기술해설

실시간 시스템 및 제어 컴퓨터의 기본 특성

김 학 배*
(* 연세대 공대 전기공학과 조교수)

1. 서 론

제어 시스템의 일부분으로서 컴퓨터를 실시간 운용한 최초의 시도는 아날로그 연산 요소를 사용하는 것을 위주로 하여 1950년대에 Brown과 Campbell의 논문에 의해 제안되었다[1]. 제어 및 감시를 요하는 시스템(플랜트)은 관련 산업 기술의 발전과 더불어 그 동적 특성이나 목표 요소가 상당히 복잡해지고 있고, 이러한 이유로 기계적 부품이나 단순한 아날로그 회로 및 필터 등으로 구현되었던 시스템 제어기는 컴퓨터의 성능 향상 및 가격의 현실화라는 긍정적인 동기 부여와 구현을 위한 현대 제어이론의 뒷받침으로 급격히 디지털 컴퓨터에 의해 대체되는 중이다.

특히, 시스템의 이상 동작이 심각한 경제적인 손실이나 인명피해 등의 엄청난 결과를 야기할 수 있는 비행기, 우주선, 핵발전소, 대규모 공정제어, 및 발전소 등과 같은 시스템에는 정밀한 제어 기능과 고 신뢰도를 지닌 디지털 컴퓨터의 활용이 필수적이다. 예를 들어, 21세기에 등장할 고도의 연료절감형 비행기는 현재 조종사에 의한 수동 조작이나 기계적 제어기에 의존할 수 없는 빠르고 정확한 제어동작을 요구할 것으로 예상된다. 설계 시에 연료의 효율성이라는 목표를 추구하다 보면 불가피하게 비행기는 보다 불안정한 시스템의 동적 특성을 지니게 되는데(edge of instability), 시스템의 안정성 유지 및 비상신호에 대해 효과적인 대응을 하기 위해서는 보다 신속하고 정확하며 일관된 제어 작업이 요구된다. 물론 이러한 극단적인 예 이외에도, 많은 응용 분야에서 컴퓨터에 의한 실시간(real-time) 제어 작업을 당연시하는 시스템들이 늘어나고 있다. 일반적으로 "온라인(on-line)" 적 속성으로 단순하게 이해되기 쉬운 실시간 시스템은 시스템의 정확성(correctness)이 작업의 논리적 결과뿐만 아니라 그 결과가 생성된 시간에도 좌우된다는 독특한 성질에 의해 특성 지어진다[2]. 다시 말해, 시스템이 물리적인 주변환경에서 어떤 입력을 받아 (제어) 작업을 수행하여 출력을 내는 과정이 적시성(timeliness)을 지녀야하는, 즉 논리적으로도 정확한 출력이 데드라인(deadline)으로 일컫어지는 어떤 시간 한계치를 넘지 않아야만 시스템 동작의 정확성이 보장되어 예기치 않은 결과를 피할 수 있는 특성으로 정의될 수 있다. 이는 궤환제어 작업인 경우 센서(sensor)로부터 입력신호를 수집하여 플랜트의 구동기(actuator)에 적절한 출력신호를 데드라인[1] 이내에 제공해야만 제어작업이 유효하게 된다.

실시간 시스템의 이러한 특성을 고려할 때 제어컴퓨터는 연산 장치로서 그 자체만의(stand-alone) 성능 분석은 더 이상 의미가 없고 시스템 내의 하나의 기능적 요소로서 자체 시스템의 상황을 고려하여 설계 및 평가되어야 한다[3]. 이러한 면에서 실시간 제어 컴퓨터를 embedded computer라 부르기도 하고, 그 성능 및 신뢰도는 주변환경(플랜트를 포함한 넓은 의미에서)의 특성 측면에서 보았을 때 동적(dynamic)이고, 예측가능하며(predictable), 유연해야(flexible)한다. 따라서, 본 논문은 다음 장에서 실시간 시스템의 기본 특성에 대해 보다 자세하게 살펴본 후, 제 3장에서 플랜트의 동적 특성을 관련 제어이론을 활용하여 실시간 시스템의 가장 중요한 정보인 작업 데드라인을 유도하는 방법을 살펴본다. 제 4장에서는 효율적인 실시간 제어 컴퓨터를 설계 및 평가하기 위해 실시간 시스템 분야에서 수행되는 연구동향으로 다음과 같은 세부분야를, (i)스케쥴링, (ii)실시간 컴퓨터 언어, (iii)실시간 운영체계(O/S), (iv)실시간 동기화 (v)실시간 H/W 구조(architecture), (vi)실시간 통신, (vii)고장허용기법(fault-tolerance), 설명하고, 마지막 장에서 향후의 관련 연구 과제들에 대한 간단히 소개로 결론짓는다.

2. 실시간 시스템의 개요

전형적인 실시간 시스템은 제어되는 플랜트[2]와 제어기로 구성되는데, 자동화생산시스템을 예를 들면 공장내의 각종

1). 비실시간 시스템인 경우 데드라인은 무한한 값으로 나타나지나, 실시간 시스템의 데드라인은 제어기의 성능과는 무관하게 플랜트의 특성에 좌우된다.

2) 플랜트는 제어컴퓨터와 상호 동작하는 넓은 의미의 주변환경으로 볼 수 있다.

로봇, 조립라인 및 기기 등은 플랜트에 해당되고 이러한 공장의 모는 생산활동을 관리하고 연결시켜주는 컴퓨터 및 인터페이스 등이 제어기의 범주에 든다. 제어기는 센서를 통해 주변환경의 현재 상태에 대한 정보를 수집하고 수행하고자 하는 플랜트의 mission이 성공리에 이루어 질 수 있도록 구동기에 적절한 출력신호를 제공하기 위해 연산작업을 수행한다.

앞에서 언급한 바와 같이, 실시간 시스템에서는 이러한 단위 과정이 어떤 데드라인 내에 마쳐져야 그 연산/제어 작업이 유효하게 된다. 이러한 작업은 주기적인(periodic) 경우가 대부분이고 이는 제어기 내의 클럭에 의해 작업이 시작되는데 제어 작업이 데드라인을 어기는 경우 시스템에 심각한 영향을 주는 time-critical 속성을 지닌 hard 실시간 시스템과 단지 데드라인을 어긴 단위 제어 작업의 무효화로 시스템의 평균적 성능에 미세한 영향을 주는 soft 실시간 시스템으로 분류할 수 있다. 이 외에도 플랜트 상태에 이상 현상(미리 정해진 threshold에 의해 판단)이 감지되면 비상신호(alarm)가 제어 작업을 시작하는 비주기적인(aperiodic) 작업도 있는데[3] 이 경우는 대부분 시스템의 극한 상황을 방지하기 위해선 제어 작업의 반응시간이 데드라인을 넘지 않아야 하는 hard 실시간 시스템인 경우가 전형적이다. 시스템의 이상 현상은 주변환경의 급격한 변화나 시스템 요소의 고장(faults)으로 인해 발생하기 쉽고, 비주기적으로 발생하는 이러한 이상 현상으로부터 시스템의 안정성을 유지하기 위해 시스템 재구성(reconfiguration) 및 고장 회복 기법을 포함한 제어 작업이 따라야 하는데, 실시간적인 속성의 작업은 반드시 데드라인 안에 마쳐지도록 제어기가 설계되어야 하고, 비실시간적인 속성의 작업들에 대해선 다른 실시간 작업들의 데드라인 내의 작업 완료에 영향을 최소화하는 조건 내에서 가능한 신속하게 작업을 끝내야 한다.

이러한 면들을 고려할 때 실시간 시스템은 다음과 같은 세 가지 중요한 특성들을 내포함을 관찰할 수 있다: (i)시간은 실시간 시스템에서의 다루어지는 가장 중요한 자원이다. 따라서, 앞서 분류된 가능한 여러 제어 작업들이 데드라인을 포함한 시간 조건/정보를 바탕으로 제어기내에서 배당되거나(assign) 시간상으로 스케쥴링되어야 한다, (ii)시스템의 고장이 심각한 결과로 이어지는 hard 실시간 시스템에서는 시스템 신뢰도(reliability)는 매우 중요한 평가요소이다. 물론 soft 실시간이나 비실시간 시스템들에게는 시스템 신뢰도보다는 성능(performance)인자가 중요시되는 것 또한 부시할 수 없다. 그리고 (iii)컴퓨터의 성능이 비실시간 응용시에는 컴퓨터 자체의 절대적인 기능을 위주로 설계 및 분석이 되었으나 실시간 시스템에서는 주변환경 속에 하나의 동적 요소로서 전체 시스템의 성능 및 신뢰도에 막대한 영향을 미치는 상대적 특성으로 고려되어야 하므로 컴퓨터의 사양 선정, 설계, 및 평가 시에 주변 환경(environments, 또는 좁은 의미로 플랜트를 지칭)에 대한 명확한 정보 수집 및 분석이 선행되어야 한다.

실시간 시스템의 또다른 중요한 특성 중에 하나는 제어기의 작업(입력수집, 연산/통신, 출력변환)수행이 예측가능(predictable)해야 한다는 것이다. 다시 말해서, 시스템 내에서 어떠한 가정(예를 들어, 주어진 시간 내에 가능한 시스템 요소 고장의 수가 threshold가 되는 어떤 수보다 작다는)이 만족된다면 해당 응용 작업들에 대한 모든 시간적 제한이 지켜져야 한다는 것을 제어기 설계시에 확신할 수 있어야 한다. 물론 예측에 대한 100% 확신을 위해서는 사전에 모든 제어 작업들에 대한 정보(갯수, 데드라인, 연산 자원, 작업간 선행 관계 등)와 작업수행 시에 발생할 수 있는 가능한 전체 환경변화에 대한 정보들을 알고 있어야 하나, 동적인 환경의 특성과 stochastic한 제어 작업의 속성들을 비추어 볼 때, 이것은 단순한 제어작업이 주기적으로 반복되는 경우와 같은 특수한 경우를 제외하고는 비현실적이다. 따라서 보다 현실적인 의미로 "확률적(probabilistic)" 또는 "실행중 결정적(run-time deterministic)"인 의미로 predictability를 해석할 수 있다. 전자는 전체 작업중 일부분이 확실히 데드라인을 지킨다는 의미이거나 각각의 작업이 어떤 확률값을 갖고 데드라인을 지킨다는 의미로 해석될 수 있고, 후자는 한 작업이 시작 전에 그 때의 시스템의 부하상황을 감안하여 그 작업이 데드라인을 지킬 수 있을 지의 여부를 검사해 작업의 시작이나 포기를 결정하는 것으로 이는 제어기의 설계시에는 예측이 불가능하나 시스템 운용 중에는 각 작업에 대한 예측을 동적으로 가능하게 하므로 수시로 변화하는 비주기 작업이나 동적 부하공유(load sharing)에 유용한 의미이다.

실시간 시스템에서의 제어 작업들에 대한 시간이나 예측가능성에 관한 특성/제약조건 외에도 고려해야 할 다른 제약성(constraint)이 있다. (비실시간 시스템에서도 성능개선을 위한 설계 및 평가에 다음의 제한조건 정보가 필요함)

- 자원(resource)제약성 : 작업실행을 위해 제어기내의 프로세서(CPU), 입출력(I/O) 디바이스, 통신 네트웍, 파일, 및 데이터 베이스/구조 등의 자원이 필요
- 선행(precedence)제약성 : 하나의 복잡한 작업은 복수개의 하위 작업(subtask)들로 나누는 경우 각 하위작업간에 순서에 주의
- 동시성(concurrency)제약 : 만약 여러 작업들이 하나의 자원을 동시에 이용하는 경우(예, 동일한 센서값을 여러 작업들이 수집)에는 일관성(consistency) 문제에 유의
- 통신 요구 : 연산 속도 향상을 위해 분산 작업인 경우 제어기 네트웍 구조를 바탕으로 자료교환이 필요

3) 이외에도 customer나 operator에 의해 작업이 시작될 수도 있다. 전자의 예는 데이터베이스 시스템에서의 transactions을 고려할 수 있고 후자로는 수동(manual)모드로의 제어작업 전환 시이다. 이런 경우들은 주로 시스템의 극한 상황을 고려하기보다 평균적이 성능이 주 관심이 되는 soft 실시간 시스템인 경우가 대부분이다.

- dependability와 성능 제약 : 각 작업에 대해 신뢰도, availability, 및 성능인자에 대해 기준/목표치 필요

이러한 실시간 시스템과 작업들의 특성 및 제약성들 중 가장 중요한 것은 역시 시간에 대한 제약성이고 이는 플랜트, 즉 주변환경의 특성에 의해 결정된다.

3. 제어 작업의 데드라인 정보

실시간 시스템에서의 시간 제약성, 즉 데드라인 정보는 그 중요성에도 불구하고 제어기설계 시에 미리 결정된 값으로 가정되거나, 간단한 모의 장비를 이용한 원시적 실험을 통해 부적절하게 측정되어 왔는데[3,4,5,6] 이런 유도방법들은 효율적인 제어기의 설계나 정확한 시스템신뢰도의 평가에 장애가 되어왔다. 데드라인유도의 분석적 방법으로는 제어기의 제어법칙계산 지연시간의 시스템안정성(stability)에의 영향을 분석하는 연구가 주로 수행되었다[4,5]. 또한 이동로보트제어의 특수한 정적상황에서 역시 시스템의 안정성을, 제한된 가정 하에서 유도하는 연구도 진행되었다[6]. 그러나 이러한 모든 기존의 유도방법들은 현실적인 계산 가능을 위해 단순히 컴퓨터제어기의 지연시간이 기본시간단위인 샘플링 주기의 한두 배일 때의 상황만을 고려하거나, 기본적으로 랜덤(stochastic)변수인 데드라인을 고정된 상수로 취급하는 비합리적인 가정 하에서만 유효하다. 물론 컴퓨터 프로그램을 이용한 시뮬레이션을 통해 항공기의 착륙 시에 제어기의 데드라인을 수치적으로 완벽하게 유도한 연구도 있으나[3], 이 또한 사용된 가정의 제한성 때문에 데드라인유도의 일반적 방법이라고 보기 어렵다. 이러한 기존의 데드라인정보 유도방법들의 한계를 극복하기 위해 플랜트의 동적 성질과 제어기의 반응지연의 주된 원인인 제어법칙계산시간 및 제어기의 내적/외적 고장 등의 발생특성(빈도 및 지속주기)이 고려되며 플랜트의 데드라인을 유도하는 회귀(recursive)법을 응용한 수치적 방법이 제안되었다[7,8]. 실시간제어시스템의 가장 단순한 모델인 선형불변(LTI)시스템의 귀환(feedback)제어로부터 실제적 모델인 인공위성궤도제어, 로켓트발사궤적제어, 그리고 이동로보트 충돌방지제어 등에서 단순화과정을 거쳐 각각의 데드라인을 시간 및 상태의 함수로서 구하는 예로써, 제시된 방법의 유용성이 검증되었다.

앞에서 언급한 바와 같이 실시간시스템에 있어 데드라인은 주어진 작업의 결과가 그 효력을 갖기 위해 반드시 지켜야하는 시간의 한계치이고, 이 값을 유도하기 위해 먼저 작업의 무효화나 비정상적인 결과에 의해 발생하는 시스템의 고장 및 극한 상황에 대한 정의가 구체화되어야 한다. 먼저 수학적으로 다루기 쉬운 형태로 시스템의 안정성(stability) 유지조건이 거론될 수 있다. 제어플랜트의 안정성은 선형불변(LTI)인 경우 시스템 전달함수나 상태방정식을 통해 유도된 고유치(eigenvalue)를 검사함으로써 간단히 안정성 유지여부를 확인할 수 있다. 이 경우에 시스템 전달함수나 상태방정식을 적절하게 변환하여 제어작업의 무효화로 인한 영향이 내포되도록 공식화하고 그로 인해 변환된 시스템의 동적 특성을 관찰하며 시스템의 안정성을 검사한다. 데드라인에 해당하는 변수D의 값을 변화/증가시키면서 --- 물론, 이 경우 변화된 데드라인에 따라 제어작업의 무효화정도도 변화하게 됨이 주시된다 --- 결과로 얻어진 시스템 전달함수나 상태방정식의 안정성을 위의 방법으로 연속해서 검사해 안정성을 잃게 하는 최소치의 변수 $D_{\min}$가 시스템의 해당 작업에 대한 데드라인이 된다. 이를 수학적으로 공식화해 표현하면 아래 식 (1)과 같이 데드라인이 정의되며 이를 구하는 기본과정은 아래와 같이 정리된다.

$$D(X) = \sup_{U(k-N)\in U_A} \{ N : \lambda_{\max}(k, X, U(k-N)) < 1 \} \quad (1)$$

위 식에서 k, X, U, D 는 각각 시간, 시스템 상태, 입력, 데드라인을 나타내는 변수들이고 U_A는 가능한 입력 공간, 그리고 $\lambda_{\max}$는 k, X, U, D 등에 좌우되는 시스템의 최대 고유치이다. 위의 정의를 바탕으로 다음과 같은 단계를 거쳐 플랜트의 동적 특성에 좌우되는 제어 작업의 데드라인을 유도할 수 있다.

① 정상적인 제어작업시의 시스템의 상태방정식 모델링
② 데드라인 변수의 정의 및 최소시간단위로의 가정
③ 데드라인 간과시의 작업무효화내포를 위한 상태방정식 변환
④ 변환된 상태방정식에 대한 고유치 유도를 통한 안정성 검사
⑤ 안정성손실시까지 ②부터 ④까지의 단계의 반복
⑥ 안정성을 잃게 하는 최소치의 변수값을 데드라인으로 확정

위의 과정은 해당 응용시스템에 따라 구체화될 수 있으며 만약 시스템에 대한 기본가정이 변한다면 이에 따르는 단계들도 적절하게 수정되어야한다.

시스템안정성 손실 못지 않게 시스템의 허용공간이탈이 데드라인유도를 위한 시스템의 고장 또는 극한상황의 정의될 때가 있다. 예를 들어, 착륙하는 항공기의 피치(pitch)각 등의 중요 각도들이 허용범위 내에 머물러야 지면과의 충돌을 피할 수 있으며, 자동화된 공장의 물품이동을 위한 로봇은 산재해있는 장애물들을 피하기 위해서는 역시 허용공간을 항상 유지해야 한다.

$$D(X(k_0)) = \sup_{U(k-N)\in U_A} \{N: \Phi(k, k_0, X(k_0), U(k-N)) \in X_A\} \quad (2)$$

위 식에서 X_A는 시스템상태의 허용공간으로 정의되고 시간 k_0에서 출발한 시스템의 궤적은 지연시간이 N일 때 식(3)

에 의해 전개된다.

$$X(k_0)=\Phi(k, k_0, X(k_0), U(k-N)) \quad (3)$$

이 경우의 데드라인은 같은 방법에 의해 식(2)와 같이 정의되며 위의 유도과정에서 네 번째 단계의 안정성검사 대신 허용공간이탈여부가 매 주기당 검사됨으로 대체되는 것 이 외에 나머지는 같은 절차 및 방식이 적용된다. 보다 구체적으로 이 경우에 매 주기당의 시스템의 궤적을 주어진 동적 특성 방정식으로부터 계산하고 이의 허용공간에의 이탈여부를 각 주기에 대해 조사하는 방법을 취하는 것을 고려할 수 있는데 경우에 따라 허용공간이 쉽게 얻어질 수도 있지만 대부분의 경우 해석적인 방법에 의한 이의 유도는 불가능하다. 따라서 해당 허용공간의 공식적인 유도절차 또한 관심을 끄는 연구과제이고 이의 유도는 대략 수치적으로 다음과 같은 개념 하에서 이루어진다. 만약 시스템의 최종상태가 X_f이고 그 허용공간이 다음 조건(terminal constraint)에 의해 제한될 때,

$$X(k_f) \in X_A^f \leftrightarrow X(k_0+N) \in X_A^2 \quad (4)$$

위 식에서 X_A, 즉 허용공간은 상태방정식으로부터 유도되는 시스템 궤적의 역 추적으로부터 유도된다.

$$X_A^0=\{ X \mid \Phi(k_f, k_0, X(k_0), U(k_f-N)) \in X_A^f \} \quad (5)$$

4. 실시간 제어컴퓨터의 특성

4.1 실시간 스케쥴링

스케쥴링은 주어진 작업들과 작업수행에 필요한 시스템 내의 자원이 주어졌을 때 각각의 작업들의 실행 장소 및 시간을 결정하는 과정이다. 실시간 시스템에서는 스케쥴링 의해 명백하게 좌우되는 작업 완료 시간을 제한성을 만족시켜야 하는 것이 가장 기본적인 문제이므로, 스케쥴링은 실시간 시스템에서 널리 연구되어온 과제이다. 스케쥴링은 Operational Research(OR)에서도 job- shop이나 flow-shop과 같은 모델을 대상으로 연구되어져 왔는데, 이 경우에는 기계장비나 공장 cell 등과 같은 시스템 자원에 작업 수행에 필요한 자원 사용 순서 결정이 주된 스케쥴링 문제로서 스케쥴링에 걸리는 시간의 최소화를 위해 스케쥴링 테이블 구성에 의한 정적인 모델을 이용하여 관련 비용, 전체 작업 완료 시간, 평균 tardiness, 및 tardy 작업수의 최소화 등을 목표로 하는 것이 대부분의 추세이다[9]. 이에 반해, 실시간 제어 컴퓨터 설계를 위한 스케쥴링은 예측가능성(predictability)의 극대화와 작업의 적시성(timeliness)의 보장을 목표로 CPU, 기억장소, 및 데이터 링크 등의 효율적인 컴퓨터 시스템 자원 분배에 치중하여 동적인 스케쥴링 개발에 초점을 맞춰 왔다. 물론, 시스템 종류에 따른 요구조건과 평가항목(metric)이 다르고 작업연산시간, 필수 자원, 우선순위(priorities 또는 criticalities)에 따른 중요성 수준, 선행(precedence)관계, 통신 요구, 및 시간 제한 조건 등의 작업 특성의 다양성 때문에 스케쥴링 문제는 더욱 복잡하다. 실시간 시스템에서는 주어진 작업이 시간 제한 조건(데드라인)을 만족시키는 지의 여부를 검사하는 schedulability 분석[10]의 수행여부, 이러한 분석이 어떤 모드로 행해지는 지(즉, 고정된 방법 또는 동적인 모드), 그리고 분석결과가 직접 새로운 스케쥴 및 계획(plan)을 세우는 데에 활용여부에 의해, 모든 실시간 스케쥴링 기법들을 다음과 같은 네 가지 종류로 분류한다.

- static table-driven 방식 : static schedulability 분석과 휴리스틱(heuristic) 방법에 의해 실행중 작업시작 시간을 결정하는 테이블 형태의 스케쥴러를 off-line으로 구성하는데 주로 주기적인 작업에 응용되고 동적 환경에 유연하지 못한 단점이 있다[11,12].

- static priority-driven preemptive 방식 : 전형적인 비실시간 시스템의 비주기성 작업에 응용된 방식으로 스케쥴링은 외향적인(explicit) 스케쥴러보다는 작업의 우선순위(priority)에 의해 결정되는데, 실시간시스템에서는 schedulability 분석을 선행하여 해당작업의 수행 적합성을 먼저 검사한다. 가장 사용빈도가 높은 작업에 우선순위를 부여하는 rate-monotonic 스케쥴링[13]과 cycling 스케쥴링[14]이 이 방식의 대표적인 예이다.

- dynamic planning-based 방식 : 동적으로 발생하는 작업에 대해 시간제한성이 만족될 수 있는 가능성 여부(feasibility)가 실행 중에 체크되고 admission 제어를 통해 개별적으로 생성되는 각각의 작업에 대해 predictability를 보장한다[15].

- dynamic best effort 방식 : feasibility 검사는 하지 않고 시스템은 작업 특성에 의해 동적인(dynamic) 우선순위를 계산하고 이에 의해 각 작업 시작 시간이 결정된다. 이 방식은 시뮬레이션에 의해 그 유용성이 검증되며 그 결과에 따라 작업의 우선수위를 갱신(update)하지만 각 작업이 데드라인 조건을 지키는지의 여부에 대한 보장을 할 수 없고 작업수행이 완료이전에도 멈추어 질 수 있다[16].

embedded 시스템은 점차 그 주종이 고성능 및 고신뢰도의 다중프로세서 컴퓨터로 전환되고 있기 때문에, 전형적으로 dynamic task arrivals와 주기적, 비주기적인 작업에 대한 on-line 스케쥴링을 가정한다면 스케쥴링 알고리듬들과 스케쥴러 설계는 병렬 실행 platform으로 향한다. 하나의 효율적인 on-line 다중프로세서 알고리듬의 예는 [17]에서 제안되었는데, 이것은 데드라인 스케쥴링을 수행하는 프로세서의 지역 스케쥴러들과 함께 프로세서들에게 작업들을

할당하는 전역스케쥴러들로 구성된 분산(distributed) 스케쥴러로 구현된다. 이러한 내용의 연구는 [15]에서 보다 확장시켜 심도있게 다루어지는데. 이러한 다중프로세서 스케쥴러들은 노드(node)들 사이에서는 global task를 분배하는 방법과 각 노드에서 작업의 채택과 기각의 결정 문제를 공통적으로 다루는 분산시스템상의 스케쥴러와 유사하다[18].

4.2 실시간 컴퓨터 언어

실시간 시스템의 제어 작업을 위한 프로그래밍에 활용되는 컴퓨터 언어(language)는 일반적인 경우(비실시간용)와 같이 운용자나 사용자가 내재해 있는 H/W의 구조와의 직접적인 연관이 없는 high-level이어야 하고[4], 이러한 high-level 프로그램에 응용되는 많은 formal analysis 기법이나 도구들이 활용되어야 한다. 실시간 시스템은 predictability가 가장 기본적인 특성 중의 하나이므로 프로그램 실행시간에 대한 예측성 보장을 위해 동적 구조체(structure), 순환(recursion), 및 무한 루프의 사용이 금지되거나 조심스럽게 다루어져야 하고, 모든 프로그램의 구조체 및 모듈들은 시스템의 시간제한성 만족도, 실행 자원의 이용 가능성, 프로그램 모듈의 schedulability 등에 비추어 분석될 수 있어야 한다. 또한, 동시성(concurrency)의 특성을 내포한 실시간용 컴퓨터 언어는 다음과 같은 기능들을 지니어야 한다: (i)S/W 요소로서의 모듈의 구축, (ii)작업(task)의 생성 및 관리 (iii)interrupts와 디바이스의 용이한 조절(handling), (iv)작업간의 통신, (v)배타적 상호관계(mutual exclusion), 그리고 (vi)exceptional handling. 이러한 기능들이 구현되는 방법에 의해 실시간 언어가 분류될 수 있는데, 최소한의 기능들을 지니며 필요한 경우 (각 사용자가 표준화를 고려해 또다른 기능을 생성함으로써) 확장 가능한 Moulea-2와 같은 단순한 clean 언어가 있고, 프로그래머가 선택할 수 있는 광범위의 기능들을 지닌 Ada, CONIC, 및 CUTLASS 등의 보다 복잡하며 표준화된 언어들이 있다.

4.3 실시간 운영체제 (O/S)

실시간 제어 컴퓨터의 핵심 부분인 실시간 O/S는 프로세스 관리와 동기화, 메모리 관리, 프로세스간의 통신 및 I/O 제어 등의 기능에 시간 제한성을 고려한 predictability를 부여하는 방향으로 개발되고 있다. 실시간 운영체제는 개발목적 및 기능에 따라 다음과 같은 세 가지로 분류된다: (i)small and proprietary kernels, (ii)상용 O/S의 실시간 확장용, 그리고 (iii)순수 연구용.

첫째는 매우 속도가 매우 빠르고 작업들에 대해 높은 예측성을 요구하는 응용분야를 위해 적합하나, 상당한 개발 및 유지비용이 필요하므로 대규모 응용분야에의 적용이 쉽지 않다. 기능 면에서도 속도와 예측성을 위해 최소한의 필요 부분이외는 생략했으며 특수 목적을 위한 homegrown-kernel[19,20]과 VCOS, pDOS, pSOS, VxWorks, QNX, VRTX32 등의 commercial-offsprings의 두 가지 종류로 나뉘어진다.

둘째는 UNIX, POSIX, MACH, 및 CHORUS 등의 기종의 상용 timesharing 운영체제에 timer, 우선순위 스케쥴링, 공유 메모리, 실시간 파일, semaphore, 통신, 동기 및 비동기 I/O, 비동기 event 보고, threads 등의 실시간 관련 기능들을 추가해 개발한 RT-UNIX[21]나 RT-MACH[22] 등으로 높은 이식성, 기능성, 과 S/W 개발환경의 잇점에도 불구하고 속도나 예측가능성 면에서 일반적인 실시간 응용에는 한계가 있다.

셋째로 연구용 실시간 O/S는 앞에서 설명된 두 종류 실시간 O/S 방식들의 단점을 보완하면서 새로운 paradigm 및 관련 실시간 프로세스 모델 및 이론을 활용하여 kernel level과 application level의 양쪽에서의 predictability를 강조하는 실시간 운영체제들로서 대표적 예로는 HARTOS[23], Spring[24], MARUTI[25], ARTS[26], CHAOS[27], DARK[28], 및 MAR[29] 등이 있다.

4.4 실시간 Synchronization

동기화(synchronization)는 다음의 이유 때문에 실시간 시스템에서 중요한 의미를 갖는다: (i)작업들은 그들이 상호 배타적으로 접근(exclusive access)하려는 shared resource들에 대해서 발생되는 blocking 때문에 예상치 못한 delay를 경험할 수 있고 (ii)동기화 방법들은 앞서 언급된 실시간 분야의 응용에서 중요하게 부각되는 다중자원의 작업스케쥴링에의 해결 방법을 찾아내는데 도움이 된다. Video stream과 그와 관계된 출력장치들이 서로서로 연결되게 스케쥴된 multi-media쪽에서의 응용이나 CPU 프로세싱이 센서입력 프로세싱과 동기화 되어야만 하는 군수기기 등에서의 그 응용 예를 찾을 수 있다.

실시간 동기화의 가장 기본적인 문제는 시스템의 모든 클럭들을 동기화 함으로써 global time을 설정하는 일인데, 이는 고장난 클럭의 Byzantine fault[30]에 의해 문제가 복잡해진다. 이런 고장 하에서도 동기화를 성공시키는 조건이 [31]에서 유도되었고, 이를 바탕으로 S/W 및 H/W 동기화 방법들이 제안되었다.

Consistency-based[30], convergence-averaging[32], 및 convergence-nonaveraging [33] 등의 방식으로 분류되는 S/W 동기화는 단지 클럭 메시지의 교환만을 통해 동기화를 이루는 방식으로 유연하고 경제적이나 기본 시간단위에 제한(H/W 방식보다 granularity가 큼)이 있고 최악의 경우 skew가 크므로 엄격한 실시간 응용에는 적합하지 않다. 이에 반해, phase-locked 루프의 원리에 근거해 각 노드에 특수한 H/W(기준 신호를 선정하는)를 활용해 tight한 동기화를 얻을 수 있는 H/W 방식에는 동기화가 가능한 최대 클럭 수가

4) 그러나, 시스템 S/W 및 어셈블리어(또는 기계어) 등을 통한 간접적인 연관은 고려되어야 한다.

4개로 제한된 한계성을 지닌 최초의 H/W Byzantine resilient 방식이 [34]에서 제안되었고 이런 한계를 극복한 알고리듬이 [35]에 의해 개발되었다. 또한, 이러한 결과 등에 내재한 fully connected 클럭-네트웍에 대한 불합리한 가정과 무시할 수 없는 전송(propagation) 지연을 해결하기 위한 연구가 관련 H/W의 비용 상승을 전제로 [36]에서 수행되었고, 이러한 모든 문제점의 보완과 비용절감을 위해 hybrid 방식 [37] 및 확률적 동기화 방식[38]이 제안되었다.

4.5 실시간 H/W architecture

Hard 실시간 시스템은 일반적으로 특수한 목적을 지니므로 이러한 응용 분야에서는 H/W 구조 역시 일반적이지 않은 예가 대부분이다. 실시간 시스템을 위해 개발된 다양한 architecture 들로부터[39] 이러한 특별한 목표와 기능의 구현을 위한 설계시의 많은 일반적인 개념과 원리들이 도출되었는데, 그 일부는 다음에 나열된 규칙들에 의해 설명된다.

- 일반적인 off-the-shelf 요소 및 부품들로 특수한 목적의 configuration을 개발
- 처음부터 고장허용(fault-tolerance) 및 실시간용 특성을 위주로 설계
- 다중 버스에 대한 필요성 및 기능상의 분할(functional partition) 이행
- 단위 모듈이 첨가될 때에 관련 오버헤드 증가의 시스템의 확장에 대한 심각한 영향
- 열악한 주변환경에 대한 민감성 감소를 위해 중요 코드의 ROM에의 저장
- on-line testability 기능 제공
- private 메모리의 읽기전용 코드를 위한 활용과 global 메모리의 공유 데이터 저장
- 동적 저장기능을 갖는 지역 메모리와 캐쉬(cache)의 시간제한성하의 응용에의 부적합성

이러한 규칙들을 바탕으로, 일반적으로 다중프로세서로 구성된 실시간 architecture 설계 시에는 각 노드의 개별 프로세서들에 대해 예측 가능한 명령어 실행, 메모리 접근, 및 context 스위칭 등을 통해 외부와의 상호작용, interrupt 조절, 실시간 작업 실행시의 predictability 및 속도가 보장되어야 하고, 노드간의 통신이나 고장허용 기법들도 실시간 분산제어기의 predictability를 보장하기 위해 중요하게 다루어지는 문제들이다.

4.6 실시간 통신

실시간 응용에서는 통신의 값이 메시지가 수령자에게 성공적으로 도착하는 시간에 의존한다. 일반적으로 바람직한 전송시간은 데드라인을 야기시키는 특별한 최대 delay 또는 latency로 제한되어 있다. 이러한 delay 제한은 하나의 응용계층, end-to-end 시간 제약이다. 만약 메시지가 데드라인 내에 도착하지 못하면 end 응용에서는 그 값이 매우 감소되어있거나 몇몇 환경에서는 사라진 것으로 간주하기도 한다. 즉 데드라인을 넘어서는 delay가 일어나면 이러한 메시지는 응용에서 무시하고 잃어버린 것으로 간주한다. Hard 실시간 작업을 위한 데이터 지향 전송응용에서는 낮은 latency를 바탕으로 이러한 전송 손실이 전혀 없어야 하므로 네트웍 활용도보다 네트웍 전송지연의 결정적인 값으로의 예측이 선행되어야 한다. Ethernet와 같은 비실시간 데이터 전송에 많이 응용되어온 CSMA/CD 프로토콜의 일반적인 메시지 전송시간의 예측 불가능성을 극복하고자, 두 개의 real 및 virtual 클럭(clock)을 활용한 전역(global) FIFO 방법이 제안되었고[40], 메시지 도착 시간대신 Minimum Laxity First, Earliest Deadline First, 및 Shortest Length First 등과 같은 스케쥴링 기법을 활용한 메시지 stamping이 제안되기도 했다[41]. 이외에도, 버스 및 링과 같은 토플로지(topology)를 갖는 다중매체접근 네트웍에서의 동기적인 전송방법으로 응용되는 토큰 프로토콜을 전송 데드라인을 고려하여 timed-token 프로토콜로 변형한 연구[42]가 수행되었고, 이 결과는 임의의 데드라인을 허용하면서 지역최적화를 요구한 연구[43]나 보다 높은 링크 활용도를 보장하는 결과[44,45]로 확장되었다.

Point-to-point로 상호 연결되어 있는 네트웍을 가진 분산 시스템은 실시간 통신을 위한 좋은 후보이다. 왜냐하면 고장허용기법 뿐만 아니라 병렬 프로세싱과 통신은 다중프로세서와 모든 종류의 노드쌍 사이의 상호 연결 path를 통하여 달성될 수 있기 때문이다. 반면에 한 메시지가 소스와 목적지 사이를 이동하는데 거치는 각각의 노드/링크와 multihops에 임의적으로 도착하는 메시지들의 사이에 충돌이 있기 때문에 메시지를 적시에 전달할 수 있는 것을 보장하기는 어렵다. 이러한 어려움을 극복하고 데드라인 안에서 신뢰성 있게 packet 전송을 위해 실시간채널(real-time channel)이라는 새로운 전송모드가 [46]에서 제안되고 이를 활용한 실시간 전송을 위한 packet-switching 네트웍에서의 프로토콜에 대한 다수의 연구결과가 있다[10,47,48].

4.7 실시간 시스템의 fault-tolerance

고장허용은 시스템이 어떤 부품/요소가 고장(fault)난 상태에서도 주어진 기능 수행을 보장할 수 있는 능력으로 정의되는데, 진동, 습기, 온도, 전자장, 먼지, 및 화학약품 등의

열악한 환경 하에서도 연속적이고 신뢰성 있는 작업수행을 해야하는 실시간 시스템의 기본 특성을 감안할 때 이는 제어기 설계시 필수적인 사항이 된다. 시스템 신뢰도 향상을 위해 각 시스템의 이루고 있는 기본요소들의 신뢰성을 높이는 접근 방법(fault-avoidance)을 생각할 수 있으나 기술적, 경제적인 문제로 인한 한계가 있고 실제적으로는 효과적인 이상검출(detection/ diagnosis)을 통해 각 요소의 고장을 분리(isolation)하고 그 영향으로부터 신속하고 정확하게 회복(recovery)하는 접근방식(fault-tolerance)이 널리 쓰이고 있다[49]. 고장허용제어기의 설계는 시간 여분(time redundancy)과 공간 허용(spatial redundancy)의 tradeoff로 특성화되는데 기본적인 시간여분 방식으로는 retry, rollback, 그리고 restart 등이 응용되고 공간여분 방식으로는 기본단위(core) 이외의 H/W 및 S/W 등이 첨가되는 것이다. 고장허용 기법은 또한, 여분의 공간을 활용하는 방식의 차이에서 여분의 H/W 및 S/W 모듈을 항상 활용하는 static redundancy와 고장 검출 시에만 시스템 reconfiguration을 통해 활용하는 dynamic redundancy로 분류된다.

비실시간 응용 시는 시간이 저렴한 자원이기 때문에 공간 최적화에 중점을 두었으나 실시간 시스템인 경우에는 시간은 더 이상 공간의 절약을 위해 무시될 수 없는 귀중한 자원이므로 데드라인을 지키며 공간의 최적화를 위한 제어기의 설계는 매우 중요한 연구 과제중의 하나로 인식되어 왔다. [50]은 가장 단순하고도 일반적이 공간여분 방식인 삼단여분(Triple Modular Redundant, TMR)을 대상으로 이러한 구조의 시스템에 작업의 데드라인을 만족시키는 조건하에서 시간 여분을 이용해 최적의 혼용(hybrid)여분으로 설계하는 기법을 제안하였다. 또한 [51]에서 제안한 바와 같이 반복적인 작업/연산을 통해 그 결과가 개선되는 응용분야에서 데드라인을 지키기 위한 반응시간의 단축을 위해 결과의 정확도를 떨어뜨리는, 즉 imprecise computation 방식은 또다른 실시간 시스템에서의 고장허용기법이라 하겠다. 작은 데드라인을 지닌 엄격한 시스템인 경우 공간여분만을 활용하면 데드라인 조건을 항상 만족할 것 같아 보이나, 공간 여분의 응용 시에 redundancy를 관리하는 데에도 시간 오버헤드가 필요하므로 시간에 공간을 trade 시키는 데에도 잠정적인 한계가 있음을 명심해야 한다[52]. 이러한 방법들을 바탕으로 설계된 고장허용기법의 predictability를 위해, [53]에서는 시스템의 일시적 또는 영구적인 고장 상태가 발생했을 때 이의 회복을 위한 방법(recovery policy)이 다각도로 검토되고 이들의 각각 소요시간들을 유도하기 위해 연구가 수행되었다.

5. 결론 및 향후 연구과제

무수히 많은 응용 분야에서 디지털 컴퓨터의 사용 증가로 인해, 실시간 시스템에 대한 연구는, 플랜트의 동적 특성에 대한 정보 및 지식 습득을 위한 제어 공학 뿐만 아니라 최적의 제어 컴퓨터 설계 및 평가를 위한 컴퓨터 공학에 있어서의 중요한 학문 및 실용기술 개발 분야로 자리잡고 있다. 앞에서, 실시간 제어 및 연산의 새로운 분야에서 다양한 과제, 배경, 및 지금까지의 대표적인 연구 결과들이 소개되었다. 기본적인 연구 이론이 해석학적인 방법이나 컴퓨터 시뮬레이션을 통해 증명 또는 검증되어 왔으나 실제의 응용 분야에 대한 적용을 통한 검증된 것은 거의 없다. 이러한 면에서 실험용 testbed 또는 prototype 제작을 통해 내재해 있는 가정의 유효성을 입증하면서 개발된 기법/해법과 실제 상황과의 해법 사이의 차이를 줄여 나가고, 실시간 응용 분야 자체도 검증된 state-of-the-art 결과와 수집된 아이디어를 활용해서 리엔지니어링(reengineering)되어야 한다.

미래에 더욱 복잡해질 응용분야 및 제어 컴퓨터에 대한 더욱 향상된 고기능 및 고신뢰도의 요구 때문에 보다 심도 있게 다루어 질 몇 가지 연구 과제들을 다음과 같이 제시한다.

- formal specification과 verification
- 작업할당(task assignment) , 스케쥴링과 작업 및 message 데드라인
- 작업실행시간의 분석과 특성화(characterization)
- 고장허용과 보안성(security) 문제의 통합화
- 노드(단일 프로세서 집합) 및 시스템(다중 노드) level을 고려한 실시간 architecture
- 실시간 transaction/databases
- 초고속 네트웍에서의 실시간 통신 (생산시스템 자동화용, 멀티미디어 전송용, 분산다중병렬 컴퓨터용)
- 다양한 입출력(센서, 구동기, display 디바이스)의 통합화와 데드라인 만족을 위한 스케쥴링
- 실시간 인공지능(Artificial Intelligence, AI)기법

참 고 문 헌

[1] G. Brown and D. Campbell, "Instrument engineering: its growth and promise in process-control problems", no.72 vol.2, *Mechanical Engineering*, 1950

[2] J. Stankovic, "Misconceptions about real-time: A serious problem for next-generation systems", *IEEE Comput.*, pp.10-19, Oct. 1988

[3] K. Shin, C. Krishna, and Y. Lee, "A unified method fro evaluating real-time computer controller and its application," *IEEE Trans. on Automat. Contr.*, vol. AC-30, no.4, pp.357-366, Apr. 1985

[4] K. Hirai and Y. Satoh, "Stability of a system with variable time delay," *IEEE Trans. on Automat. Contr.*, pp.373-376, Apr.

1986

[5] K. Zahr and C. Slivinsky "Delay in multivariable computer controlled linear systems" *IEEE Trans. on Automat. Contr.*, vol. AC-19, no.8, pp.442-443, Aug. 1974

[6] K. Shin and X. Cui, "Effects of computing time delay on real-time control systems," *Proc. of 1988 ACC*, pp.1071-1076, 1988

[7] K. Shin and H. Kim, "Derivation and application of hard deadlines for real-time control systems," *IEEE Trans. on Systems, Manu. & Cybernetics*, vol.22, no.6, pp.1403-1413, Nov. 1992

[8] H. Kim and K. Shin, "On the maximum feedback delay in a linear/nonlinear control systems with input disturbances caused by controller computer failures," *IEEE Transactions on Control Systems Technology*, vol.2, no.2, pp.110-122, Jun. 1944

[9] E. Coffman, *Ed., Computer and job/shop scheduling theory*, New York: Wiley, 1976

[10] D. Ferrari and D. Verma, "A scheme for real-time channel establishment in wide-area networks", *IEEE J. Selected. Areas. Commun.*, vol.8, pp.368-379, Apr. 1990

[11] E. Lawler and C. Martel, "Scheduling periodically occurring tasks on multiple processors", *Informat. Proc. Lett.*, Feb. 1981

[12] J. Xu and L. Parnas, "Scheduling processes with release times, deadlines, precedence, and exclusion relations", *IEEE Trans. Software Eng.*, pp.360-369, Mar. 1990

[13] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment", *J. Amer. Compt. Mach.*, vol.20, no.1, pp.46-61, 1973

[14] C. Locke, "Software architecture for hard real-time applications: cyclic executives vs. fixed priority executives", *Real-Time Systems*, vo.4, no.1, Kluwer, Mar. 1992

[15] K. Ramamritham, J. Stankovic, and W. Zhao, "Distributed scheduling of tasks with deadlines and resource requirements", *IEEE Trans. Comput.*, vo.38, no.8, pp.1110-1123, Aug. 1989

[16] C. Locke, "Best-effort decision making for real-time scheduling", *Ph.D. dissert.*, CMU, Pittsburgh., PA, May. 1985

[17] S. Baruah, G. Koren and etc., "On the competitiveness of online real-time scheduling", in *Proc. Real-time Systems Symp.*, Dec. 1991

[18] K. Ramamritham and J. Stankovic, "Dynamic task scheduling in distributed hard real-time systems", *IEEE Trans. Software*, vo.1, no.3, pp.65-75, Jul. 1984

[19] L. Alger and J. Lala, "Real-time operating system for a nuclear power plant computer", in *Proc. Real-Time Syst. Symp.*, Dec. 1986

[20] V. Holmes, D. Harris, and etc., "Hawk: An operating system kernel for a real-time embedded multiprocessor", *Sandia Nat. Labs. Rep.*, 1987

[21] B. Furht, D. Grostick, and etc., Real-Time Unix Systems, *Norwell, MA: Kluwer*, 1991

[22] H. Tokuda, T. Nakajima, and P. Rao, "Real-time Mach: Towards a predictable real-time system", in *Proc. Usenix Mach Workshop*, Oct. 1990

[23] D. Kandlur, D. Kiskis, and K. Shin, "A real-time operating system for HARTS", in *Mission Critical Operating Systems*, A. Agrawala, K. Gordon, and P. Hwang, Eds. ISO Press, 1992

[24] J. Stankovic and K. Ramamritham, "The Spring kernel: A new paradigm for hard real-time operating systems", *IEEE Software*, vol.8, no.3, pp.62-72, May 1991

[25] O. Gudmundsson, D. Mose, and etc., MARUTI, An environment for hard real-time applications", in *Mission Critical Operating Systems*, A. Agrawala, etc., Eds. ISO Press, 1992

[26] H. Tikuda, and C. Mercer, "ARTS: A distributed real-time kernel", *ACM Operating Systems Rev.*, vol.23, no.3, Jul. 1989

[27] K. Schwan, A. Geith, and H. Zhou, "From $Chaos^{base}$ to $Chaos^{arc}$: A family of real-time kernels", *Proc. Real-Time Systems Symp.*, pp.82-91, Dec. 1990

[28] R. Scoy, J. Bamberger, and R. Firth, "An overview of DARK", in *Mission Critical Operating Systems*, A. Agrawala, K. Gordon, and P. Hwang, Eds. ISO Press, 1992

[29] A. Damm, J. Reisinger and etc., "The real-time operating system of Mars", *Operating Syst. Rev.*, pp.141-157, Jul. 1989

[30] L. Lamport and P. Melliar-Smith, "Synchronizing clocks in the presence of faults", *J. Assoc. Comput. Mach.*, pp.52-78, Jan. 1985

[31] D. Dolev and J. Halpern, "On the possibility and impossibility of achieving clock synchronization", in *Proc. Symp. on Theory of Computing*, pp. 504-511, Apr. 1984

[32] J. Lundelius-Welch and N. Lynch, "A new fault-tolerant algorithm for clock synchronization", *Informat. and Computat.*, vol.77, no.1, pp.1-36, 1988

[33] T. Srikanth and S. Toueg, "Optimal clock synchronization", *J. Assoc. Comput. Mach.*, vo.34, no.3, pp.626-645, Jul. 1987

[34] T. Smith, "Fault-tolerant clocking system", Proc. Fault-Tolerant Computing Symp., pp.262-264, 1981

[35] C. Krishna, K. Shin, and R. Butler", Ensuring fault-tolerance of phase-locked clocks", *IEEE Trans. Comput.*, vo. C 34, no.8, pp.752-756, Aug. 1985

[36] K. Shin and P. Ramanathan", Transmission delays in hardware clock synchronization", *IEEE Trans. Comput.*, vo.37, no.11, pp.1465-1467, Nov. 1988

[37] P. Ramanathan, D. Kandlur and K. Shin, "Hardware assisted software clock synchronization for homogeneous distributed systems", *IEEE Trans. Comput.*, vo.39, no.4, pp.514-524, Apr. 1990

[38] F. Cristian, "Probabilistic clock synchronization", *Tech. Rep. RJ 6462 (62550)*, IBM Almaden Res. Center, Sep. 1988

[39] K. Shin, "HARTS: A distributed real-time architecture", *IEEE Computer*, vol.24, no.5, pp.25-36, May 1991

[40] M. Molle and L. Kleinrock, "Virtual time CSMA: Why two clocks are better than one", *IEEE Trans. Commun.*, vol. COM-33, no.9, pp.919-933, Sep. 1985

[41] W. Zhao and K. Ramamritham, "Virtual time CSMA protocols for hard real-time communications", *IEEE Trans. Software Eng.*, vol. SE-13, no.8, pp.938-952, Aug. 1987

[42] G. Agrawal, B. Chen, and etc., "Guaranteeing synchronous message deadlines with the timed token protocol", in *Proc. Distributed Computing Systems*, pp.468-475, Jun. 1992

[43] Q. Zheng and K. Shin, "Synchronous bandwidth allocation in FDDI networks", *Proc. ACM Multimedia'93*, pp.31-38, 1993

[44] G. Agrawal, B. Chen, and etc., "Optimal synchronous capacity allocation for hard real-time communication with the timed-token

rpotocol", in *Proc. Real-Time Systems Symp.*, pp.198-207, Dec. 1992

[45] M. Hamdaoui and P. Ramanathan, "Selection of timed token protocol parameters to guarantee message deadlines", *Tech. Rep. ECE-92-10*, Univ. of Wisconsin, Nov. 1992

[46] D. Ferrari, "Cliant requirements for real-time communication services", *Tech. Rep.: TR-90-007*, International Computer Science Institute, Berkeley, Mar. 1990

[47] D. Anderson, "A software architecture for network communication", in Proc. Distributed Computing Systems, pp.376-383, Jun. 1988

[48] D. Kandlur and K. Shin, "Traffic routing for multicomputer networks with virtual cut-through", IEEE Trans. Comput., vo.41, no.10, pp.1257-1270, Oct. 1992

[49] D. Siewiorek and R. Swarz, *The Theory and Practice of Reliable System Design*, Digital Press, 1982

[50] 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--1225, Nov. 1996

[51] J. Liu, K, Lin, and etc., "Algorithms for scheduling imprecise computations", *Computer*, vol.24, pp.58-69, May 1991

[52] C. M. Krishna, K. Shin, and R. Butler, "Synchronization and fault-masking in redundant real-time systems", in *Dig. Papers, FTCS-14*, pp.1752-157, Jun. 1984

[53] H. Kim and K. Shin, "Evaluation of fault-tolerance latency from real-time applications perspectives," *Tech. Rep. CSE-TR-201-94*, CSE Division, EECS Dept., The Univ of Michigan, 1994

저 자 소 개

김학배(金鶴培)

1965년 10월 10일생. 1988년 서울대 공대 전자공학과 졸업. 1990년 미국 미시간대 대학원 전기공학과(EECS) 졸업(석사). 1994년 8월 동 대학원 졸업(공박). 1994년 9월~1996년 8월 미국 National Research Council(NRC) Research Associate at NASA Langley Research Center. 1996년 9월~현재 연세대학교 전기공학과 조교수. 관심분야 실시간제어, 자동화공학, 고장허용기법 및 신뢰도 평가.