재실행과 Rollback 기법을 사용한 TMR 고장의 시간여분 복구 기법

강 명 석[†] · 손 병 희[†] · 김 학 배[††]

요 약

본 논문에서는 복잡해져 가는 제어 컴퓨터의 높은 신뢰성 확보를 위해 시간 여분(time redundancy)의 일종인 재실행과 rollback 기법을 TMR 구조에 적절하게 혼용하는 방법을 제안한다. 재실행과 rollback 기법은 약간의 추가 시간만으로 재구성(reconfiguration) 없이도 일시적인 결함(fault)에 의해 발생한 TMR 고장(failure)의 회복을 위해 상호 보완적으로 사용될 수 있다. 이를 위해 고장 검출시 가능한 모든 시스템의 고장상태 확률을 추정하였으며, 이를 바탕으로 전체 작업의 평균 실행시간이 최소가 되는 최적의 재실행과 rollback 횟수를 유도하였다. 또한 제안된 방법과 다른 고장회복 기법을 적용했을 때의 평균 실행 시간을 정량적으로 비교하여 그 우수성을 검증하였다.

키워드 : TMR 시스템, 시간여분, 재실행, 롤백, 차폐오류

A Time-Redundant Recovery Scheme of TMR failures Using Retry and Rollback Techniques

Myungseok Kang[†] · Byounghee Son[†] · Hagbae Kim[††]

ABSTRACT

This paper proposes an integrated recovery approach applying retry and rollback techniques to recover the TMR failure. Combining the time redundancy techniques with TMR system is apparently effective to recover the TMR failure(or masked error) primarily caused by transient faults. These policies need fewer reconfigurations at the cost of extra time required for the time redundant schemes. The optimal numbers of retry and rollback to minimize the mean execution time of tasks are derived for the proposed method through computing the likelihoods of all possible states of the failed system. The effectiveness of the proposed method is validated through examining certain numerical examples and simulations conducted with a variety of parameters governing environmental characteristics.

Key Words : TMR System, Time-redundancy, Retry, Rollback, Masked Error

1. 서 론

고장 포용(fault-tolerance) 시스템은 일반적으로 추가적인 설비, 즉 여분(redundancy)을 이용하여 고장을 검출하고 고장의 영향을 제거하여 고장의 존재 하에서도 작업이 올바르게 수행되도록 하는 시스템이다. 이러한 시스템 중 TMR (Triple Modular Redundancy)은 공간여분(하드웨어 및 소프트웨어)을 정적으로 활용하는 가장 간단한 구조를 지닌 대표적인 고장 포용 기법 중의 하나로 널리 활용되고 있다 [1-5]. TMR 시스템은 하나의 고장(failure) 난 모듈을 차폐(masking)할 수 있는 최소의 공간 여분을 사용한다. 또한, TMR 시스템은 세 개의 동일한 작업을 수행한 모듈들에서 실행 결과를 받아 이 세 개의 값을 서로 비교하여 이들 중에 두 개 이상의 결과가 같은 값을 가지면 이 결과를 출력으로 발생시키는 다수결 보팅(majority voting) 작동에 기초한다. 이처럼 TMR의 고장 난 모듈은 자체 고장이 거의 발생하지 않는 단순한 조합 논리회로를 사용하여 구성된 보터(voter)의 작동에 의해 검출될 수 있다. 이러한 보터는 간단한 비교 검출[6, 7]을 사용한 형태에서 자체 검진과 고장에 안전한 TSCC(Totally Self-Checking Circuit)[8]까지 확장될 수 있다. 이러한 보팅 기법을 사용할 때 임의의 한 모듈의 결과가 잘못된 값일지라도 나머지 두 개의 모듈의 값이 옳은 값이라면 잘못된 결과를 가지는 모듈의 영향은 무시될 수 있다. 그러므로 하나의 에러는 보팅 작동에 의해 차폐될 수 있으며, 이러한 에러를 차폐 오류(masked error)라 한다.

※ 이 논문은 2006년도 교육인적자원부 BK21 사업의 일환인 연세대학교 전기전자공학부 TMS 사업단의 지원을 받아 연구되었음.
† 준 회 원 : 연세대학교 대학원 전기전자공학과 박사과정
†† 정 회 원 : 연세대학교 전기전자공학과 부교수
논문접수 : 2005년 12월 9일, 심사완료 : 2006년 7월 25일

일반적인 시스템은 여분의 개수가 제한되어 있으므로 고장 모듈이 발생했을 경우 재구성(reconfiguration) 기법을 사용하여 고장 모듈을 복구하는 횟수는 제한된다. 공간 여분에 잘못된 결과를 가지고 있는 프로그램의 일부분을 재실행 또는 프로그램 전체를 재시작하는 시간 여분을 추가적으로 적용하면 일시적인 결함(fault)으로 인한 고장을 효과적으로 복구할 수 있다. 실제로 컴퓨터 시스템에서 발생하는 결함의 90% 이상이 일시적인 것으로 관찰되는 것을 볼 때[9] TMR 고장 회복에 시간 여분의 적용 효율성은 상당한 합리성을 지닌다고 볼 수 있다.

시간 여분에는 instruction 재실행(retry), 프로그램 rollback, 프로그램 재시작(restart) 등의 기법들이 있다. 시간 여분의 가장 간단한 형태인 재실행 기법에 대한 연구[10]가 진행되어 왔으나 대다수 단순 시스템에서의 사용을 목적으로 하고 있다. 좀 더 복잡한 시스템에 재실행 기법을 적용한 예로는 TMR 다중 프로세서에서 신뢰도와 효용성을 높이기 위해 재실행 기법을 사용한 경우[11], 동적 고장(dynamic failure) 확률이 최소가 되도록 재실행 주기를 구한 연구[12] 등이 있다. 또한 재시작 기법을 TMR 시스템에 적용한 연구로는 [13] 등이 있고 TMR 시스템에서 TMR 고장의 복구를 위해 rollback 기법을 혼용한 연구로는 [14] 등이 있다.

재실행 기법은 모듈에서 결함/에러가 발생하는 즉시 검출된다는 가정 하에서 빈번한 보팅이 이루어질 때 TMR 고장 회복에 효과적으로 적용될 수 있으나, 이러한 가정은 에러/고장의 발생과 검출 사이의 시간으로 정의되는 error-latency가 무시할 수 있을 만큼 작을 때에만 그 효력이 있다. 만약 에러가 발생한 시간과 보팅 작동에 의해 검출된 시간 사이의 간격이 재실행 주기보다 크다면 재실행 기법은 실패하고 재실행 주기만큼의 시간만 낭비하게 된다. 일반적으로 TMR 시스템에서 결함/에러는 발생 즉시 검출될 수 없으며, 별도의 자체 검출 장치 없이는 보팅 순간에만 이산적으로 검출될 수 있다. 그러므로 발생 즉시 고장을 검출하기 위한 빈번한 보팅은 시간비용과 보터 고장의 확률이 증가시키는 단점을 가진다. 또한, [13]에서와 같이 TMR 고장 복구를 위해 재시작 기법을 적용하는 것은 고장 난 모듈의 분리, 교체, 그리고 프로그램의 리로딩 및 초기화 등이 요구되기 때문에 상당한 시간을 필요로 한다. 그리고 rollback 기법에서는 검증된 결과에 대한 checkpoint를 두게 되는데 이를 위해서는 모듈 외에 checkpoint에서의 기록을 저장해야 하는 기억 장소를 두어야 한다. rollback 시 이 기억 장소로부터 이전 checkpoint 시의 결과 값들을 불러와 모듈에 다시 셋업 하는 데 그만큼 retry에 비해 더 높은 time overhead를 가지게 된다.

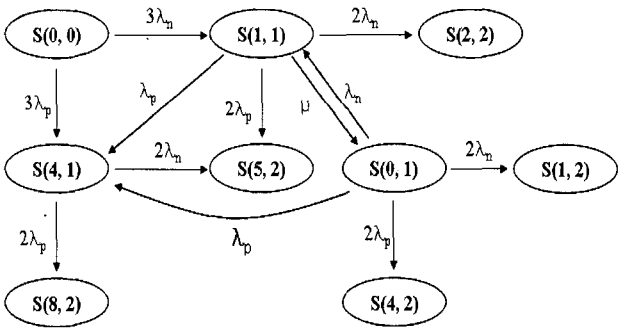
이와 함께, 지금까지 재실행과 rollback 기법을 혼용하는 방법에 대한 연구[15, 16]가 이루어져 왔다. 그러나 이 연구들은 TMR 시스템이 아닌 단순 시스템에서 재실행과 rollback 기법을 적용하였을 때의 fault tolerance latency를 유도하거나[15], 단순 시스템에서 rollback 기법의 적용 전에 재실행 기법을 몇 번 시도하는가에 대한 연구들이었다[16].

본 논문에서는 이러한 세 가지 시간 여분 기법들을 TMR 시스템에 적용하였을 때 나타나는 단점을 극복하기 위해, 영구적인 결함과 일시적인 결함의 두 가지 유형의 결함이 발생할 수 있는 환경 하에 있는 TMR 시스템에서 TMR 고장을 복구하기 위한 방법으로 재실행과 rollback 기법을 혼용하는 방법을 제안한다. 또한 해석적 수식 계산을 통하여 제안한 방법을 분석하고, 시뮬레이션을 통한 정량적 측면의 성능 비교를 수행한다. TMR 시스템에서의 재실행 기법과 rollback 기법을 혼용하는 방법은 rollback 기법만을 적용한 경우와 같이 프로그램 중간에 checkpoint를 두게되며, 작업을 수행하던 도중 TMR 고장의 발생이 검출되었을 경우 우선 재실행 기법을 이용하여 TMR 고장의 회복을 시도한다. 재실행을 통해 고장이 회복되면 시스템은 작업을 계속하여 수행한다. 그러나 재실행을 통해 고장이 회복되지 않았을 경우에는 rollback 기법을 통해 고장 회복을 시도하게 된다. 여기에 차폐 오류에 대하여도 재실행 기법을 적용하여 고장 회복을 이루어 시스템의 효용성 증가를 기대할 수 있다.

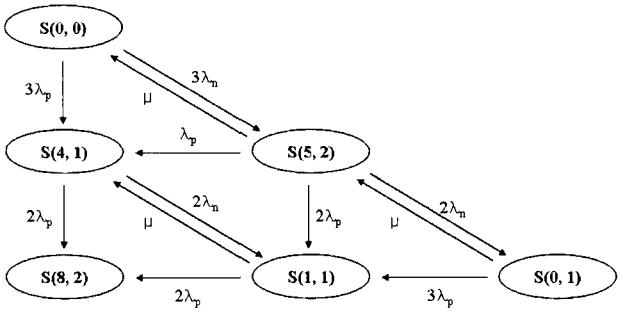
2. 기본가정 및 시스템 상태확률 유도

TMR 시스템에서 각 모듈의 고장 검출과 이에 대한 복구는 시스템의 정확하고 신속한 작업 수행에 필수적인 요소이다. 이때 모든 시스템 상태들의 확률을 계산하지 않고 단지 TMR 고장이 검출되는 시간만을 가지고 시스템의 정확한 상태를 추정(estimation)한다는 것은 불가능하다. 본 장에서는 TMR 고장을 효과적으로 회복하기 위해 시스템의 모든 상태확률들을 유도한다. 시스템의 상태확률을 유도하기 위해 다음과 같은 가정을 세운다.

- 개별 모듈의 고장은 독립적으로 발생한다.
- 여러 TMR 구조 중 Mathur와 Avizienis[17]가 제안한 혼합형 시스템을 기본 모델로 한다.
- 영구 및 일시적인 결함이 $\lambda_p(\lambda_n)$ 의 비율을 가진 시불변 Poisson process에 의해 발생하고, 일시적인 결함의 활동 기간은 평균 $1/\mu$ 을 가지고 지수 함수적으로 분포되어 있다.



(그림 1) TMR 고장 발생까지의 상태 천이도



(그림 2) TMR 고장 발생 후 검출까지의 상태 천이도

시스템의 정확한 상태를 추정하기 위해 초기 상태에서 작업이 시작하여 보팅 주기(X_f) 사이에서 천이되는 가능한 시스템의 모든 상태 확률을 유도한다. (그림 1)과 (그림 2)는 Markov-chain을 이용하여 시스템의 가능한 모든 상태를 나타낸 것이다. (그림 1)은 어떤 초기 상태에서 작업이 시작하여 TMR 고장이 일어나기까지의 상태 천이를, (그림 2)는 TMR 고장이 일어나고 이를 검출할 때까지의 상태 천이를 나타낸다. 시스템의 상태를 표시하는 $S(x, y)$ 는 $x=4a+b$ (a는 영구적 결함이 일어난 모듈 개수, b 는 일시적인 결함이 일어난 모듈 개수)와 이러한 결함에 의해 야기된 고장을 가지고 있는 모듈의 개수 y 로 나타낸다[14]. 본 논문에서는 고장이 발생한 모듈의 수와 활동 중인 결함의 성질에 따라 10개 상태로 모델을 구성하였다. 실제로 모델을 구성하는 데에는 20개로 구별되는 상태들이 필요하지만, 이중 확률적으로 중요하다고 판단되는 10개의 상태만을 사용한다. 이 경우 2개 이상의 모듈에 결함이 발생한 상태들을 묶어 시스템에 동일한 영향을 끼치는 상태끼리 하나의 상태로 병합하였다.

(그림 1)과 (그림 2)에 대한 Markov-chain 미분 방정식의 계수 행렬 $\boldsymbol{M}$ [14]과 초기 상태확률 행렬 $\boldsymbol{P}(0)$ 를 이용하여 시스템 상태확률을 구하기 위한 선형 미분 방정식 $\dot{\boldsymbol{P}}(t)=\boldsymbol{M}\cdot\boldsymbol{P}$ 의 해를 구하면 식 (1)과 같다. 행렬 $\boldsymbol{T}=e^{Mt}$ 는 상태 천이 행렬이며 이는 식 (2)를 통해 계산될 수 있다. 또한, 모든 시스템 상태 확률들은 $\boldsymbol{P}(X_f)=\boldsymbol{T}(X_f)\cdot\boldsymbol{P}(0)$ 를 통해 식 (3)과 같이 계산할 수 있다.

$$\boldsymbol{P}(t)=e^{Mt}\cdot\boldsymbol{P}(0) \tag{1}$$

$$\boldsymbol{T}=e^{Mt}=\mathcal{L}^{-1}[\boldsymbol{R}(s)],\ \boldsymbol{R}(s)=[sI-\boldsymbol{M}]^{-1} \tag{2}$$

$$\boldsymbol{P}(0)=[\pi_{S(0,0)}(0)\ \pi_{S(0,1)}(0)\ \pi_{S(1,1)}(0)\ \pi_{S(4,1)}(0)\ \pi_{S(0,2)}(0)\ \pi_{S(1,2)}(0)\ \pi_{S(2,2)}(0)\ \pi_{S(4,2)}(0)\ \pi_{S(5,2)}(0)\ \pi_{S(8,2)}(0)]^T$$

$$\boldsymbol{P}(X_f)=\begin{bmatrix}\boldsymbol{A}\\---\\\boldsymbol{B}\end{bmatrix} \tag{3}$$

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

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

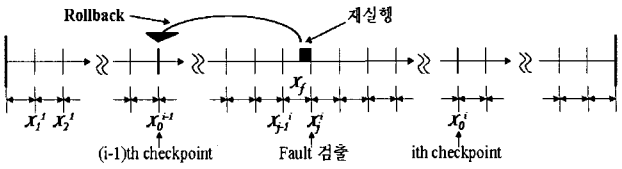
$$\pi_n(X_f)=\sum_i \pi_i(0)\pi^i_n(X_f) \tag{4}$$

식 (4)에서 첨자 i 와 n 은 초기 상태와 보팅 순간 X_f 에서의 상태를 각각 나타낸다. TMR 고장이 보팅 순간에 검출되었을 때 시스템은 $\boldsymbol{B}$ 행렬 내의 상태에 있다고 추정되며, 만약 시스템이 어떠한 결함이나 이로 인한 고장을 내재하고 있지 않다면 $\boldsymbol{A}$ 행렬 중 $S(0, 0)$ 의 상태에 머무르고 있고, 시스템이 보팅 순간에 차폐 오류를 가지고 있다면 $\boldsymbol{A}$ 행렬 중 $S(0, 1)$, $S(1, 1)$, $S(4, 1)$ 의 상태에 머물고 있음을 의미한다.

3. 재실행 및 rollback 기법을 이용한 TMR 고장 회복 상태 확률

본 장에서는 TMR 고장 발생 시 재실행과 rollback 기법을 동시에 적용하여 고장 복구 후의 상태 확률을 유도한다. 재실행과 rollback을 적용하여 고장 발생에 따라 작업 실행을 종료할 확률값을 유도하기 위한 변수들은 다음과 같다.

- T : 고장 없이 전체작업이 완료되었을 때의 시간.
- X_f : 보팅 사이의 간격, 일정한 값 유지.
- X^i_j : $i-1$ 번째와 i 번째 checkpoint 사이에서 $i-1$ 번째 checkpoint로부터 j 번째 보팅 순간($1\le j\le t_i$)을 나타냄.
- t_i : $i-1$ 번째와 i 번째 checkpoint 사이의 보팅 횟수. m 개의 checkpoint에 대한 전체 보팅 횟수 n 은 $n=\sum_{i=1}^{m} t_i$.
- r_t : TMR 고장에 허용된 최대의 재실행 주기.
- K_T : 최대 재실행 횟수.
- K_B : 최대 rollback 횟수.



(그림 3) TMR 고장 회복을 위한 재실행과 rollback 기법 적용

TMR 고장이 검출되었을 때 시스템은 $\boldsymbol{B}$ 행렬 중 하나의 상태에 있으며, 재실행이나 rollback이 성공한 후에 상태 확률들은 이러한 사실을 바탕으로 Bayesian 정리에 의해 갱신될 수 있다. 성공적인 재실행과 rollback은 $\boldsymbol{A}$ 행렬의 상태 확률 값들을 증가시키며 $\boldsymbol{B}$ 행렬의 상태 확률 값들을 감소시킨다. 이와 같은 사실을 바탕으로 제안하고자 하는 모델에서 $i-1$ 번째와 i 번째 checkpoint 사이의 j 번째 보팅 순간에서 가능한 시스템 상태 확률 행렬 $P(X^i_j)$ 는 식 (5)와

같이 유도된다. $P_j^i(0)$ 는 $i-1$ 번째와 i 번째 checkpoint 사이의 j 번째 보팅 순간에서의 상태 확률 행렬을 유도하기 위한 초기 상태 확률 행렬을 나타내며, 작업을 시작할 때는 어떤 고장이나 결함도 가지고 있지 않다고 가정한다.

$$P(X_j^i) = T(X_f) \cdot P_j^i(0) = \begin{bmatrix} A(X_j^i) \\ --- \\ B(X_j^i) \end{bmatrix} \tag{5}$$

보팅 순간에 고장 검출 시 재실행이 성공하기 위해서는 고장이 재실행 주기 내에서 발생하여야 한다. TMR 고장 발생 시 재실행의 성공을 위한 전제 조건으로 식 (5)를 식 (6)으로 변형할 수 있다.

$$T(X_f - r_t) \cdot P_j^i(0) = \begin{bmatrix} A(X_j^i - r_t) \\ --- \\ B(X_j^i - r_t) \end{bmatrix} \tag{6}$$

$$T(r_t) \cdot \left[A(X_j^i - r_t) \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0\right]^T = \begin{bmatrix} A(r_t) \\ --- \\ B(r_t) \end{bmatrix}$$

TMR 고장이 $i-1$ 번째와 i 번째 checkpoint 사이의 j 번째 보팅 순간에서 검출될 확률값 $F^i(j)$ 와 $i-1$ 번째와 i 번째 checkpoint 사이에서 TMR 고장이 검출될 전체 확률값 F_{total}^i 은 식 (7) 과 같이 유도된다. 또한 $i-1$ 번째와 i 번째 checkpoint 사이에서 TMR 고장이 일어났을 때 검출될 평균 거리 $R(i)$ 는 식 (8)과 같이 계산된다.

$$F^i(j) = \sum B(X_j^i),\ 1 \le j \le t_i,\ F_{total}^i = \sum_{j=1}^{t_i} F^i(j) \tag{7}$$

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

$i-1$ 번째와 i 번째 checkpoint 사이에서 TMR 고장이 일어났을 때 k 번째 재실행 후의 상태 확률을 계산하기 위한 초기 상태 확률 행렬 $P_j^{i\ retry(v)}(0)$ 과 v 번째 재실행 실시 후 $i-1$ 번째와 i 번째 checkpoint 사이의 j 번째 보팅 순간에서의 상태 확률 행렬은 식 (9)과 같이 유도된다.

$$P_j^{i\ retry(v)}(0) = [\pi_{j\ S(0,2)}^{i\ retry(v)}(0) \;\; 0 \;\; \pi_{j\ S(1,2)}^{i\ retry(v)}(0) \;\; \pi_{j\ S(4,2)}^{i\ retry(v)}(0) \;\; 0 \;\; 0 \;\; \pi_{j\ S(2,2)}^{i\ retry(v)}(0) \;\; 0 \;\; \pi_{j\ S(5,2)}^{i\ retry(v)}(0) \;\; \pi_{j\ S(8,2)}^{i\ retry(v)}(0)]^T\}$$

$$\text{for} \;\; \pi_{j\ n}^{i\ retry(v)}(0) \in B_{j\ Total}^{i\ retry(v-1)}(r_t) \;,\; B_j^{i\ retry(0)}(r_t) \equiv B_j^i(r_t)$$

$$T(r_t) \cdot P_j^{i\ retry(v)}(0) = \begin{bmatrix} A_j^{i\ retry(v)}(r_t) \\ ----- \\ B_j^{i\ retry(v)}(r_t) \end{bmatrix} \tag{9}$$

$i-1$ 번째와 i 번째 checkpoint 사이에서 발생한 TMR 고장에 대해 K_T 번의 재실행이 실패한 후 rollback 기법이 적용될 확률과 rollback을 위한 초기 상태확률행렬을 유도하면 식 (10)와 같다.

$$B_{Total}^{r(0)(i)} = \sum_{j=1}^{t_i} \left[B_j^{i\ retry(K_T)}(r_t) + B(X_j^i - r_t)\right]$$

$$P_1^{rk(i)}(0) = [\pi_{S(0,2)}^{rk(i)}(0) \;\; 0 \;\; \pi_{S(1,2)}^{rk(i)}(0) \;\; \pi_{S(4,2)}^{rk(i)}(0) \;\; 0 \;\; 0 \;\; \pi_{j\ S(2,2)}^{i\ retry(l)}(0) \;\; 0 \;\; \pi_{j\ S(5,2)}^{i\ retry(l)}(0) \;\; \pi_{j\ S(8,2)}^{i\ retry(l)}(0)]^T \tag{10}$$

$$\text{for} \;\; \pi_n^{rk(i)}(0) \in B_{Total}^{r(k-1)(i)}$$

k 번째 rollback 후에 $i-1$ 번째와 i 번째 checkpoint 사이에의 j 번째 보팅 순간에서의 상태 확률 행렬 $P^{rk(i)}(X_j^i)$ 은 식 (11)과 같이 유도할 수 있다.

$$P^{rk(i)}(X_j^i) = T(X_f) \cdot P_j^{rk(i)}(0) = \begin{bmatrix} A^{rk}(X_j^i) \\ --- \\ B^{rk}(X_j^i) \end{bmatrix} \tag{11}$$

$$P_j^{rk(i)}(0) = \left[A^{rk}(X_{j-1}^i) \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0\right]^T,\;\; 2 \le j \le t_i$$

k 번째 rollback 후 $i-1$ 번째와 i 번째 checkpoint 사이에서 TMR 고장이 발생했을 경우에, 검출될 평균 거리 $R^{rk}(i)$ 는 식 (12)과 같이 유도된다.

$$F^{rk(i)}(j) = \sum B^{rk}(X_j^i),\ 1 \le j \le t_i,\ F_{Total}^{rk(i)} = \sum_{j=1}^{t_i} F^{rk(i)}(j)$$

$$R^{rk}(i) = \frac{1}{F_{total}^{rk(i)}}\left(\sum_{j=1}^{t_i} j \cdot (X_f + T_v) \cdot F^{rk(i)}(j)\right) \tag{12}$$

또한, k번째 rollback 실시 후 $i-1$ 번째와 i 번째 checkpoint 사이에서 TMR 고장이 다시 검출될 전체 상태 확률 행렬 $B_{Total}^{rk(i)}$ 은 $\sum_{j=1}^{t_i} B^{rk}(X_j^i)$ 와 같으며, k번째 rollback 후에 고장 상태에서 정상 상태로 천이된 상태 확률 행렬 $A_{Total}^{rk(i)}$는 $A^{rk(i)}(X_{t_i}^i)$와 같다. 만약 TMR 고장이 K_B 번의 rollback 후에도 다시 검출되었다면 TMR 고장의 원인이 된 결함의 성질은 쉽게 파악될 수 없다. 그러나 일시적인 결함에 의해 유도된 TMR 고장이 K_B 번의 rollback 후에도 회복되지 않을 확률과 rollback을 실시하는 동안에 다시 TMR 고장이 연속으로 일어날 확률은 매우 작기 때문에 이러한 고장들은 영구적인 결함에 의해 일어났다고 가정하며, K_B 번의 rollback 후의 첫 번째 보팅 순간에서 검출된다고 가정한다. 실제적으로 재구성을 실시하기 전의 rollback의 횟수를 결정하는 문제도 중요한 문제가 될 수 있다. 즉, K_B 번의 rollback 주기보다 지속시간이 긴 일시적 결함은 시스템의 관점에서 영구적 결함과 같은 영향을 미치게 되므로, 제안된 모델은 실제적 모델과 크게 다르지 않다. $\sum B_{Total}^{rK_B} \simeq \sum B^{rK_B}(X_1^i)$ 라고 근사화 되며, K_B 번의 rollback 후에 TMR 고장이 발생했을 때 검출되는 시간상의 거리는 X_f 라고 정의한다. i 번째 checkpoint 순간에서 K_B 번의 rollback 후에 갱신된 상

태 확률 $\pi_n^{u(i)}$ 은 제안한 방법에 대해 식 (13)와 같이 유도된다.

$$\pi_n^{u(i)}=\begin{cases}\pi_n^i+\sum_{v=1}^{K_T}\pi_n^{i\ retry(v)}+\sum_{k=1}^{K_B}\pi_n^{rk(i)} & \text{for } n\in[S(0,0),\ S(0,1),\ S(1,1),\ S(4,1)]\\ 0 & otherwise\end{cases} \tag{13}$$

$$P_1^{i+1}(0)=\left[\pi_{S(0,0)}^{u(i)}\ \ \pi_{S(0,1)}^{u(i)}\ \ \pi_{S(1,1)}^{u(i)}\ \ \pi_{S(4,1)}^{u(i)}\ \ 0\ 0\ 0\ 0\ 0\ 0\right]^T$$

$$\pi_n^i\in \boldsymbol{A}_{Total}^{(i)},\ \pi_n^{i\ retry(v)}\in \boldsymbol{A}_{Total}^{i\ retry(v)},\ \pi_n^{rki}\in \boldsymbol{A}_{Total}^{rk(i)}$$

식 (13)에서 $\pi_n^{i\ retry(v)}$ 는 프로그램이 i 번째 checkpoint 순간에서의 v 번째 재실행 후 n 상태에 존재할 확률을 나타내고, $\pi_n^{rk(i)}$ 은 i 번째 checkpoint 순간에서 k 번째 rollback 후에 n 상태에 존재할 확률을 나타낸다. i 번째 checkpoint 순간에서의 갱신된 확률 $\pi_n^{u(i)}$ 은 i 번째와 $i+1$ 번째 checkpoint 사이에서의 상태 확률을 유도하기 위한 초기 상태 확률값들이 된다.

4. 전체 작업의 평균 실행 시간 유도

작업이 m 개의 checkpoint들과 n 개의 보팅들과 함께 수행되고, w_i $(1\le i\le m)$을 작업의 처음부터 i 번째 checkpoint가 끝났을 때까지의 시간, 그리고 W_i 를 $W_i=E(w_i)$, 즉 w_i 의 평균값이라고 정의하면, $w=w_m$ 과 $W=W_m$ 은 각각 전체작업의 실행시간과 평균값이 된다. v_i 를 $i-1$ 번째와 i 번째 checkpoint 사이의 작업 실행 시간을 나타낸다고 정의하면, 재실행, rollback과 재구성 후의 w_i $(2\le i\le m)$ 는 식 (14)과 같이 회귀적으로 갱신된다.

$$w_i=w_{i-1}+v_i \tag{14}$$

v_0 기간 동안에 TMR 고장이 일어났을 때는 재실행 기법만이 적용된다. 두 번의 재실행 후에도 TMR 고장이 다시 검출된다면 재구성이 적용될 것이다. 작업이 재구성 된 후에 변수 w_1 의 확률 값은 갱신된다. 평균 실행 시간을 유도하기 위한 시간 비용들은 다음과 같다.

- T_c : checkpointing 시에 요구되는 시간 비용.
- t_{rec} : 시스템 재구성을 위해 요구되는 시간.
- t_{rest} : 재시작을 실시하기 위해 요구되는 시간.
- t_{roll} : rollback을 실시하기 위해 요구되는 시간.
- T_v : 보팅의 시간 비용.
- T_{copy} : 정상적인 모듈의 값을 고장 난 모듈에 복사하는데 요구되는 시간.

평균 실행 시간을 계산하기 위한 필수 파라미터들은 아래와 같다.

- $p(i)$: 작업의 $i-1$ 번째와 i 번째 checkpoint 사이를 재실행, rollback이나 재구성 없이 실행을 마칠 확률.
- $pv(i)$: TMR 고장이 일어났을 때 v 번째 재실행을 통해 회복될 확률.
- $hk(i)$: TMR 고장이 일어났을 때 k 번째 rollback을 통해 회복될 확률.
- $c(i)$: TMR 고장이 재구성을 통해 회복될 확률.

$$T(i)=\sum P_1^i\left(X_{t_i}^i\right),\ c(i)=\frac{\sum B_{m1\ Total}^{rK_B(i)}}{T(i)}$$

$$pv(i)=\frac{\sum \boldsymbol{A}_j^{i\ retry(v)}(r_t)}{T(i)},\ hk(i)=\frac{\sum A^{rk}\left(X_{t_i}^i\right)}{T(i)}$$

$$p(i)=\frac{\sum A\left(X_{t_i}^i\right)}{T(i)}$$

$$p(i)+\sum_{v=1}^{K_T}pv(i)+\sum_{k=1}^{K_B}hk(i)+c(i)=1$$

와 같이 되고, $A(1)=t_1\cdot(X_f+T_v)+T_c$ 라 하면

$$w_1=\begin{cases}A(1) & with\ \ p(1)\\ A(1)+v\cdot r_t & with\ \ pv(1)\\ \sum_{d=1}^{k}R^{r(d-1)}(1)+A(1)+K_T\cdot r_t+k\cdot t_{rest} & with\ \ hk(1)\\ \sum_{d=0}^{K_B}R^{r(d)}(1)+K_T\cdot r_t+K_B\cdot t_{rest}+t_{rec}+w_1 & with\ \ c(1)\end{cases} \tag{15}$$

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

여기서, 다시 $A(i)=t_i\cdot(X_f+T_v)+T_c$ 라 하면

$$v_1=\begin{cases}A(i) & with\ \ p(i)\\ A(i)+v\cdot r_t & with\ \ pv(i)\\ \sum_{d=1}^{k}R^{r(d-1)}(i)+A(i)+K_T\cdot r_t+k\cdot t_{roll} & with\ \ hk(i)\\ \sum_{d=0}^{K_B}R^{r(d)}(i)+K_T\cdot r_t+K_B\cdot t_{roll}+t_{rec}+w_1 & with\ \ c(i)\end{cases} \tag{16}$$

식 (15)과 (16)을 사용하여 W_1 과 W_i 는 식 (17)과 (18)과 같이 유도된다.

$$W_1=\frac{1}{1-c(1)}[(A(1)(1-c(1))+K_T\cdot r_t(\sum_{k=1}^{K_B}hk(1)+c(1)) \tag{17}$$

$$+r_t\cdot\sum_{v=1}^{K_T}v\cdot pv(1)+\sum_{k=1}^{K_B}hk(1)[k\cdot t_{rest}+\sum_{d=1}^{k}R^{r(d-1)}(1)]$$

$$+(\sum_{d=0}^{K_B}R^{r(d)}(1)+K_B\cdot t_{rest}+t_{rec})\cdot c(1))]$$

$$W_i = \frac{1}{1-c(i)} W_{i-1} + \frac{1}{1-c(i)}[(A(i)(1-c(i)) + K_T \cdot r_t(\sum_{k=1}^{K_B} hk(i) + c(i))$$
$$+ r_t \cdot \sum_{v=1}^{K_T} v \cdot pv(i) + \sum_{k=1}^{K_B} hk(i) \cdot [k \cdot t_{roll} + \sum_{d=1}^{k} R^{r(d-1)}(i)]$$
$$+ (\sum_{d=0}^{K_B} R^{r(d)}(i) + K_B \cdot t_{roll} + t_{rec}) \cdot c(i))], \quad 2 \le i \le m \tag{18}$$

또한, 식 (18)을 $m-1$ 번 반복적으로 계산하면, 전체 작업의 평균 실행 시간 W_m 은 식 (19)과 같이 유도된다.

$$W_m = (\prod_{t=2}^{m} \frac{1}{1-c(t)}) W_1 + \sum_{j=2}^{m} [(\prod_{t=j}^{m} \frac{1}{1-c(t)})(A(j)(1-c(j)) \tag{19}$$
$$+ K_T \cdot r_t(\sum_{k=1}^{K_B} (hk(j) + c(j)) + r_t \cdot \sum_{v=1}^{K_T} v \cdot pv(j)$$
$$+ \sum_{k=1}^{K_B} hk(j) \cdot [k \cdot t_{roll} + \sum_{d=1}^{k} R^{r(d-1)}(j)]$$
$$+ (\sum_{d=0}^{K_B} R^{r(d)}(j) + K_B \cdot t_{roll} + t_{rec}) \cdot c(j))]$$

5. 평균 실행시간 비교 분석

본 장에서는 TMR 시스템에 제안된 방법을 적용시킨 후 해석적 방법에 의해 계산된 수치값과 난수 발생에 의한 컴퓨터 시뮬레이션 결과값들을 비교한다. 그리고 여러 가지 시스템 파라미터들을 변화시키면서 재실행과 rollback 횟수 및 제안된 방법과 일반적인 고장 회복 기법을 적용한 경우의 실행 시간을 비교한다. 해석적 방법에 의한 수치 계산에 사용되는 시스템 파라미터들의 값은 식 (20)과 같다.

$$T_c = 0.5,\ t_{rec} = 1,\ t_{rest} = 0.2,\ t_{roll} = 0.2 \tag{20}$$
$$T = 100,\ T_v = 0.005,\ r_t = 0.02$$

시스템이 정상 상태에서 작업을 시작하여 TMR 고장이 일어났을 때 $S(1,2)$, $S(2,2)$, $S(4,2)$, $S(5,2)$ 및 $S(8,2)$의 상태들에 존재할 조건부 확률값의 해석적 방법에 의한 수치 계산값과 난수 발생에 의한 컴퓨터 시뮬레이션 결과의 비교는 <표 1>에 나타난다. <표 1>의 결과를 보면 $S(1,2)$가 발생했을 때, 즉 한 개의 일시적 결함에 의한 TMR 고장 확률이 약 87%, $S(2,2)$가 발생했을 때, 즉 두 개의 일시적 결함에 의한 고장 확률이 약 10%로 일시적 결함에 의한 TMR 고장 발생이 90% 이상임을 확인할 수 있다. 이로부터 TMR 고장에 대한 시간여분 기법의 적용이 타당함을 알 수 있다.

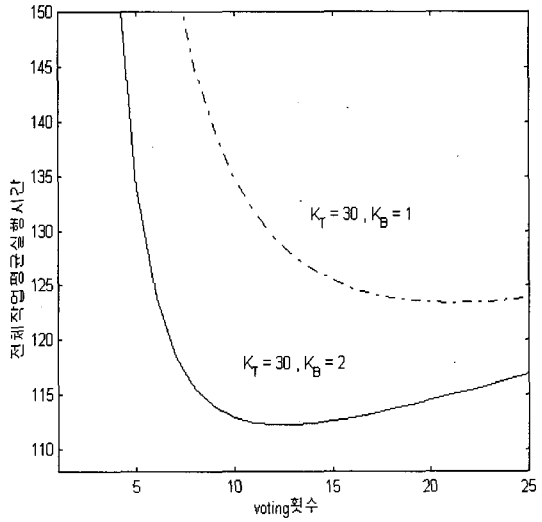
〈표 1〉 TMR 고장이 일어났을 때의 고장 상태들의 조건부 확률값 비교

($\lambda_n = 0.02,\ \lambda_p = 0.0002,\ \mu = 0.5$, 시뮬레이션 횟수 : 50000)

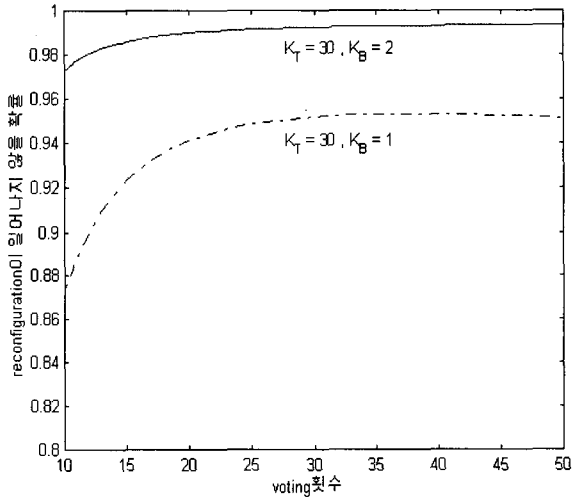
	해석적 수치결과	시뮬레이션결과	오 차
$S(1,2)$	0.8700	0.8706	0.0689%
$S(2,2)$	0.1054	0.1048	0.5725%
$S(4,2)$	0.0087	0.0088	1.1364%
$S(5,2)$	0.0157	0.0156	0.6410%
$S(8,2)$	1.463×10^{-4}	1.253×10^{-4}	16.7598%

(그림 4)에서는 제안된 방법에서 rollback의 횟수에 따른 전체 작업 평균 실행시간을 나타내었다. 그림에서 볼 수 있듯이 rollback의 횟수가 클수록 전체 작업 시간이 줄어듦을 알 수 있다. 또한 (그림 5)는 rollback의 횟수가 커질수록 재구성이 일어나지 않을 확률이 커짐을 보여주고 있다.

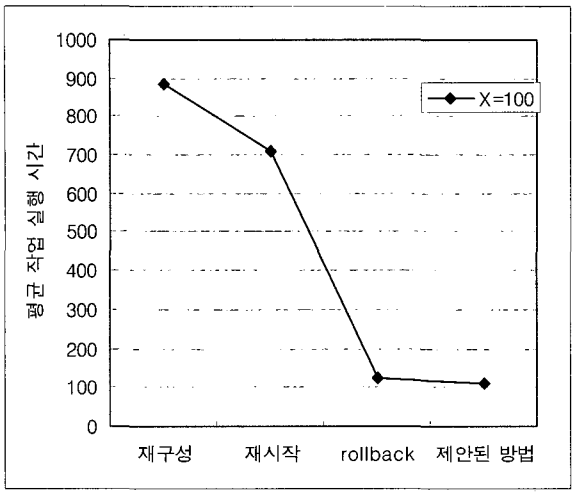
(그림 6)은 본 논문에서 제안된 방법을 TMR 시스템에 적용한 경우와 일반적인 고장 회복 기법인 재구성 기법, 시간여분 기법의 형태인 재시작 기법과 rollback 기법이 각각 단일하게 적용된 경우의 작업시간에 따른 각각의 평균 실행 시간의 비교를 나타낸 것이다. 시뮬레이션 결과에서는 TMR



(그림 4) 여러 가지 (K_T, K_B)값에 따른 전체 작업 평균 실행 시간 ($\lambda_n = 0.02,\ \lambda_p = 0.0002,\ \mu = 1$)



(그림 5) 여러 가지 (K_T, K_B)값에 따른 재구성이 일어나지 않을 확률 ($\lambda_n = 0.02,\ \lambda_p = 0.0002,\ \mu = 1$)



(그림 6) 재구성, 재시작, rollback 기법과 제안된 방법의 평균 실행 시간 비교

고장의 복구에 재실행과 rollback 기법을 통합하여 적용한 기법이 다른 시간 여분 기법을 적용했을 때에 비해 매우 효율적으로 TMR 고장을 복구하는 것을 볼 수 있다.

6. 결 론

본 논문에서는 기존의 TMR 시스템에 시간여분 기법인 재실행과 rollback 기법을 혼용하여 TMR 시스템만으로는 복구하기 힘들었던 TMR 고장의 복구 방안에 대해 제안하였다. 먼저 고장 검출시의 모든 시스템의 상태 확률들을 Markov-chain을 이용하여 추정한 후 TMR 고장을 회복하는 해석적 연구를 진행하였다. 재실행 기법에서는 차폐오류와 TMR 고장 모두에 대해 재실행을 하였으며, rollback 기법에서는 전체작업 평균 실행시간과 여분 모듈의 재구성이 일어나지 않을 확률값을 계산하였다. 제안된 재실행과 rollback 기법의 혼용 방법에서는 전체작업 평균 실행시간을 최소화하고 재구성일 일어나지 않을 확률이 최대가 될 수 있도록 하는 재실행과 rollback 기법의 시도 횟수를 유도하였다.

시스템의 효율화 문제에서 컴퓨터 시뮬레이션 및 해석적 방법에 의한 결과를 통해 볼 수 있듯이 TMR 시스템에서 일시적이 결함으로 야기된 TMR 고장은 제안된 통합 시간 여분 기법에 의해 매우 효과적으로 복구될 수 있음을 알 수 있다. 이 혼용기법은 재실행만 적용 시 이를 위해 빈번한 보팅을 함으로써 발생하는 시간비용과 보터 고장 확률 증가와 rollback만의 적용시 발생하는 시간비용의 증가를 단축시킬 수 있을 것으로 기대된다. 결론적으로 제안된 방법은 다른 회복 기법보다 여분의 모듈의 사용 없이 TMR 고장을 매우 효과적으로 회복할 수 있으며, 이러한 방법은 다른 회복 기법에 비해 적은 수의 여분 모듈을 가지고 시스템의 신뢰도를 높일 수 있는 설계 방향을 제시할 수 있다.

향후에는 TMR 시스템의 고장을 복구하기 위하여 재실행 기법, rollback과 roll-forward 기법을 혼용하여 적용했을 때의 비교 분석을 통하여 시스템 효율화 관점에서의 신뢰도를 검증할 것이다.

참 고 문 헌

[1] A. Hopkins Jr., T. Smith III, and J. Lala, "FTMP-a highly reliable fault-tolerant multi-processor for aircraft," Proceedings of the IEEE, Vol.66, No.10, pp.1221-239, October, 1978."

[2] M. Kameyama and T. Higuchi, "Design of dependent-failure-tolerant microcomputer system using triple-modular redundancy," IEEE Trans. on Computers, Vol.C-29, No.2, pp. 202-205, February 1980.

[3] P. Ezhilchelvan, J. Helary, M. Raynal, "Building responsive TMR-based servers in presence of timing constraints," Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05), pp.267-274, May, 2005.

[4] Yu. Shu-Yi, E.J McCluskey, "On-line Testing and Recovery in TMR Systems for Real-Time Applications," International Test Conference (ITC2001), pp.240-249, Oct., 2001.

[5] H. Choo, H. Youn, S. Yoo, "Two-dimensional TMR with partial majority selection and forwarding," Proceedings of the IEEE International Symposium on ISIE2001, pp.482-487, June, 2001.

[6] C. Ramamoorthy and Y. Han, "Reliability analysis of systems with concurrent error detection," IEEE Trans, Computers, Vol.24, No.9, pp.868-878, Sept., 1975.

[7] X. Zhuo and S. Li, "A new design method of voter in fault tolerant redundancy multiple-module multi-microcomputer system," Digest of Papers FTCS-3, pp.472-475, June, 1983.

[8] N. Gaitanis, "The design of totally self-checking TMR fault-tolerant systems," IEEE Trans. Computers, Vol.37, No. 11, pp.450-1454, Nov., 1988.

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

[10] Y. Lee and K. Shin, "Optimal design and use of retry in fault-tolerant computing systems," Journal of the ACM, Vol. 35, pp.45-69, January, 1988.

[11] P. Chande, A. Ramani, and P. Sharma, "Modular TMR multiprocessor system," IEEE Trans. on Industrial Electronics, Vol.36, No.1, pp.34-41, February, 1989.

[12] 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.

[13] 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.

[14] J. Yoon and H. Kim, "Time-redundant recovery policy of TMR failures using rollback and roll-forward methods," IEE Proc.-Comput. Digit. Tech, Vol.147, No.2, pp.124-132, March, 2000.

[15] H. Kim and K. Shin, "Evaluation of Fault Tolerance Latency from Real-time Application's Perspectives," IEEE Transactions on Computers, Vol.49, No.1, January, 2000.

[16] I. Koren, Z. Koren and S. Y. H. Su, "Analysis of a Class of Recovery Procedures," IEEE Transactions on Computers, Vol.C-35, No.8, August, 1986.

[17] 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.

강 명 석

e-mail : mskang@yonsei.ac.kr
2001년 원광대학교 컴퓨터공학과(학사)
2003년 원광대학교 컴퓨터공학과(석사)
2003년-현재 연세대학교 전기전자공학과 박사과정
관심분야 : 실시간 고장포용 시스템, 지능형 홈네트워크

손 병 희

e-mail : diana@yonsei.ac.kr
1995년 우석대학교 계산통계학과(학사)
1995년-2001년 ㈜한국프로페이스 기술지원팀 주임
2000년~2003년 연세대학교 공학대학원 전기공학과 (석사)
2005년~현재 연세대학교 대학원 전기전자공학과 박사과정
관심분야 : 실시간 시스템 Hard-deadlines, Reliability

김 학 배

e-mail : hbkim@yonsei.ac.kr
1988년 서울대학교 전자공학과(학사)
1990년 미국 미시간대학교 전기 및 컴퓨터공학과(석사)
1994년 미국 미시간대학교 전기 및 컴퓨터공학과(박사)
1996년~현재 연세대학교 전기전자공학과 부교수
관심분야 : 실시간 시스템, 인터넷 웹서버 기술, 디지털시스템 고장포용 및 신뢰도 평가분야