될 수 있으며, 이러한 에러를 차폐오류(masked error)라 한다. 일반적인 시스템은 여분의 개수가 제한되어 있으므로 고장 모듈이 발생했을 때 재구성 기법을 사용하여 복구하는 횟수는 제한된다. 공간여분에 잘못된 결과를 가지고 있는 프로그램의 일부분을 재실행 또는 프로그램 전체를 재시작하는 시간여분을 추가적으로 적용하면 일시적인 결함으로 인한 고장을 효과적으로 복구할 수 있다. 실제로 컴퓨터 시스템에서 결함의 90%이상이 일시적인 것으로 관찰되는 것을 볼 때[3] TMR 고장회복에 시간여분의 적용 효율성은 상당히 합리성을 지닌다고 볼 수 있다.

시간여분과 공간여분을 혼용한 과거의 연구사례로는 모듈화된 TMR 다중프로세서에 재실행(retry) 기법을 적용한 연구[4], TMR 고장으로 인한 동적 고장의 확률이 최소가 되도록 최적의 재실행 주기를 유도한 연구[5], 그리고 TMR 고장이 검출되었을 때 재시작(restart) 기법을 적용한 연구[6] 등이 있다. 재실행 기법은 빈번한 보팅을 요구하여 보터의 고장확률을 증가시키며, 재시작 기법은 고장난 모듈의 분리, 교체, 그리고 프로그램의 리로딩 및 초기화 등이 요구되기 때문에 상당한 시간을 필요로 한다. 이러한 단점을 극복하기 위해 본 논문에서는 영구적인 결함과 일시적인 결함의 두가지 유형의 결함이 발생할 수 있는 환경하에 있는 TMR 시스템에서 TMR 고장을 효과적으로 복구하기 위해 또다른 형태의 시간여분 기법인 rollback과 roll-forward 기법을 적절히 혼용하는 방법을 제안한다. Rollback 기법은 프로그램의 중간에 checkpoint들을 두어 주기적으로 충분한 양의 정보를 저장하여 고장이 검출되었을 때 프로그램을 처음부터 다시 실행하지 않고 검출시점에서 가장 최근의 checkpoint에 저장되어 있는 정보를 사용하여 이를 다시 실행함으로써 손상된 프로그램의 해당부분을 회복하는 기법이다. TMR 시스템에의 roll-forward 기법의 적용은 일반적으로 회복기법의 적용대상이 아닌 차폐오류가 검출되었을 때 이를 즉시 복구하지 않고 프로그램을 계속 진행시켜 후 checkpoint 순간에서 고장나지 않은 모듈의 값을 고장난 모듈에 (동기화를 고려하여) 복사하는 기법이다. 지금까지 rollback과 roll-forward 기법에 관련된 연구[7, 8]는 많이 되어 있지만 이들은 대부분 독립형 시스템에 적용된 연구로 TMR과 같은 다중 모듈을 갖는 시스템에 직접적으로 적용하는데는 문제가 있다. 본 논문에서는 TMR 고장이 일어날 확률을 줄이고, TMR 고장이 검출됐을 때 이의 복구를 위해 TMR 시스템에 rollback과 roll-forward 기법을 적용하는 다음과 같은 구체적인 두가지 방법을 제안한다. 그리고 해석적 수식 계산을 통한 정성적 측면과 시뮬레이션을 통한 정량적 측면의 성능 비교를 수행한다.

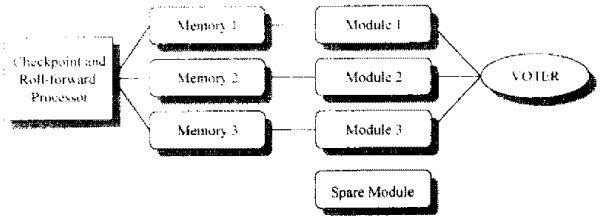
- **방법 1** : TMR 고장이 검출되었을 때에만 rollback 기법을 적용한다.
- **방법 2** : 방법 1에 차폐오류 복구를 위해 roll-forward 기법의 적용을 추가한다.

본 논문에서는 TMR 고장의 회복을 위해 먼저 가능한 모든 시스템 상태확률들을 계산하고 방법 1 또는 2를 적용하여 작업을 실행하는 동안 재구성될 확률을 최소화하면서 작업평균실행시간이 최소화시키는 checkpoint 간격벡터를 유도한다. 이러한 연구결과는 다양한 환경하에서 해석적으로 계산된 각종 수치예제 및 시뮬레이션 결과들을 통해 기존 방법에 대한 상대적인 우수성을 검증한다.

2. 기본가정 및 시스템 상태확률 유도

앞으로 제안될 문제해결을 위해 다음과 같은 합리적인 가정을 세운다.

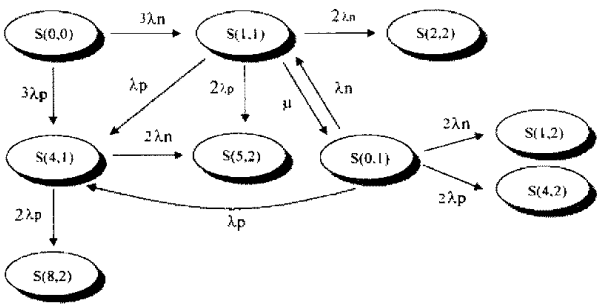
- 개별 모듈의 고장 발생은 독립적이다.
- 다양한 TMR 구조중 그림 1과 같은 혼합형 TMR 시스템을 기본모델로 한다.
- 어떤 시간에 프로그램의 정보들을 저장하기 위해 충분한 용량을 가지고 있는 캐쉬와 같은 추가적인 저장기구를 가정한다.



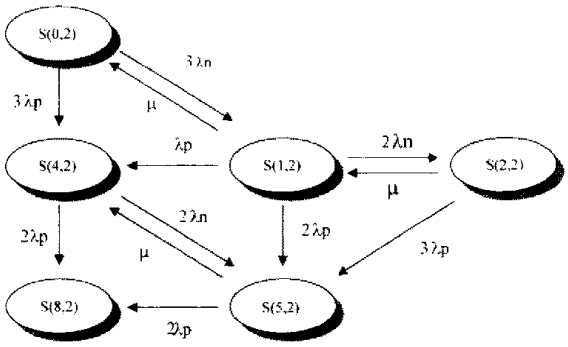
(그림 1) 혼합형 TMR 시스템
(Fig. 1) A hybrid TMR System

또한 영구 및 일시적인 결함은 λ_p(λ_n)의 비율을 가진 시불변 Poisson 프로세스에 의해 발생하고, 일시적인 고장의 활동기간은 평균 $1/\mu$을 가지고 지수함수적으로 분포되어있다고 가정한다. 모든 시스템 상태들의 확률을 계산하지 않고 단지 TMR 고장이 검출되는 시간만을 가지고 시스템의 정확한 상태를 추정하기는 불가능하므로 초기상태로부터 보팅 주기(X_f)사이에서 천이되는 가능한 시스템의 모든 상태확률을 유도한다. 그림 2와 3은 Markov chain을 이용하여 시스템의 가능한 모든 상태를 나타낸 것으로 그림 2는 어떤 초기상태에서 TMR 고장이 일어나기까지의 상태 천이를, 그림 3은 TMR 고장이 일어나고 이를 검출할 때까지의 상태 천이를 나타낸 것이다. 시스템의 상태를 표시하는 $S(x, y)$는 $x=4a+b$ (a는 영구적 결함이 일어난 모듈의 개수, b는 일시적인 결함이 일어난 모듈의 개수)와 이러한 결함에 의해 야기된 고장을 가지고 있는 모듈의 개수 y로 나타낸다.

본 논문에서는 고장난 모듈의 수와 현재 활동중인 결함의 성질에 따라 10개 상태로 모델을 구성하였으며, 2개 이상의 모듈에 결함이 일어난 다양한 상태들을 묶어 시스템에 동일한 영향을 끼치는 상태끼리 하나의 상태로 병합하였다.



(그림 2) TMR 고장 발생까지의 상태 천이도
(Fig. 2) CTMC till the occurrence of a TMR failure



(그림 3) TMR 발생후 검출까지의 상태천이도
(Fig. 3) CTMC till detection of a TMR failure

고장을 야기시킨 결함이 작업의 중간 및 최종 결과를 발생하기 전에 사라질 수도 있지만, 일시적인 결함으로 인해 발생한 고장도 작업 실행 동안에 잘못된 결과를 지속적으로 유지시킬 수 있으므로 영구적인 고장과 같은 영향을 작업의 실행 결과에 미치게 된다.

그림 2와 3에 대한 Markov-chain 미분 방정식의 계수행렬 $\mathbf{M}$을 이용하여 초기조건 $\mathbf{P}(0)$ 를 가지고 선형미분 방정식 $\dot{\mathbf{P}}(t)=\mathbf{M}\cdot\mathbf{P}$에 대한 해를 구하면 식 (1)과 같다. 행렬 $\mathbf{T}\equiv e^{\mathbf{M}t}$ 는 상태 천이행렬로 식 (2)를 통해 계산될 수 있으며, 또한 모든 시스템 상태의 확률들은 식 (3)를 통해 계산할 수 있다.

$$\mathbf{M}=\begin{array}{c}\\ S(0,0)\\ S(0,1)\\ S(1,1)\\ S(4,1)\\ S(0,2)\\ S(1,2)\\ S(2,2)\\ S(4,2)\\ S(5,2)\\ S(8,2)\end{array}\begin{array}{c}\begin{array}{cccccccccc}S(0,0) & S(0,1) & S(1,1) & S(4,1) & S(0,2) & S(1,2) & S(2,2) & S(4,2) & S(5,2) & S(8,2)\end{array}\\ \begin{bmatrix} -3\lambda_n-3\lambda_p & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & -3\lambda_n-3\lambda_p & \mu & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 3\lambda_n & \lambda_n & -3\lambda_n-3\lambda_p-\mu & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 3\lambda_p & \lambda_p & \lambda_p & -2\lambda_n-2\lambda_p & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & -3\lambda_n-3\lambda_p & \mu & 0 & 0 & 0 & 0\\ 0 & 2\lambda_n & 0 & 0 & 3\lambda_n & -2\lambda_n-3\lambda_p-\mu & 2\mu & 0 & 0 & 0\\ 0 & 0 & 2\lambda_n & 0 & 0 & 2\lambda_n & -3\lambda_p-2\mu & 0 & 0 & 0\\ 0 & 2\lambda_p & 0 & 0 & 3\lambda_p & \lambda_p & 0 & -2\lambda_n-2\lambda_p & \mu & 0\\ 0 & 0 & 2\lambda_p & 2\lambda_n & 0 & 2\lambda_p & 3\lambda_p & 2\lambda_n & -\mu-2\lambda_p & 0\\ 0 & 0 & 0 & 2\lambda_p & 0 & 0 & 0 & 2\lambda_p & 2\lambda_p & 0\end{bmatrix}\end{array}$$

$$\mathbf{P}(t)=e^{\mathbf{M}t}\cdot\mathbf{P}(0) \qquad (1)$$

$$\mathbf{T}=e^{\mathbf{M}t}=\mathcal{L}^{-1}[\,\mathbf{R}(s)],\ (\mathbf{R}(s)=[s\,\mathbf{I}-\mathbf{M}]^{-1}) \qquad (2)$$

$$\mathbf{P}(0)=\left[\pi^{(0)}_{S(0,0)}\ \pi^{(0)}_{S(0,1)}\ \pi^{(0)}_{S(1,1)}\ \pi^{(0)}_{S(4,1)}\ \pi^{(0)}_{S(0,2)}\ \pi^{(0)}_{S(1,2)}\ \pi^{(0)}_{S(2,2)}\ \pi^{(0)}_{S(4,2)}\ \pi^{(0)}_{S(5,2)}\ \pi^{(0)}_{S(8,2)}\right]^{T}$$

$$\mathbf{P}(X_f)=\mathbf{T}(X_f)\cdot\mathbf{P}(0)=\left[\frac{\mathbf{A}}{\mathbf{B}}\right] \qquad (3)$$

$$\mathbf{A}=\left[\pi^{X_f}_{S(0,0)},\pi^{X_f}_{S(0,1)},\pi^{X_f}_{S(1,1)},\pi^{X_f}_{S(4,1)}\right]^{T}$$

$$\mathbf{B}=\left[\pi^{X_f}_{S(0,2)},\pi^{X_f}_{S(1,2)},\pi^{X_f}_{S(2,2)},\pi^{X_f}_{S(4,2)},\pi^{X_f}_{S(5,2)},\pi^{X_f}_{S(8,2)}\right]^{T}$$

TMR 고장이 보팅 순간에 검출시 시스템은 **B**행렬 내의 상태에 있다고 추정되며, 만약 시스템이 어떠한 결함이나 이로 인한 고장을 내재하고 있지 않다면 **A** 행렬 중 $S(0,0)$ 의 상태에 머무르고 있고, 시스템이 보팅 순간에 차폐오류를 가지고 있다면 **A**행렬 중 $S(0,1)$, $S(1,1)$, $S(4,1)$ 의 상태에 위치함을 의미한다.

3. 최적의 checkpoint 간격벡터 유도

본 절에서는 각각의 checkpoint 사이의 간격들로 구성되어 있는 간격벡터를 유도한다. 최적의 checkpoint 간격벡터는 작업의 평균실행시간이 최소일 때의 간격벡터로 규정하며, 이의 최적값과 재구성 없이 작업을 종료할 확률을 유도하는데 필요한 변수들은 다음과 같이 정의된다.

- X_f : 보팅사이의 간격으로 일정한 값 유지.
- X_j^i : $i-1$번째와 i번째 checkpoint 사이에서 $i-1$번째 checkpoint으로부터 j번째 보팅 순간 $(1 \le j \le t_i)$을 나타냄.
- t_i : $i-1$ 번째와 i번째 checkpoint 사이의 보팅횟수. m개의 checkpoint에 대한 전체 보팅 횟수 n은 $n = \sum_{i=1}^{m} t_i$

TMR 고장 검출시 시스템은 **B**행렬중 하나의 상태에 있으며, rollback 성공후에 상태확률은 이를 바탕으로 Bayesian 정리에 의해 갱신된다. 성공적인 rollback은 **A**행렬의 상태확률값들을 증가시키며, **B**행렬 상태확률값들을 감소시킨다. TMR 시스템에 방법 1과 2를 각각 적용하면 S_B상태중 $S(0,2)$, $S(1,2)$, $S(4,2)$ 는 rollback을 통해 S_A상태로 천이될 수 있으며, S_A 상태중 $S(0,1)$ 은 roll-forward 기법을 통해 $S(0,0)$ 상태로 천이될 수 있다. 이러한 사실을 바탕으로 제안된 모델에서 $i-1$ 번째와 i번째 checkpoint 사이의 j번째 보팅 순간에서 가능한 시스템 상태 확률행렬 $\mathbf{P}(X_j^i)$은 식 (4)와 같이 유도된다. 식(4)에서 $\mathbf{P}_j^i(0)$은 $i-1$ 번째와 i번째 checkpoint 사이의 j번째 보팅 순간에서의 상태 확률행렬을 유도하기 위한 초기상태 확률행렬을 나타내며, 작업을 시작할 때는 어떤 고장이나 결함도 가지고 있지 않다고 가정한다.

$$\mathbf{P}(X_j^i) = \mathbf{T}(X_f) \cdot \mathbf{P}_j^i(0) = \begin{bmatrix} \mathbf{A}(X_j^i) \\ \mathbf{B}(X_j^i) \end{bmatrix} \quad (4)$$

$$\mathbf{A}(X_j^i) = \begin{bmatrix} \pi^i_{j\,S(0,0)} & \pi^i_{j\,S(0,1)} & \pi^i_{j\,S(1,1)} & \pi^i_{j\,S(4,1)} \end{bmatrix}^T$$

$$\mathbf{B}(X_j^i) = \begin{bmatrix} \pi^i_{j\,S(0,2)} & \pi^i_{j\,S(1,2)} & \pi^i_{j\,S(2,2)} & \pi^i_{j\,S(4,2)} \\ \pi^i_{j\,S(5,2)} & \pi^i_{j\,S(8,2)} \end{bmatrix}^T$$

$$\mathbf{P}_j^i(0) = \begin{bmatrix} \mathbf{A}(X_{j-1}^i) & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

for $\quad 2 \le j \le t_i$

$$\mathbf{P}_1^i(0) = [\,1 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0\,]^T$$

$i-1$ 번째와 i번째 checkpoint 사이에서 TMR 고장이 일어났을 때 검출될 평균거리 $R(i)$ 는 식 (6)와 같이 계산된다.

$$F^i(j) = \sum \mathbf{B}(X_j^i), \quad 1 \le j \le t_i, \quad F^i_{total} = \sum_{j=1}^{t_i} F^i(j) \quad (5)$$

$$R(i) = \frac{1}{F^i_{total}} \left(\sum_{j=1}^{t_i} j \cdot (X_f + T_v) \cdot F^i(j) \right) \quad (6)$$

식 (5)에서 $\sum \mathbf{B}(X_j^i)$는 $\sum_{\pi^i_n \in \mathbf{B}(X_j^i)} \pi^i_n$ 을 나타내며, 본 논문에서는 편의상 행렬의 모든 원소들의 합을 간단히 $\sum \mathbf{B}(X_j^i)$ 식으로 표시하기로 한다. 또한, $i-1$번째와 i번째 checkpoint 사이에서 TMR 고장이 일어날 상태 확률행렬 $\mathbf{B}^{f0(i)}_{Total}$은 $\sum_{j=1}^{t_i} \mathbf{B}(X_j^i)$ 같다. $i-1$ 번째와 i번째 checkpoint 사이에서 TMR 고장이 일어났을 때 k번째 rollback 후의 상태확률을 계산하기 위한 초기상태 확률행렬 $\mathbf{P}_1^{rk(i)}(0)$ 과 k번째 rollback 실시 후 $i-1$ 번째와 i번째 checkpoint 사이의 j번째 보팅 순간에서의 상태 확률행렬 $\mathbf{P}^{rk(i)}(X_j^i)$은 식 (7)과 같이 유도할 수 있다.

$$\mathbf{P}_1^{rk(i)}(0) = [\,\pi^{rk(i)}_{S(0,2)}(0) \;\; 0 \;\; \pi^{rk(i)}_{S(1,2)}(0) \;\; \pi^{rk(i)}_{S(4,2)}(0) \;\; 0$$

$$0 \;\; \pi^{rk(i)}_{S(2,2)}(0) \;\; 0 \;\; \pi^{rk(i)}_{S(5,2)}(0) \;\; \pi^{rk(i)}_{S(8,2)}(0)\,]^T, \; \pi^{rk(i)}_n(0) \in \mathbf{B}^{rk_{-1}f0(i)}_{Total}$$

$$\mathbf{P}^{rk(i)}(X_j^i) = \mathbf{T}(X_f) \cdot \mathbf{P}_j^{rk(i)}(0) = \begin{bmatrix} \mathbf{A}^{rk}(X_j^i) \\ \mathbf{B}^{rk}(X_j^i) \end{bmatrix} \quad (7)$$

$$\mathbf{P}_j^{rk(i)}(0) = \begin{bmatrix} \mathbf{A}^{rk}(X_{j-1}^i) & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

for $\quad 2 \le j \le t_i$

k번째 rollback 실시 후 $i-1$ 번째와 i번째 checkpoint 사이에서 TMR 고장이 다시 검출될 확률행렬 $\mathbf{B}_{\text{Total}}^{\text{rk(i)}}$은 $\sum_{j=1}^{t}\mathbf{B}^{\text{rk}}(X_j^i)$와 같으며, k번의 rollback 후에 $i-1$ 번째와 i번째 checkpoint 사이에서 TMR 고장이 다시 발생했을 경우에, 검출될 평균기대 $R^{\text{rk}}(i)$는 식 (8)과 같이 유도된다.

$$F^{\text{rk(i)}}(j)=\sum \mathbf{B}^{\text{rk}}(X_j^i)\ ,\ 1\le i\le t_i$$

$$F_{\text{totoal}}^{\text{rk(i)}}=\sum_{j=1}^{t_i}F^{\text{rk(i)}}(j)$$

$$R^{\text{rk}}(i)=\frac{1}{F_{\text{totoal}}^{\text{rk(i)}}}\left(\sum_{j=1}^{t_i}j\cdot(X_f+T_v)\cdot F^{\text{rk(i)}}(j)\right) \quad (8)$$

만약 TMR 고장이 두번의 rollback 후에도 다시 검출되었다면 TMR 고장의 원인이 된 결함의 성질은 쉽게 파악될 수 없다. 실제로 이러한 TMR 고장은 영구적인 결함에 의해 일어날 수도 있고, 상대적으로 긴 지속시간을 가진 일시적인 결함이나 다른 고장의 원인에 의해 야기되어 발생할 수도 있기 때문이다. 그러나 일시적인 결함에 의해 유도된 TMR 고장이 두번의 rollback 후에도 회복되지 않을 확률과 rollback을 실시하는 동안에 다시 TMR 고장이 두번 연속 일어날 확률은 매우 작기 때문에 이러한 고장들은 영구적인 결함에 의해 일어났다고 가정하며, 두번의 rollback 후의 첫번째 보팅 순간에서 검출된다고 가정한다. 실제적으로 재구성을 실시하기 전의 rollback의 횟수를 결정하는 문제도 중요한 문제가 될 수 있다. i번째 checkpoint 순간에서 두번의 rollback 후에 갱신된 상태확률 $\pi_n^{u(i)}$은 방법 1과 방법 2에 대해 각각 식 (9), (10)과 같다.

(i) 방법 1

$$\pi_n^{u(i)}=\begin{cases}\pi_n^i+\sum_{k=1}^{2}\pi_n^{rk(i)} & \text{for } n\in[S(0,0),S(0,1),S(1,1),S(4,1)]\\ 0 & \text{otherwise}\end{cases} \quad (9)$$

$$\mathbf{P}_1^{i+1}(0)=[\ \pi_{S(0,0)}^{u(i)}\ \ \pi_{S(0,1)}^{u(i)}\ \ \pi_{S(1,1)}^{u(i)}\ \ \pi_{S(4,1)}^{u(i)}\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0]^T$$

(ii) 방법 2

$$\pi_n^{u(i)}=\begin{cases}\pi_n^i+\pi_{S(0,1)}^i+\sum_{k=1}^{2}\pi_n^{rk(i)}+\sum_{k=1}^{2}\pi_{S(0,1)}^{rk(i)} & \text{for } n\in[S(0,0)]\\ \pi_n^i+\sum_{k=1}^{2}\pi_n^{rk(i)} & \text{for } n\in[S(1,1),S(4,1)]\\ 0 & \text{otherwise}\end{cases} \quad (10)$$

$$\mathbf{P}_1^{i+1}(0)=[\ \pi_{S(0,0)}^{u(i)}\ \ 0\ \ \pi_{S(1,1)}^{u(i)}\ \ \pi_{S(4,1)}^{u(i)}\ \ 0\ 0\ 0\ 0\ 0\ 0]^T$$

$$\pi_n^i\in\mathbf{A}_{\text{Total}}^{(i)},\ \ \pi_n^{r1(i)}\in\mathbf{A}_{\text{Total}}^{r1(i)},\ \ \pi_n^{r2(i)}\in\mathbf{A}_{\text{Total}}^{r2(i)}$$

$$\mathbf{A}_{\text{Total}}^{(i)}=\mathbf{A}(X_{t_i}^i),\ \ \mathbf{A}_{\text{Total}}^{rk(i)}=\mathbf{A}^{rk(i)}(X_{t_i}^i)$$

식 (9),(10)에서 $\pi_n^{rk(i)}$은 i번째 checkpoint 순간에서 k번째 rollback 후에 n상태에 존재할 확률을 나타내며, i번째 checkpoint 순간에서의 갱신된 확률 $\pi_n^{u(i)}$은 i번째와 $i+1$ 번째 checkpoint 사이에서의 상태확률을 유도하기 위한 초기상태 확률값들이 된다. TMR 고장이 두번의 rollback을 통해 성공적으로 회복될 확률행렬은 $\sum\mathbf{A}_{\text{Total}}^{r1(i)}+\sum\mathbf{A}_{\text{Total}}^{r2(i)}$이며, 재구성은 두번의 rollback이 모두 실패하였을 때 연이어 시작하게 되므로 rollback을 적용한 후에 재구성될 확률값은 감소하게 된다. 그리고, Roll-forward 기법이 성공하여, 증가된 $S(0,0)$상태 확률값은 $\pi_{S(0,1)}^i+\sum_{k=1}^{2}\pi_{S(0,0)}^{rk(i)}+\sum_{k=1}^{2}\pi_{S(0,1)}^{rk(i)}$와 같게 된다. 작업이 m번의 checkpoint와 함께 수행된다면, 재구성없이 작업의 실행이 끝났을 때의 시스템 상태확률행렬과 전체 확률값은 식 (11)와 같으며, 재구성될 전체 확률값은 $\sum_{i=1}^{m}\sum\mathbf{B}_{\text{Total}}^{r2(i)}=1-P_{\text{end}}$이다.

$$\mathbf{P}(X_{\text{end}})=\mathbf{P}^{m+1}(0)\ ,\ P_{\text{end}}=\sum\mathbf{P}(X_{\text{end}}) \quad (11)$$

3.1 전체 작업의 평균실행시간 유도

작업이 m개의 checkpoint들과 n개의 보팅들과 함께 수행되고, $w_i\ (1\le i\le m)$을 작업의 처음부터 i번째 checkpoint가 끝났을 때까지의 시간, 그리고 W_i를 $W_i=E(w_i)$, 즉 w_i의 평균값이라고 정의하면, $w=w_m$와 $W=W_m$는 각각 전체 작업의 실행시간과 이의 평균값이 된다. v_i를 $i-1$ 번째와 i번째 checkpoint 사이의 작업 실행시간을 나타낸다고 정의하면 rollback, roll-forward와 재구성 후의 $w_i\ (2\le i\le m)$는 식(12)과 같이 회귀적으로 나타낼 수 있다.

$$w_i=w_{i-1}+v_i \quad (12)$$

v_0 기간동안에 TMR 고장이 일어났을 때는 roll-

back 대신에 재시작 기법이 적용된다. 두번의 재시작 후에도 TMR 고장이 다시 검출된다면 재구성이 적용될 것이다. 작업이 재구성된 후에는 변수 w_1의 확률값은 갱신된다. 평균실행시간을 유도하기 위한 시간 비용들은 다음과 같다.

- T_c : checkpointing시에 요구되는 시간비용
- t_{rec} : 시스템 재구성을 위해 요구되는 시간
- t_{roll} : rollback 실시를 위해 요구되는 시간
- t_{rest} : 재시작 실시를 위해 요구되는 시간
- T_v : 보팅의 시간비용
- T_{copy} : 정상적인 모듈의 값을 고장난 모듈에 복사하는데 요구되는 시간

방법 1 : 방법 1에서 평균실행시간을 계산하기 위한 필수 파라미터들은 아래와 같다.

- p(i) : $i-1$ 번째와 i번째 checkpoint 사이를 rollback이나 재구성없이 실행을 마칠 확률
- r1(i), r2(i) : 각각 첫 번째 rollback, 두번째 rollback을 통해 회복될 확률
- c(i) : 재구성을 통해 회복될 확률

$$T(i)=\sum P_t^i(0),\quad p(i)=\frac{\sum A(X_{t_i}^i)}{T(i)}$$

$$r1(i)=\frac{\sum A^{r1}(X_{t_i}^i)}{T(i)},\quad r2(i)=\frac{\sum A^{r2}(X_{t_i}^i)}{T(i)}$$

$$c(i)=\frac{\sum B_{Total}^{r2(i)}}{T(i)}$$

$$p(i)+r1(i)+r2(i)+c(i)=1$$

와 같이 정의되고,

$$w_i=w_{i-1}+v_i,\quad w_i\quad 2\le i\le m$$

$$v_i=\begin{cases} t_i(X_f+T_v)+T_c & \text{with } p(i)\\ R(i)+t_i(X_f+T_v)+t_{roll}+T_c & \text{with } r1(i)\\ R(i)+R^{r1}(i)+t_i(X_f+T_v)+2t_{roll}+T_c & \text{with } r2(i)\\ R(i)+R^{r1}(i)+X_f+T_v+2t_{roll}+t_{rec}+w_i & \text{with } c(i)\end{cases}$$

$w_1=v_1$이며, t_{roll}대신에 t_{rest}을 대입한다. 결과적으로 위의 수식들을 사용하여 전체 작업의 평균 실행시간 $W=W_m$은 식 (13)과 같이 유도된다.

$$A(i)=t_i(X_f+T_v)+T_c$$

$$W_1=\frac{1}{1-c(1)}[A(1)(1-c(1))+(R(1)+T_{rest})(1-p(1))+(R^{r1}(1)+t_{rest})(1-p(1)-r1(1))+(X_f+T_v+t_{rec})c(1)]$$

$$W_m=\prod_{k=2}^{m}\left(\frac{1}{1-c(k)}\right)W_1+\sum_{i=2}^{m}\left[\left(\prod_{k=i}^{m}\frac{1}{1-c(k)}\right)(A(j)(1-c(j))+(R(j)+t_{roll})(1-p(j))+(R^{bl}(j)+t_{roll})(1-p(j)-r1(j))+(X_f+T_v+t_{rec})c(j))\right] \quad (13)$$

방법 2 : 방법 2에서 평균실행시간을 계산하기 위한 필수 파라미터들은 아래와 같다.

- p(i) : $i-1$ 번째와 i번째 checkpoint 사이를 roll-back, roll-forward 또는 재구성없이 실행을 마칠 확률
- q(i), q1(i), q2(i) : 각각 rollback이 0번, 1번, 2번 실행된 후에 차폐오류가 검출될 확률
- r1(i), r2(i), c(i) : 방법 1과 같이 정의된다.

$$T(i)=\sum P(X_{t_i}^i),\quad p(i)=\frac{\pi_{S(0,0)}^i}{T(i)}$$

$$q(i)=\frac{\sum A(X_{t_i}^i)-\pi_{S(0,0)}^i}{T(i)},\quad r1(i)=\frac{\pi_{S(0,0)}^{r1(i)}}{T(i)}$$

$$q1(i)=\frac{\sum A^{r1}(X_{t_i}^i)-\pi_{S(0,0)}^{r1(i)}}{T(i)},\quad r2(i)=\frac{\pi_{S(0,0)}^{r2(i)}}{T(i)}$$

$$q2(i)=\frac{\sum A^{r2}(X_{t_i}^i)-\pi_{S(0,0)}^{r2(i)}}{T(i)}$$

$$c(i)=\frac{\sum B_{Total}^{r2(i)}}{T(i)}$$

$$p(i)+q(i)+q1(i)+q2(i)+r1(i)+r2(i)+c(i)=1$$

와 같이 정의되고,

$$w_i=w_{i-1}+v_i,\quad w_i\quad 2\le i\le m$$

$$v_i=\begin{cases} t_i(X_f+T_v)+T_c & \text{with } p(i)\\ t_i(X_f+T_v)+T_c+T_{copy} & \text{with } q(i)\\ R(i)+t_i(X_f+T_v)+t_{roll}+T_c+T_{copy} & \text{with } q1(i)\\ R(i)+R^{bl}(i)+t_i(X_f+T_v)+2t_{roll}+T_c+T_{copy} & \text{with } q2(i)\\ R(i)+t_i(X_f+T_v)+t_{roll}+T_c & \text{with } r1(i)\\ R(i)+R^{bl}(i)+t_i(X_f+T_v)+2t_{roll}+T_c & \text{with } r2(i)\\ R(i)+R^{bl}(i)+X_f+T_v+2t_{roll}+t_{rec}+w_i & \text{with } c(i)\end{cases}$$

방법 1과 마찬가지로 $w_1 = v_1$이며, t_{roll}대신에 t_{rest}을 대입하고, 전체 작업의 평균실행시간 $W = W_m$은 식 (14)와 같이 유도된다.

$$A(i) = t_i \cdot (X_f + T_v) + T_c$$

$$W_1 = \frac{1}{1-c(1)}[A(1)(1-c(1)) + (R^{bl}(1)+t_{rest})(q2(1)+r2(1)+c(1)) + T_{copy}(q(1)+q1(1)+q2(1)) + (X_f+T_v+t_{rec})c(1) + (R(1)+t_{rest})(1-p(1)-q(1))]$$

$$W_m = \prod_{k=2}^{m}\left(\frac{1}{1-c(k)}\right)W_1 + \sum_{j=2}^{m}\left[\left(\prod_{k=j}^{m}\frac{1}{1-c(k)}\right)(A(j)(1-c(j)) + T_{copy}(q(j)+q1(j)+q2(j)) + (X_f+T_v+t_{rec})c(j) + (R^{bl}(j)+t_{roll})(q2(j)+r2(j)+c(j)) + (R(j)+t_{roll})(1-p(j)-q(j)))\right] \quad (14)$$

4. 수치예제

본 절에서는 TMR 시스템에 제안된 두가지 방법을 적용시킨 후 해석적 방법에 의해 계산된 수치값과 난수 발생에 의한 컴퓨터 시뮬레이션 결과치를 비교한다. 그리고 여러가지 시스템 파라미터들을 변화시키면서 최적의 checkpoint 간격벡터를 구하는 예제들을 보여준다. 시뮬레이션에 가정된 기본 시스템 모델은 그림 1과 같으며, 해석적 방법에 의한 수치계산에 사용되는 시스템 파라미터들의 값은 식 (15)와 같다.

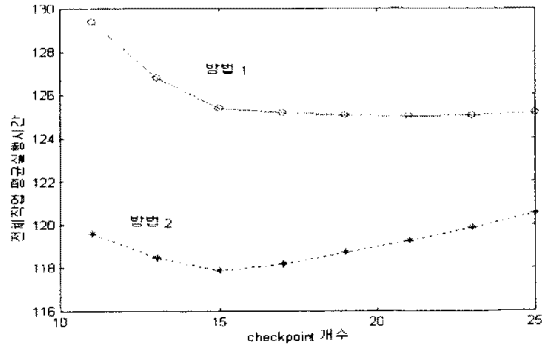
$$T_c = 0.5, \quad t_{rec} = 1, \quad t_{rest} = 0.2, \; t_{roll} = 0.2$$
$$T = 100, \; T_v = 0.005, \quad T_{copy} = 0.5 \quad (15)$$

표 1은 그림 2와 3의 Markov-chain model을 기본으로 구한 MTTF 값과 난수발생에 의한 컴퓨터 시뮬레이션 결과값을 서로 비교하여 보여주고 있다.

〈표 1〉 MTTF의 비교
〈Table 1〉 Comparison of MTTF
(λ_n = 0.02, λ_p = 0.0002, μ = 0.5, 시뮬레이션횟수 : 50000)

	해석적 수치결과	시뮬레이션결과	오차
MTTF	41.2541	41.3114	0.1387%

그림 4는 최적의 보팅 횟수(재구성될 확률값이 최소일때의 횟수)를 가질 때 checkpoint 개수에 따른 전체 작업의 평균실행시간을 나타내고 있다. 각 checkpoint



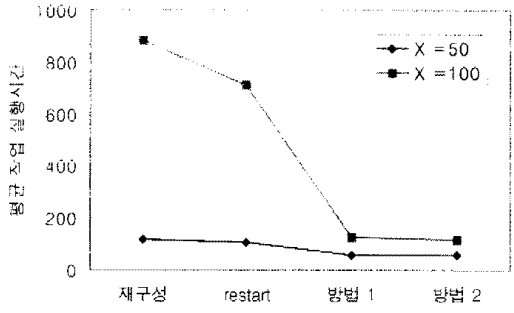
(그림 4) Checkpoint 수에 따른 평균실행시간
(Fig. 4) The mean execution time versus the number of checkpoints
(보팅 횟수 : 31, λ_p = 0.0002, λ_n = 0.02, μ = 1)

개수에서의 평균실행시간은 각각의 checkpoint 개수를 가질 수 있는 여러가지 벡터중에서 평균실행시간이 최소일 때의 값이다. 실제로 전체작업 평균실행시간을 최소화기키는 최적의 checkpoint 간격벡터는 표 2와 같다. 표 2에서 각 숫자는 checkpoint들 사이의 보팅 횟수를 나타낸다.

〈표 2〉 최적의 checkpoint 간격벡터
〈Table 2〉 The optimal checkpoint vector
(λ_n = 0.02, λ_p = 0.0002, 시뮬레이션횟수 : 99000)

방법 1 (μ = 2)			
최적의 checkpoint간격벡터		[4 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2]	
보팅 횟수	checkpoint 개수	49	20
전체작업 평균실행시간	해석적 수치결과	121.7880	오차
	시뮬레이션 결과	122.1220	0.2735%
재구성이 일어나지 않을 확률	해석적 수치결과	0.9737	오차
	시뮬레이션 결과	0.9714	0.2368%

방법 2 (μ = 2)			
최적의 checkpoint간격벡터		[4 4 3 3 3 3 3 3 3 3 3 3 3 3 3 3]	
보팅 횟수	checkpoint 개수	47	15
전체작업 평균실행시간	해석적 수치결과	116.4179	오차
	시뮬레이션 결과	116.2842	0.1150%
재구성이 일어나지 않을 확률	해석적 수치결과	0.9781	오차
	시뮬레이션 결과	0.9758	0.2357%



(그림 5) 평균실행시간 비교
(Fig. 5) Comparison of the mean execution time
(λ_p = 0.0002, λ_n = 0.02, μ = 1)

본 논문에서 제안된 두가지 방법을 최적의 checkpoint 간격벡터를 가지고 TMR 시스템에 적용한 경우와 일반적인 고장회복기법인 재구성 기법을 사용한 경우, 그리고 시간여분의 다른 형태인 재시작 기법을 적용한 경우의 작업시간에 따른 각각의 평균실행시간의 비교는 그림 5에 나타나 있으며, 서로의 성능을 비교하기 위해 표 3에 구체적인 값을 나타내었다. 재구성과 재시작 기법을 적용한 경우의 보팅 횟수는 제안된 두가지 방법의 보팅 횟수와 같다고 가정하였다.

〈표 3〉 재구성, 재시작 기법과 제안된 방법들의 평균실행시간
〈Table 3〉 The mean execution time when reconfiguration, restart, or two proposed methods are applied
(λ_p = 0.0002, λ_n = 0.02, μ = 1)

	전체작업시간(X)	
	T = 50 (보팅 횟수: 16)	T = 100 (보팅 횟수: 31)
재구성	115.6009	884.6890
재시작 기법	106.3790	709.7826
방법 1	60.6315	125.0328
방법 2	58.1443	117.8916

5. 결 론

다양한 조건하의 시뮬레이션 및 해석적 방법에 의한 수치예제를 통해 볼 수 있듯이 TMR 시스템에서 일시적인 결함으로 야기된 TMR 고장은 제안된 두가지 방법에 의해 시스템 구축비용의 효율화 관점에서 매우 효과적으로 복구될 수 있다. 또한 일반적으로 차폐오류가 검출되었을 때 roll-forward 기법을 사용하여 TMR 고장이 일어날 확률을 줄이는 방법 2가 방법 1보다 더욱 효과적임을 알 수 있다. 빈번한 checkpoint는 상당한 시간비용을 요구하는 단점이 있고, 빈도수가 적은 보팅 횟수는 영구적인 결함에 의해 야기될 수 있는 TMR 고장의 확률을 증가시키는 단점이 있으므로, 최적의 checkpoint 간격벡터는 빈번한 보팅과 드문 checkpoint를 사용하는 것이 효과적임을 보였다. 결론적으로 제안된 방법 1과 2는 다른 회복기법보다 여분모듈을 거의 사용하지 않고 TMR 고장을 매우 효과적으로 회복할 수 있으며, 이러한 방법은 적은 수의 여분모듈을 가지고 시스템의 신뢰도를 높일 수 있는 설계 방향을 제시한다.

참 고 문 헌

[1] A. Hopkins Jr., etc., "FTMP-a highly reliable fault-tolerant multiprocessor for aircraft," *Proceedings of the IEEE*, Vol.66, No.10, pp.1221-1239, Oct. 1978

[2] N. Gaitanis, "The design of totally self-checking TMR fault-Tolerant systems," *IEEE Trans. Computers*, Vol.37, No.11, pp.1450-1454, Nov. 1988.

[3] S. McConnel, etc., "The measurement and analysis of transient errors in digital computer systems," *in Digest of Papers, FTCS-9*, pp.67-70, June 1979.

[4] P. Chande, etc., "Modular TMR multiprocessor system," *IEEE Trans. on Industrial Electronics*, Vol.36, No.1, pp.34-41, February 1989.

[5] H. Kim and K. Shin, "Design and analysis of an optimal instruction retry policy for TMR controller computers," *IEEE Trans. on Computers*., Vol.45, No.11, pp.1217-1226, Nov. 1996.

[6] K. Shin and H. Kim, "A time redundancy approach to TMR failures using fault-state likelihoods," *IEEE Trans. on Computers*, Vol.43, No.10, pp.1151-1162, Oct. 1994.

[7] K. Chandy and C. Ramamoorthy, "Rollback and recovery strategies for computer programs," *IEEE Trans. Computers*, Vol.C-21, No.6, pp.5465-556, June. 1972.

[8] D. Pradhan and N. Vaidya, "Roll-Forward and

rollback recovery : performance-reliability trade off," *IEEE Trans. Computers*, Vol.46, No.3, pp. 372-378, Mar. 1997.

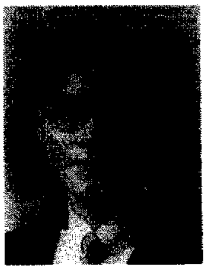
윤 재 영

e-mail : jyyoon@bubble.yonsei.ac.kr

1997년 2월 연세대학교 전기공학과 졸업(학사)

1997년 3월~1998년 현재 연세대학교 대학원 기계전자공학부 전기전공 석사과정

관심분야 : Fault-tolerant 시스템, 실시간 제어시스템



김 학 배

e-mail : hbkim@bubble.yonsei.ac.kr

1988년 서울대학교 전자공학과 졸업(학사)

1988년~1994년 The Univ. of Michigan EECS 공학석사 및 박사

1994년~1996년 NASA Langley 연구센터 NRC 연구원

1996년~1998년 현재 연세대학교 기계전자공학부 전기전공 교수

관심분야 : 실시간 시스템, Fault-tolerant computing, 자동화 및 제어 시스템