論文 94-19-7-11

실시간 시스템에서 여러 부정확한 스케쥴링 기법하에서의 부정확한 확률에 관한 비교분석

正會員 安 貴 任* 正會員 高 健**

Comparative Analysis on Imprecision Probability Under Several Imprecise Scheduling Schemes in Real Time Systems

Gwi Im Ahn*, Kern Koh** *Regular Members*

要 約

실시간 시스템에는 정확한 계산 기법과 부정확한 계산 기법이 있다. 부정확한 계산 기법은 실시간 시스템에서 스케쥴링의 유연성을 제공하는 수단이다. 지금까지 큐잉이론을 이용한 부정확한 스케쥴링에 관한 연구들은 태스크들의 평균 결과의 질과 평균 대기시간을 절충할 때의 비용과 장점들을 명확하게 수량화하는 것들이었다. 이 논문은 두개의 부정확한 스케쥴링 방법을 사용할때, 어떤 태스크가 부정확한 계산이 될 부정확한 확률들을 구하였으며 또한, 이런 두 방법의 부정확한 확률들의 여러 단조형 부정확한 시스템 매개변수들에 대한 의존성을 비교 분석하였다.

ABSTRACT

There are two computation techniques in real time systems : precise and imprecise computation. The imprecise computation technique is a means to provide scheduling flexibility in real time systems. The studies on imprecise scheduling using queueing theoretical formulation up to date are to explicitly quantify the costs and benifits in trade-off between the average result quality and the average waiting time of tasks. This paper uses two imprecise scheduling schemes and solves the imprecision probability, the probability of any task being imprecise under two imprecise scheduling schemes and analyzes the dependence of the imprecision probability on several parameters of the monotone imprecise system.

*東義大學校 電算統計學科
Dept. of Computer Science and Statistics Dongeui University
**서울大學校 電算統計學科
Dept. of Computer Science and Statistics Seoul National University

論文番號 : 9481
接受日字 : 1994年 3月 14日

I. 서 론

경성 실시간 시스템에서 태스크는 종료시한(dead-

line)이라는 시간 제약조건을 가진다. 즉, 모든 경성 실시간 시스템의 태스크들은 자신들의 종료시한 내에 실행이 종료(completion)되어야 만 하는데 만일 그렇게 되지 못했다면 시간 결함이 발생했다고 하며 시간 결함이 발생한 태스크에 의해 만들어진 그때까지의 중간 결과는 거의 쓸모가 없어진다[1, 3, 4, 12]. 불행히도 동적 알고리듬의 다양한 실행시간같은 요인들은 모든 경성 실시간 태스크들이 자신들의 종료시한을 만족시켜 정확한 결과를 만드는데 많은 어려움을 야기시킨다[1]. 또한 시스템의 고장이나 과부하들도 경성 실시간 태스크들의 시간 결함을 야기시키는 요인으로 생각할 수 있다.

시간 제약 조건을 갖는 실시간 시스템의 스케쥴링 유연성을 위한 수단으로 최근에 부정확한 계산 기법[13, 14]이 제안되고 있는데 이러한 기법에 기반을 둔 시스템을 부정확한 시스템(imprecise system)이라고 부른다. 부정확한 시스템은 위에서 언급한 여러요인들로 인해 정확한 결과(precise result)를 종료시한 내에 얻을 수 없을 때 수용가능한 대략적인 결과를 사용자에게 제공함으로써 시간 결함을 예방하고 점진적인 신뢰성감소를 이루어 준다. 이러한 부정확한 시스템의 응용예로서 화상 처리와 레이더 추적등을 생각할 수 있는데 이러한 응용예들은 너무 늦게 즉 종료시한 후에 만들어진 완벽한 상이나 정확한 목표물 위치보다는 오히려 종료시한 내에 만들어진 퍼지 상(fuzzy image)의 프레임이나 대략적인 목표물 위치가 더 좋을 수도 있는 예들이다. 따라서, 부정확한 계산 기법은 항상 모든 태스크들의 종료시한을 만족시켜야 하는 실시간 시스템의 어려움을 어느 정도 완화시켜 준다고 할 수 있다.

시간 결함으로 인한 나쁜 결과를 최소화시키기 위한 기본 전략으로 모든 태스크들을 동등하게 취급하지 않고 중요한 태스크들과 덜 중요한 태스크들로 구별하여 전자를 강제적인 태스크들, 후자를 선택적인 태스크들이라고 한다. 강제적인 태스크들은 종료시한 전까지 반드시 스케쥴되어 실행되어야 하는데 반해 선택적인 태스크들은 만일 과부하 상태가 발생한다면 일시적인 과부하동안 미 완성 시킨 채로 실행을 종결(termination)시킬 수도 있다. 이때 태스크들은 개별적인 작업단위 혹은 통신단위를 의미하기로 한다.

부정확한 계산 기법에서는 위에서 언급한 기본 전략을 사용하며 조금 더 나아가 프로그래머들로 하여금 각 태스크들을 두개의 서브태스크들 즉, 한개의 강제적인 서브태스크와 한개의 선택적인 서브태스크로 논리적인 분해가 되도록 구성하게 한다[1, 2, 12]. 강제적인 서브태스크는 수용가능한 정확도의 결과를 출력하기 위한 그 태스크의 부분으로 시스템은 고장이 없는 한 강제적인 서브태스크를 종료시한 전까지 꼭 종료시켜야 만 한다. 선택적인 서브태스크는 결과의 질(result quality)을 향상시키는 그 태스크의 부분으로 만들어진 결과에 오차가 생기더라도 시스템은 필요하다면 빠른 응답시간을 위해 선택적인 서브태스크를 혹은 그 일부분을 완성시키지 않고 중간에 종결시킬 수도 있다. 부정확한 계산 기법은 최소한 강제적인 서브태스크만 종료시한을 만족시켜 주면 그 태스크의 결과는 유용하다고 본다.

태스크들이 단조형(monotone)으로 설계되었다면 스케쥴링할 때 최대의 유연성을 갖는다고 한다. 태스크가 중간에 종결되어 만들어진 중간 결과의 질이 이후로 더 오랫동안 실행되더라도 감소하지 않는 태스크들을 단조형이라고 일컫는다. 단조형 태스크가 종료될 때 만들어진 결과는 정확(precise)하다고 하며 즉, 태스크에게 오차가 전혀 없다고 한다. 그런데 태스크가 종료되기 전에 종결되었다면 종결 때까지 만들어진 중간 결과는 그때까지 만들어진 다른 모든 중간 결과 중에서 가장 좋다고 한다. 이러한 결과는 그 태스크의 강제적인 서브태스크가 종료되는 한 유용하다고 보며 그러한 결과를 부정확한 결과 혹은 근사결과라고 한다. 다시 말해 단조형 태스크가 종결되기 전에 더 오랫동안 수행되면 될수록 부정확한 결과에서 오차가 더욱 더 작아 진다고 말할 수 있다. 이 논문에서는 시스템을 이러한 단조형 태스크들을 수용하는 부정확한 시스템으로 가정하고자 한다[1, 3, 4].

부정확한 계산 기법의 출현 이후로 부정확한 스케쥴링 문제에 많은 진전이 이루어졌다. 부정확한 계산을 위해 여러 가지 계산 모델이 개발되어 왔으며, 단일프로세서에서 부정확한 일(job)들에 대한 부정확한 스케쥴링 알고리듬과, 다중프로세서에서 많은 버전을 가지고 있는 부정확한 일들과 병렬가능하고 멀티태스킹이 가능한 부정확한 계산들을 위한 많은 부정확한 스케쥴링 알고리듬들이 개발되어 왔다[3, 16, 17]. 이 논문은 지금까지 진행되어 온 부정확한 스케쥴링 문제들을 간단히 나열하고 있다. 그런데 이러한 문제들은 NP-hard라는 심각한 문제를 야기시키며 거의 모든 문제들이 최적의 스케쥴을 구하는 결정적인 공식을 요구하는 것들이다. 그래서 이 논문에서는 위 문제들에서 사용한 결정적인 공식보다는 오히려 더 융통성이 있는 큐잉이론에 입각해서 어떤 태스크

가 과부하로 인해 부정확한 계산을 하고 시스템을 빠져 나가게 될 확률 즉, 부정확한 확률을 구해보려고 한다. 이 논문은 두개의 부정확한 스케쥴링 방법하에서 부정확한 확률을 구하고 이 확률의 여러 단조형 부정확한 시스템 매개변수들에 대한 의존성을 비교분석하려고 한다.

이 논문의 Ⅱ장에서는 부정확한 시스템의 응용영역과 필요한 이유를 다루고, Ⅲ장에서는 기본적인 부정확한 계산 모델과 관련 연구를, Ⅳ장에서는 부정확한 확률 계산 모델을, Ⅴ장에서는 FCFS방법에서의 부정확한 확률 계산을 Ⅵ장에서는 LCFS방법에서의 부정확한 확률 계산을, Ⅶ장에서는 성능분석을 다루고 있다.

Ⅱ. 부정확한 시스템의 응용영역과 필요한 이유

부정확한 계산기법은 결함 허용성과 점진적인 신뢰성 감소를 위한 자연스런 방법으로 사용될 수 있다. 결함 발생시에 부정확하지만 유용한 결과를 사용하는 것은 여러 응용영역에서 필요한 방법이다. 아래에 부정확한 계산기법의 여러가지 응용영역과 필요한 이유를 구체적으로 나열해 보려고 한다.

1. 결함 허용성과 가용성(availability) 향상

한 예로써 레이다 추적은 일시적인 결함으로 인해 추적 계산을 중간에 종결시켜 부정확한 결과를 만들어 낸다. 이러한 경우 정확한 계산기법에서는 복구 연산이 필요하지만, 부정확한 계산기법에서는 중간에 종결한 부정확한 결과가 시스템으로 하여금 복귀불 추적을 가능하게만 한다면 복구 연산이 전혀 필요하지 않다. 또 한가지 예로서, 제어법계산(control law computation)에 의해 만들어진 부정확한 결과가 제어된 시스템(controlled system)을 안정적으로 만드는데 충분히 정확하다면 계산을 중간에 종결시켰던 결함(fault)은 허용된다. 이러한 시간 제약적인 내장응용(embedded application)들은 오차 복구작업을 할 시간이 없고 시스템 자원의 제한으로 인해 완전히 복제된 데이타 저장이나 태스크 수행을 허용하지 않기 때문에 부정확한 결과의 사용은 가용성을 증가시키고 결함 허용성을 제공하기 위한 효율적인 방법이라고 할 수 있다.

2. 복원성(resilience)의 향상

일반적으로 부정확한 계산기법은 분할되어 수행될 수 없는 원산(atomic operation)의 복원성을 증가시키기 위해 사용될 수 있다. 예를 들면, 버블 정렬 서버에서 이 서버를 이용하는 고객이 부분적으로 정렬된 데이타를 사용 가능하다고 한다면 부정확한 계산기법은 훨씬 더 복원성을 향상시킬 수 있다.

3. 복구(recovery)의 새로운 차원 제공

부정확한 시스템에서 복구작업은 부정확한 중간결과를 수용 불가능할 때만 필요하기 때문에 부정확한 계산기법은 전통적인 복구 방법에 새로운 차원을 제공한다. 중간결과를 수용할 수 없어서 복구 작업이 필요할 때 부정확한 계산기법은 태스크 수행도중 만들어진 이전의 여러 부정확한 계산 결과들로 부터 최종결과를 추정하는 전향복구 방법이다. 강제적인 서브태스크 수행 후 태스크에 의해 만들어진 중간 결과(정확도 표시 변수의 값)를 검사점으로 이용하는 후향복구를 사용한다. 경성 실시간 시스템과 복구의 비용이 매우 높은 분산시스템에서도 이러한 장점은 중요하다.

Ⅲ. 기본적인 부정확한 계산 모델과 관련 연구

Ⅲ-1. 기본적인 부정확한 계산 모델

모든 부정확한 계산 모델은 아래와 같은 기본적인 모델의 확장되고 변형된 유형이라고 생각하면 되는데 여러한 기본 모델은 선점가능한(preemptable) 태스크 집합 $T = \{T_1, T_2, \ldots, T_n\}$를 가지며 각 태스크 T_i는

(a) T_i의 수행시작이 가능한 시작가능시간 : r_i
(b) T_i의 종료시한 : d_i
(c) T_i의 처리시간 : τ_i
(d) T_i의 상대적인 중요성인 가중치(weight) : ω_i

등의 매개변수들을 갖는다. 각 태스크 T_i를 논리적으로 두개의 서브태스크인 강제적인 서브태스크 M_i와 선택적인 서브태스크 O_i로 나누며 m_i와 o_i를 각각 M_i와 O_i의 처리시간을 나타내기로 할 때 $\tau_i = m_i + o_i$이다. M_i와 O_i의 시작가능시간과 종료시한은 T_i의 것과 같고 선점가능한 태스크 집합 T는 M과 O를 아래와 같이 놓을 때 $T = M \cup O$라고 할 수 있다.

$$M = \{M_i \mid i = 1, \ldots, n\}$$

$$O = \{O_i \mid i = 1, \ldots, n\}.$$

단일프로세서 시스템에서의 스케쥴(schedule)은 서

로 겹치지 않는 시간간격으로 T의 태스크들에게 프로세서를 할당하는 것이다. 그러한 시간간격으로 프로세서를 한 태스크에게 할당한다면 이 태스크는 스케쥴 된다고 한다. 어떤 유효한 스케쥴(valid schedule)에서 프로세서는 한순간에 한 태스크에게만 할당되고 모든 태스크 T_i는 시작가능시간 이후에 스케쥴되어져야 한다. 게다가 프로세서가 태스크 T_i에게 할당되는 총 프로세서시간은 m_i 보다는 크거나 같고 τ_i 보다는 작거나 같아야 된다. 한 태스크에게 할당된 총 프로세서시간이 시간t에서 그 태스크의 처리시간과 같을 때를 일반적인 의미에서 시간t에서 한 태스크가 종료된다고 한다. 강제적인 서브태스크의 처리가 끝났을 때를 강제적인 서브태스크가 종료되었다고 말한다. 선택적인 서브태스크는 강제적인 서브태스크에 의존적이며 강제적인 서브태스크가 종료될 때 시작가능시간이 되어서 비록 종료가 되지 않을지라도 중간에 종결할 수가 있다. 종결된 이후로 선택적인 서브태스크에게는 더 이상의 프로세서가 할당되지 않는다. 그것의 강제적인 서브태스크가 종료했다면 T_i는 스케쥴에서 종료되었다고 하며 그것의 선택적인 서브태스크가 종료되었다면 T_i가 종결되었다고 한다. 일반적인 경성실시간 모델은 $o_i=0$(for all i)이고 연성 실시간 모델은 $m_i=0$(for all i)이다.

모든 태스크들이 종료시한을 지켜서 종료되는 유효한 스케쥴을 적정한 스케쥴(feasible schedule)이라 하며 적어도 하나의 적정한 스케쥴이 존재하면 집합 T는 스케쥴 가능하다고 한다[1]. 한 스케쥴에서 선택적인 서브태스크 O_i에게 할당되는 프로세서시간인 σ_i가 o_i와 같을 때 태스크 T_i(혹은 O_i)는 정확하게 스케쥴 되었다고 하며 이때 태스크 T_i의 오차 ε_i는 0(zero)라고 한다. 모든 태스크가 정확하게 스케쥴되는 스케쥴이 일반적인 의미에서 유효한 스케쥴이다. 만약 $(\sigma_i < o_i)$가 성립하면 T_i는 O_i의 $(\sigma_i - o_i)$만큼이 실행되지 않아서 부정확하게 스케쥴 되었다고 하며 이때 T_i의 오차 ε_i는 T_i의 부정확한 결과의 오차를 나타낸다.

Ⅲ-2. 관련연구

부정확한 스케쥴링 문제들을 나열해 보면 아래와 같다.

(1) 총 오차를 최소화 시키기 위한 문제

(2) 최대 오차 혹은 평균 오차의 최소화 문제

(3) 실행되지 않는 선택적인 태스크들 수를 최소화 시키기 위한 문제

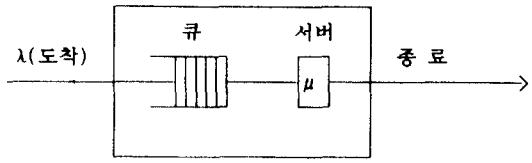
(4) 종료시한을 넘겨 끝나는 태스크들 수를 최소화 시키기 위한 문제

(5) 평균 반응 시간의 최소화 문제

지금까지 위 문제들에 대한 연구는 최적의 스케쥴(optimal schedule)을 구하기 위한 결정적인 공식들이었으며 또한, 최적의 스케쥴을 구하는 문제들은 대부분 NP-hard이다. 따라서, 이 논문은 결정적인 공식보다 훨씬 더 융통성이 있는 큐잉이론 공식을 사용하는 문제를 다루려고 한다.

큐잉이론 공식 관련 연구로서 두단계 스케쥴링이 있는데 두단계 스케쥴링 모델은 각 태스크에게 1차버전(primary version)과 2차버전(alternate version)을 제공하여 구현된다. 1차버전은 그 태스크의 강제적인 서브태스크와 선택적인 서브태스크를 합한 버전으로 정확한 결과를 만들어 내는 대신 처리시간이 길고 2차버전은 그 태스크의 강제적인 서브태스크만을 포함하는 버전으로 부정확한 결과를 산출해 내는 대신 빠른 처리시간을 제공해 준다. 각 태스크는 자신의 1차버전이나 혹은 2차버전이 스케쥴되고 실행된다. 각 태스크는 자신의 1차버전이 스케쥴되고 실행될 때는 완전단계(full level)에서 서비스를 받으며 자신의 2차버전이 스케쥴되고 실행될 때는 축소된 단계(reduced level)에서 서비스를 받는다. 두단계 스케쥴링은 시스템이 과부하가 아니고 반응시간이 짧을때는 정확한 결과를 산출해 내기 위해 태스크를 완전 단계에서 처리하여 스케쥴의 오차를 줄이고 시스템이 과부하일 때는 반응시간을 줄이기 위해 태스크들을 축소된 단계에서 처리하는 방법을 선택한다. 지금까지 이러한 모델에 행해진 연구들은 적당한 스케쥴링 방법하에서 태스크들의 평균 결과의 질과 평균 대기시간을 절충(trade-off)하는 것들이었다. 기존의 방법에서 평균 결과의 질과 평균 대기시간을 나타내는 비용으로 대기시간(waiting time)과 완전단계에서 실행되는 태스크들의 수로 나타내는 연구가 주종이었다. 반면, 이 논문에서는 평균 결과의 질을 나타내는 비용함수로서 사용하기 위하여 특정 태스크가 시스템에 도착한후 부정확한 계산이 될 부정확한 확률을 구하였다. 또한 지금까지의 연구에서 태스크를 하나의 강제적인 서브태스크와 하나의 선택적인 서브태스크로 분해했을 때의 스케쥴링이었는데 이 논문에서는 좀 더 나아가 모든 태스크들을 한개의 강제적인 서브태스크와 K개의 선택적인 서브태스크들로 논리적 분해하는 다른 단조형 부정확한 시스템을 사용한다. 따라서 이 논문은 두개의 부정확한 스케쥴링

방법하에서의 부정확한 확률을 구하고 이 확률들의 여러 매개변수들에 대한 의존성을 비교 분석하려고 한다. 이 논문에서 구한 태스크들의 부정확한 확률은 평균 대기시간과 평균 결과의 질을 절충하는 문제에서 평균 결과의 질을 나타내는 비용함수로서 부정확한 확률을 사용할 수 있다.



(그림 1) 시스템 모델

Ⅳ. 부정확한 확률 계산 모델

Ⅳ-1. 기본 가정

앞으로 구체적인 진행을 위해서 대상 시스템을 다음과 같이 가정한다.

(1) 태스크들의 처시간은 같으며 단조형(monotone)이다.

(2) 태스크들은 아래에 나타나 있는 (그림 1)의 시스템 모델처럼 하나의 공통 준비 큐(ready queue)에 평균 λ의 율로 도달하는 포아송 프로세스(poisson process)들이며 평균 μ의 율로 서버에 의해 처리된다.

(3) 각 태스크 T_i를 한개의 강제적인 서브태스크 M_i과 K개의 선택적인 서브태스크들인 O_{i1}, O_{i2}, ..., O_{iK}들로 분해한다. 또한 m_i와 o_{i1}, o_{i2}, ..., o_{iK}를 각각 M_i와 O_{i1}, O_{i2}, ..., O_{iK}들의 처리시간이라고 할 때 모든 서브태스크들의 처리시간이 같다고 가정하였으므로 $m_i = o_{i1} = \cdots = o_{iK} = 1/((K+1)\mu)$라고 둔다.

(4) 각 태스크가 갖는 한개의 강제적인 서브태스크 M_i, K개의 선택적인 서브태스크들 O_{i1}, O_{i2}, ..., O_{iK}들은 M_i, O_{i1}, O_{i2}, ..., O_{iK}순서로 실행된다고 가정한다.

(5) m_i와 o_{ij}($j=1, 2, \ldots, K$)들은 확률적으로 독립적이며 지수분포의 임의 변수들이다.

(6) 현재 서버에서 실행 중이던 태스크는 정확한 계산 혹은 부정확한 계산 중 하나를 하게 된다.

(7) 정확한 계산을 하게되는 태스크는 자신의 강제적인 서브태스크와 K개의 선택적인 서브태스크들을 다 실행시키고, 부정확한 계산을 하게 되는 태스크는 자신의 강제적인 서브태스크는 항상 실행시키고 K개의 선택적인 서브태스크들 중 $X(1 \le X \le K)$개의 선택적인 서브태스크는 실행시키지 않는다.

(8) 모든 태스크의 강제적인 서브태스크는 항상 실행된다고 가정한다.

Ⅳ-2. 스케줄링 방법

(1) 태스크들은 FCFS(first-come-first-service)방법과 LCFS(last-come-first-service) 방법으로 처리된다.

(2) 태스크가 도달한 직후 시스템안의 태스크들 수(큐안의 모든 태스크들+현재 서버의 태스크)를 계산하여 M이하인가 M을 초과하는가를 판정한다.

(3) 시스템안의 전체 태스크들 수가 M이하이면 현재 서버에서 실행 중이던 태스크는 정확한 계산을 하고 M을 초과하면 현재 서버에서 실행 중이던 태스크는 현재 실행 중이던 선택적인 서브태스크까지만 실행시키는 부정확한 계산을 한다.

(4) 정확한 계산을 하는 태스크는 자신의 강제적인 태스크 M_i와 K개의 선택적인 서브태스크들인 O_{i1}, O_{i2}, ..., O_{iK}들을 실행순서대로 모두 다 실행시킨다.

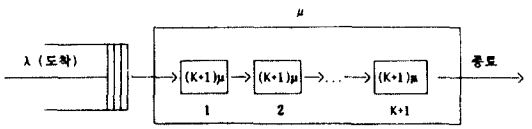
(5) 부정확한 계산을 하는 태스크는 시스템안의 전체 태스크 갯수 판명직후 강제적인 서브태스크 M_i를 실행 중이었으면 이 서브태스크까지만 수행시키고 모든 선택적인 서브태스크는 실행시키지 않고, 만약 강제적인 서브태스크를 수행시키고 나서 선택적인 서브태스크들 중 하나를 순서에 맞추어 실행 중이었으면 그 선택적인 서브태스크까지만 수행시키고 나머지 선택적인 서브태스크는 버리고 시스템을 빠져나오는 계산을 한다. 따라서 부정확한 계산은 자신의 선택적인 서브태스크들 중 X개를 실행시키지 않으며 이 X의 값은 1, 2, ..., K 중 하나가 될 수 있다.

이 논문에서 서버에서의 정확한 계산은 확률 δ_0로 발생하고 부정확한 계산에서 $X(1 \le X \le K)$개의 선택적인 서브태스크들은 실행시키지 않을 확률은 δ_x로 발생한다고 가정하며 한 태스크의 전체 서브태스크들(강제적인 서브태스크 +K개의 선택적인 서브태

스크들)이 전부 다 실행이 안되는 경우는 발생하지 않는다고 가정한다.

Ⅳ-3. 서브태스크들의 모델

각 태스크 T_i는 한개의 강제적인 서브태스크와 K개의 선택적인 서브태스크들로 분해되므로 각 서브태스크들을 모델화할 필요가 있다. 이를 위해 앞의 시스템 모델에서 보았던 서버를 좀더 구체적으로 생각해 보아야 한다. 앞의 서버의 평균 μ의 율로 태스크들이 처리되는 하나의 큰 처리시설(service facility)이라고 생각하고 각 태스크의(K+1)개의 서브태스크들을 위해서는 앞의 서버를 (K+1)개의 스테이지(stage)들을 갖는 Erlangian 서버[10]로 다시 모델화 시켜서 (그림 2)에 나타내었다. 태스크 T_i의 서브태스크들인 M_i, O_{i1}, O_{i2}, ..., O_{iK}들의 실행순서는 M_i, O_{i1}, O_{i2}, ..., O_{iK}라고 가정하고 처리시설안의 (K+1)개의 스테이지들 중 스테이지 1은 강제적인 서브태스크 M_i에게 스테이지 2, 스테이지 3, ..., 스테이지 (K+1)은 각각 선택적인 서브태스크들 O_{i1}, O_{i2}, ..., O_{iK}에게 일대일 대응시킨다. 아래의 모델은 $M/E_{K+1}/1/M$으로 생각할 수 있으며 각 스테이지들은 각각 평균 서비스 율인 $((K+1)\mu)=(1/m_i)=(1/o_{i1})=, \ldots, =(1/o_{iK})$를 가진다.



(그림 2) (K+1)-stage의 Erlangian서버 E_{K+1}

Ⅳ-4. 서버의 태스크가 정확한 계산과 부정확한 계산이 될 확률

각 태스크의 시스템 도착 직후 총 태스크 수가 M이하인 경우와 M을 초과하는 경우를 구별하여 현재 서버에서 처리되고 있는 태스크를 주시할 필요가 있다. 시스템은 새로운 태스크가 도착 하자마자 현재 시스템안의 총 태스크 수(queue안의 task 수+현재 처리되고 있는 태스크)를 계산하여 그 수가 시스템 매개변수 M을 초과하는지의 여부를 판단한다. M이하일 경우에는 현재 서버의 태스크는 정확한 계산을 하게 되므로 자신의 (K+1)개의 서브태스크들을 다 실행시킨다. 그러나 M을 초과 했을 경우에는 빠른 응답시간을 위해 현재 서버에서 처리되고 있는 태스크의 총 K개의 서브태스크들 중 X(X=1, 2, ..., K)개를 실행시키지 않게 되어, 현재 서버의 태스크는 K개의 스테이지들을 위한 스테이지들 중 뒤에서부터 X개의 스테이지들을 거치지 않고 시스템을 떠나게 된다. 따라서, 현재 서버의 태스크는 총 태스크 수가 M이하일 경우에는 평균 μ의율로 정확한 계산이 되어서 시스템을 떠나게 되고 M을 초과했을 경우에는 평균 $((K+1)\mu)/(K+1-X)$의 율로 부정확한 계산이 되어서 시스템을 떠나게 된다. 모든 태스크는 항상 자신의 강제적인 서브태스크는 실행시켜야 하므로 처리시설의 스테이지 1은 무조건 거쳐서 실행된다. 스테이지 2부터는 선택적인 서브태스크들이 시작되는 스테이지들이며 이때 정확한 계산이 될 때는 스테이지 2, 3, ..., K+1을 다 거쳐가며 모든 자신의 서브태스크들을 다 실행하며, 부정확한 계산이 될 때는 모든 선택적인 서브태스크들 중 X개의 선택적인 서브태스크들을 실행시키지 않고 처리시설을 빠져나가게 된다. 이때 X의 값은 1, 2, ..., K값 중 하나이다. 즉 부정확한 계산을 하고 처리시설을 빠져나가는 태스크들은 일률적으로 선택적인 서브태스크들의 수행을 하지 않고 처리시설은 빠져나가는게 아니고 1개의 선택적인 서브태스크들을 수행시키지 않고 빠져나가는 경우, 2개를 수행시키지 않고 빠져나가는 경우, 3개 혹은 4개, ..., K개의 선택적인 서브태스크들을 수행시키지 않고 빠져 나가는 경우를 각각 생각할 수 있다. 1개의 선택적인 서브태스크들을 실행시키지 않고 나갈 확률을 δ_1, 2개의 선택적인 서브태스크를 실행시키지 않고 나갈 확률을 δ_2, 계속하여 K개의 선택적인 서브태스크들을 실행시키지 않을 확률을 δ_K 라고 할 때 이들의 값은 $\delta_0+\delta_2+\delta_3+, \ldots, +\delta_K=1$가 된다고 할 수 있다.

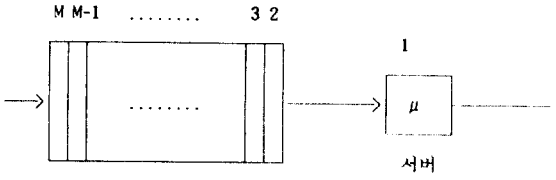
Ⅴ. FCFS방법에서의 부정확한 확률 계산

Ⅴ-1. 상태(state) 정의

우리가 해결해야 할 문제는 어떤 특정 태스크가 시스템에 도착하여 부정확한 계산이 될 확률이므로 어떤 특정 태스크가 시스템에 도착했을 때부터 시작해서 이 태스크가 정확한 계산 혹은 부정확한 계산이 되어 시스템을 떠날 때까지 시스템안에서 이동하는 위치를 추적할 필요가 있다. 그래서 우리 시스템을 (그림 3)처럼 서버를 1번으로, 큐의 맨 앞부분을 2번으로, 이런 식으로 계속하여, 큐의 맨 마지막을 M번

으로 정한다.

특정 태스크의 부정확한 확률 계산에는 아무래도 이 태스크의 시스템안에서의 위치와 시스템 전체의 태스크 수를 생각해 보아야 하므로 2차원의 상태 즉, 상태 (i, j)를 정의한다. 이때 i는 특정 태스크의 시스템안에서의 위치이고, j는 위치 i의 특정 태스크 뒤에 남아 있는 다른 태스크들의 갯수를 나타내어 i와 j를 합하면 시스템의 전체 태스크들의 수를 나타낼 수 있다.



(그림 3) 위치(position) 번호

V-2. 상태(state)들 간의 전이(transition)

V-2-1. 도착사건과 서비스사건

시스템에는 도착사건과 서비스사건이 발생할 수 있다. 어떤 특정 태스크가 시스템에 도착한 후 도착사건과 서비스사건이 발생할 때마다 이동하는 위치를 추적해 볼 필요가 있다. 이러한 내용을 아래에 구체적으로 표현한다.

CASE 1 : FCFS의 도착 사건의 발생

CASE 1.1 : 전체 태스크 수가 M이하일 때

- 도착한 태스크 : 현재 시스템 위치 맨 끝에 위치.
- 다른 태스크들 : 위치 변화 없음.

CASE 1.2 : 전체 태스크 수가 M을 초과 할 때

- 위치 1번의 태스크 : 부정확한 계산이 되어 K개 중 X(X≤K)개의 선택적인 서브태스크들을 실행하지 않고 시스템을 떠나다.
- 위치 2, 3, ..., M-1, M 번의 태스크들 : 위치 1, 2, ..., M-2, M-1으로 각각 변화.
- 도착한 태스크 : 위치 M번에 위치.

CASE 2 : FCFS의 서비스 사건의 발생

- 위치 1번의 태스크 : 서비스(정확한 혹은 부정확한 서비스)되어 시스템을 나간다.
- 위치 2, 3, ..., M 번의 태스크들 : 위치 1, 2, ..., M-1으로 변화.

V-2-2. 상태들간의 전이

상태 (i, j)에서 발생되는 전이를 생각해 보자.

CASE 1 : FCFS의 도착 사건 발생 시의 전이

CASE 1.1 : 전체 태스크 수가 M이하일 때

상태(i, j) → 상태(i, j+1)

CASE 1.2 : 전체 태스크 수가 M을 초과할 때

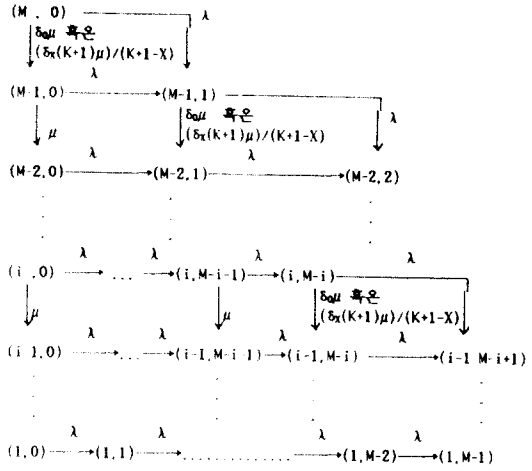
상태(i, j) → 상태(i-1, j+1)

CASE 2 : FCFS의 서비스 사건(정확한 사건 혹은 부정확한 사건) 발생 시의 전이

상태(i, j) → 상태(i-1, j)

V-3. 상태-전이-율 다이어그램(state-transition-rate diagram)

시스템의 상태(i, 0)이 되었다고 가정하고 상태-전이 율 다이어그램을 아래 (그림 4)에 나타낸다. 그림의 간단한 표현을 위하여 현재 서버의 태스크가 X개의 선택적인 서브태스크들을 버리게 되어 부정확한 계산을 하게 될 때의 확률을 δ_X라고 정의하며, 이때 X의 값은 1, 2, ..., K중의 하나가 될 수 있다.



(그림 4) FCFS의 상태-전이 율 다이어그램

우선 모든 태스크가 도착하는 율은 평균 λ이다. 다음은 서비스 율인데 (i+j)가 M이하 일 때는 항상 정확한 서비스만 일어나므로 서비스율은 평균 μ이며, (i+j)가 M일 때는 정확한 서비스와 부정확한 서비스 중 하나가 일어나므로 이들의 발생율은 각각 평균 $(\delta_0\mu)$ 혹은 평균 $\{\delta_X(K+1)\mu\}/(K+1-X)$이다.

V-4. 도착시 상태(i, 0)이 될 확률

우리는 시스템 모델을 M/M/1/M로 가정하고, P_i를 시스템이 i개의 태스크들을 가질 확률로, ρ를 제공된 부하(offered load) λ/μ로 각각 가정한다. M/M/1/M 모델에서 시스템이 현재 i개의 태스크들을 가질 확률 P_i를

$$P_i = P_o(\frac{\lambda}{\mu})^i = P_o\rho^i = \frac{\rho^i}{\sum_{j=0}^{M}\rho^j} = \frac{\rho^i(1-\rho)}{1-\rho^{M+1}}$$

로 정의할 수 있다.

다음은 특정 태스크가 시스템에 도착하여 시스템 상태(i, 0)이 될 확률을 P_{i0}라고 할 때 이를 아래에서 구해본다.

CASE 1 : i = 1, 2, ..., M−1

(1) i = 1 →

특정 태스크가 도착하여 상태(1,0)이 되려면 도착하기 전의 상태가 (0,0)이어야 하므로 도착하여 (1,0)이 되어야 할 확률은 $P_{10} = P_0$이 된다.

(2) i = 2 →

특정태스크가 도착하여 상태 (2,0)이 되려면 도착하기 전의 상태가 (1,0)이어야 하므로 도착하여 상태(2,0)이 되어야 할 확률은 $P_{10} = P_1$이 된다.

(3) i = 3 →

특정 태스크가 도착하여 상태(3,0)이 되려면 도착하기 전의 상태가 (1,1) 혹은 (2,0)이어야 하므로 도착하여 상태(3,0)이 될 확률은 $P_{10} = P_2$이 된다.

⋮

(M−1) i = M−1 →

특정 태스크가 도착하여 상태(M−1, 0)이 되려면 도착하기 전의 상태가 (M−2, 0), (M−3, 0), (M−4, 0), ..., (M−i, M−i+i−2)이어야 하므로 도착하여 상태가 (M−1, 0)이 될 확률은 $P_{i0} = P_{M-2}$이 된다.

⇒ 결론적으로 CASE 1의 모든 경우 특정 태스크가 시스템에 도착시 상태(i, 0)이 될 확률은 $P_{i0} = P_{i-1}$(i = 1, 2, ..., M−1)이 된다.

CASE 2 : i = M

(1) i = M →

특정 태스크가 도착하여 시스템 상태 (M, 0)이 되려면 도착하기전 상태가 (M−1, 0), (M−2, 1), ..., (M−i, i−1)이거나 (M, 0)일 때이다. 시스템이 상태(M−1, 0), (M−2, 1), ..., (M−i, i−1)들일 확률은 P_{M-1}이며, 시스템이 상태 (M, 0)인 후자의 경우는 특정 태스크가 도착하면 현재 서버의 태스크가 부정확한 계산이 되어 시스템을 떠나 버리므로 역시 다시 상태(M, 0)을 유지하게 되므로 확률은 P_M이다. 따라서 $P_{M0} = P_{M-1} + P_M$이다.

⇒ 결론적으로 CASE 2의 경우 특정 태스크가 시스템에 도착시 상태(M, 0)이 될 확률은 $P_{M0} = P_{M-1} + P_M$이다.

V-5. 상태(i,j)에서 여러 사건들이 발생할 확률

상태 (i, j)에서 일어날 수 있는 사건(event)은 도착 사건과 서비스 사건이며, 서비스 사건은 다시 정확한 서비스 사건과 부정확한 서비스 사건으로 나누어 볼 수 있다. 변수 t를

$$t = \sum_{X=1}^{k} (\delta_X(K+1)\mu)/(K+1-X))$$라고 정의했을 때

이러한 사건들은 각각 $(\lambda+\mu)$율 혹은, $(\lambda+\delta_0\mu+t)$율로 발생하는 포아송 프로세스들이며 우리는 시스템이 상태 (i, j)에 있을 때 이러한 사건들이 발생할 확률들을 여러 가지로 분류해서 아래의 수식으로 표현 한다.

a = Prob [i+j < M인 상태 (i, j)일 때 도착사건이 일어나는 경우]

$$= \frac{\lambda}{\lambda+\mu} \quad (1)$$

b = Prob [i+j < M인 상태 (i, j)일 때 정확한 서비스 사건이 일어나는 경우]

$$= \frac{\mu}{\lambda+\mu} \quad (2)$$

c = Prob [i+j = M인 상태 (i, j)일 때 도착 사건이 일어나는 경우]

$$= \frac{\lambda}{(\lambda+\delta_0\mu+t)} \quad (3)$$

d = Prob [i+j = M인 상태 (i, j)일 때 정확한 서비스 사건이 일어나는 경우]

$$= \frac{\delta_0\mu}{(\lambda+\delta_0\mu+t)} \quad (4)$$

e_X = Prob [i+j=M인 상태 (i, j)일 때 부정확한 서비스 사건이 일어나는 경우]

$$= \frac{\{(\delta_X(K+1)\mu)/(K+1-X)\}}{\lambda+\delta_0\mu+t} \quad (5)$$

V-6. FCFS에서 시스템 상태 (i,j)일 때 부정확한 확률

시스템이 상태(i, j)일 때 위치i의 특정 태스크가 부정확한 계산이 될 확률을 구하기 위해서 아래와 같은 함수를 정의하며 여기에서 특정 태스크가 부정확한 계산이 발생할 때는 자신의 K개의 선택적인 서브태스크들 중 X(X=1, 2, ..., K)개를 버린다고 가정한다.

$f_X(i, j)$ = Prob[FCFS에서 상태 (i, j)일 때 위치i의 특정 태스크가 언젠가는 결국 부정확한 계산이 되어 시스템을 나가는 경우]

우리는 여기서 시스템이 상태(i, j)일 때 위치i의 특정 태스크가 부정확한 계산이 될 확률을 의미하는 함수$f_X(i, j)$를 가지고서, 여러가지 시스템 상태에 있는 특정 태스크의 부정확한 확률을 구체적으로 구해 볼 수 있다.

(1) i=1인 경우

(1.1) $f_X(1,0)$, $f_X(1,1)$, ..., $f_X(1, M-2)$

이러한 상태들은 두가지 사건 즉, 도착사건과 정확한 서비스 사건이 발생할 수 있으며 이 사건들이 발생할 확률은 각각 앞절에서 언급한 확률 a, b라 할 수 있다. 그런데 이러한 사건들 중 도착 사건이 발생하는 경우만 상태가 (1, j)에서 (1, j+1)(j=0, 1, ..., M-2)로 바뀌게 되어 부정확한 확률 계산을 하게 된다. 따라서 부정확한 확률은 $f_X(1, j) = a * f_X(1, j+1)$(j=0, 1, 2, ..., M-2)가 된다.

(1.2) $f_X(1, M-1)$

이 상태에서는 도착사건과, 정확한 서비스 사건과, 부정확한 서비스 사건들이 발생할 수 있는데 이 세가지 사건 중 부정확한 계산이 되는 경우는 도착사건이 발생하는 경우이다. 따라서 이 상태에서의 부정확한 확률 $f_X(1, M-1)=c$이다.

(2) i≥2일 때의 $f_X(i,j)$

(2.1) i+j<M인 경우

이런 경우에는 도착 사건과, 정확한 서비스 사건이 발생할 수 있으며 각각의 사건들이 발생할 확률은 각각 a, b이다. 도착 사건이 발생하는 경우에는 상태가 (i, j)에서 (i, j+1)로 바뀌고, 정확한 서비스 사건이 발생할 경우는 상태가 (i, j)에서 (i-1, j)로 바뀌게 된다. 따라서, 부정확한 확률 $f_X(i, j)$는 $f_X(i, j) = a * f_X(i, j+1) + b * f_X(i-1, j)$로 쓸 수 있다.

(2.2) i+j=M인 경우

이런 경우에는 도착사건과, 정확한 서비스 사건과, 부정확한 서비스 사건들이 일어날 수 있으며 이들은 확률 c, d, e_X로 발생한다고 볼 수 있다. 따라서 도착 사건이 발생하는 경우에는 시스템의 총 테스크수가 (M+1)이므로 현재 서버의 태스크는 부정확한 계산이 되어 시스템을 나간다. 그래서 상태가(i, j)에서 (i-1, j+1)로 바뀌게 된다. 정확한 서비스 사건과 부정확한 서비스 사건의 경우에는 서버에 있는 태스크가 시스템을 나가므로 상태(i, j)에서 상태(i-1, j)로 바뀌게 되어 이러한 경우의 부정확한 확률 $f_X(i, j)$는 $f_X(i, j) = c * f_X(i-1, j+1) + d * f_X(i-1, j) + e_X * f_X(i-1, j)$로 쓸 수 있다. 위에서 언급한 모든 경우를 아래의 수식으로 묘사해볼 수 있다.

$$(1)\ f_X(1, j) = \frac{\lambda}{\lambda+\mu} f_X(1, j+1)$$

(j<M-1인 경우)

$$(2)\ f_X(1, M-1) = \frac{\lambda}{\lambda+\delta_0\mu+t}$$

$$(3)\ f_X(i, j) = \frac{\lambda}{\lambda+\mu} f_X(i, j+1) + \frac{\mu}{\lambda+\mu} f_X(i-1, j)$$

(i≥2, i+j<M인 경우)

$$(4)\ f_X(i, j) = \frac{\lambda}{\lambda+\delta_0\mu+t} f_X(i-1, j+1) + \frac{\delta_0\mu}{\lambda+\delta_0\mu+t} f_X(i-1, j) = \frac{\{(\delta_X(K+1)\mu)/(K+1-X)\}}{\lambda+\delta_0\mu+t} f_X(i-1, j)$$

(i≥2, i+j=M인 경우)

V-7. FCFS에서의 부정확한 확률

이 논문은 어느 특정 태스크가 시스템에 도착하는

시점부터 생각해 보려고 한다. 어느 특정 태스크가 도착해서 시스템 상태 (i,0)가 (i=1, 2, 3, 4, ..., M)될 확률 P_{i0}는 앞의 V-4에서 구하였듯이 다음과 같다.

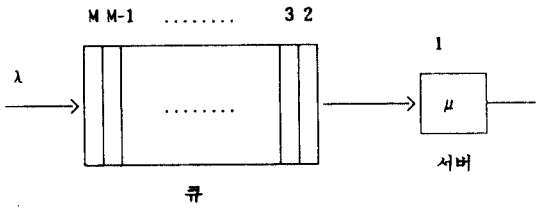
$$P_{i0}=\begin{cases} P_{i-1} & (i=1, 2, \ldots, M-1\text{인 경우}) \\ P_i+P_{i-1} & (i=M\text{인 경우}) \end{cases}$$

따라서 특정 태스크가 도착하여 초기에 상태(i,0)이 되고 이 상태에서 부정확한 계산이 될 확률을 F라고 할 때 $F=P_{i0}*f_X(i,0)$이 된다. 바로 F가 우리가 이 논문에서 구하고자 하는 FCFS방법에서의 부정확한 확률이다.

Ⅵ. LCFS방법에서의 부정확한 확률 계산

Ⅵ-1. 상태(state) 정의

역시 우리가 해결해야 할 문제는 어떤 특정 태스크가 시스템에 도착하여 부정확한 계산이 될 확률이므로 어떤 특정 태스크가 시스템에 도착했을 때부터 시작해서 이 태스크가 정확한 계산 혹은 부정확한 계산이 되어 시스템을 떠날 때까지 시스템안에서 이동하는 위치를 추적할 필요가 있다. 그래서 우리 시스템을 FCFS와 똑같은 방법으로 (그림 5)처럼 서버를 1번으로, 큐의 맨 앞부분에서부터 2, 3, ..., M번으로 정하기로 한다.



(그림 5) 위치(position) 번호

특정 태스크의 부정확한 확률 계산에는 아무래도 이 태스크의 시스템안에서의 위치와 시스템 전체의 태스크 수를 생각해 보아야 하므로 2차원의 상태 즉, 상태(i, j)를 정의한다. 이때 i는 특정 태스크의 시스템에서의 위치이고, j는 위치i 뒤에 남아 있는 다른 태스크들의 갯수를 나타내어 i와 j를 합하면 시스템 안의 전체 태스크들의 수를 나타낼 수 있다.

Ⅵ-2. 상태(state)들간의 전이(transition)

Ⅵ-2-1. 도착사건과 서비스사건

CASE 1 : LCFS의 도착사건의 발생

CASE 1.1 : 전체 태스크 수가 M이하일 때

- 도착한 태스크 : 현재 시스템 위치의 맨 끝에 도착함.
- 다른 태스크들 : 위치 변화 없음.

CASE 1.2 : 전체 태스크 수가 M을 초과할 때

- 위치 1번의 태스크 : 부정확한 계산이 되어 K개 중 X개의 (X≤K)개의 선택적인 서브태스크들을 실행하지 않고 시스템을 떠난다.
- 위치 2, 3, ..., M−1번의 태스크들 : 위치 변화 없음.
- 위치 M번의 태스크 : 위치 1의 서버로 가서 실행됨.
- 도착한 태스크 : 위치 M번에 위치.

CASE 2 : LCFS 서비스사건의 발생

- 위치 1번의 태스크 : 서비스(정확한 혹은 부정확한 서비스)되어 시스템을 나간다.
- 큐의 맨 끝번의 태스크 : 큐에서 빠져나와 서버로 가서 실행됨.
- 그외의 태스크들 : 위치 변동없음.

Ⅵ-2-2. 상태들간의 전이(transition)

상태 (i, j)에서 발생되는 전이를 생각해 보자.

CASE 1 : LCFS의 도착 사건 발생 시의 전이

CASE 1.1 : 전체 태스크 수가 M이하일 때

상태(i, j) → 상태(i, j+1)

CASE 1.2 : 전체 태스크 수가 M을 초과할 때

- 상태(1, M−1) → 상태(0, M)
- 상태(M, 0) → 상태(1, M−1)
- 그외의 상태(i, M−i) → 상태(i, M−i)

CASE 2 : LCFS의 서비스 사건(정확한 사건 혹은 부정확한 사건)발생 시의 전이

CASE 2.1 : i=1일 때

상태(1, j) → 상태(0, j)

CASE 2.2 : i=2, 3, ..., M일 때

CASE 2.1.1 : j=0일 때

상태(i, 0) → 상태(1, i−2)

CASE 2.1.2 : j가 0이 아닐 때

상태(i, j) → 상태(i, j−1)

Ⅵ-3. 상태-전이-율 다이어그램(state-transition-rate diagram)

시스템이 상태(i, 0)이 되었다고 가정하고 시작하려고 하며, 상태 천이-율 다이어그램을 아래 (그림 6)과 같이 나타내 볼 수 있다. 또한 아래 그림의 간단한 표현을 위하여 현재 서버의 태스크가 부정확한 계산을 하게될 때 X개의 선택적인 서브태스크들을 버릴 확률을 δ_X라고 나타내며 X의 값은 1, 2, ..., K라고 가정한다.

우선 모든 태스크가 도착하는 율은 평균 λ이다. 다음은 서비스 율인데 (i+j)가 M이하 일 때는 항상 정확한 서비스만 일어나므로 서비스율은 평균 μ이며, (i+j)가 M인 때는 정확한 서비스와 부정확한 서비스 중 하나가 일어 나므로 이들의 발생율은 각각 평균 $(\delta_0\mu)$ 혹은 평균 $(\delta_X(K+1)\mu)/(K+1-X)$이다.

Ⅵ-4. 특정 태스크가 시스템 도착시 상태(i,0)이 될 확률

우리의 시스템 모델은 FCFS와 같이 M/M/1/M로 가정하고, P_i를 시스템이 i개의 태스크들을 가질 확률로, ρ를 제공된 부하(offered load) λ/μ로 각각 가정한다. M/M/1/M모델에서 시스템이 현재 i개의 태스크를 가질 확률 P_i를

$$P_i = P_0\left(\frac{\lambda}{\mu}\right)^i = P_0\rho^i = \frac{\rho^i}{\sum_{j=0}^{M}\rho^j} = \frac{\rho^i(1-\rho)}{1-\rho^{M+1}}$$

로 정의할 수 있다.

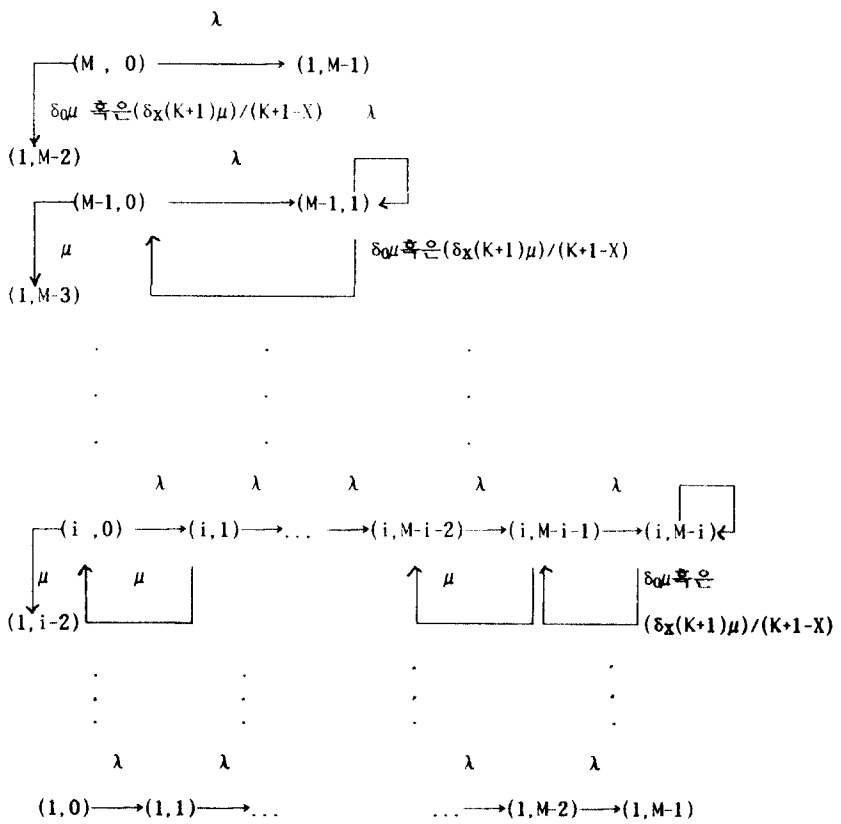
다음은 특정 태스크가 시스템에 도착하여 시스템 상태(i, 0)이 될 확률을 P_{i0}라고 나타낼 때 FCFS와 같이

$$P_{i0} = \begin{cases} P_{i-1} & (i = 1, 2, \ldots, M-1\text{인 경우}) \\ P_i + P_{i-1} & (i = M\text{인 경우}) \end{cases}$$

로 정의할 수 있다.

Ⅵ-5. 상태(i,j)에서 여러 사건들이 발생할 확률

상태 (i, j)에서 일어날 수 있는 사건(event)은 도



(그림 6) LCFS의 상태 천이 율 다이어그램

착 사건과 서비스 사건이며, 서비스 사건은 다시 정확한 서비스 사건과 부정확한 서비스 사건으로 나누어 볼 수 있다. 변수 t를

$t=\sum_{X=1}^{K}(\delta_X(K+1)\mu)/(K+1-X))$라고 정의했을 때

이러한 사건들은 각각 $(\lambda+\mu)$율 혹은, $(\lambda+\delta_0\mu+t)$율로 발생하는 포아송 프로세스들이며 이러한 사건들이 발생할 확률을 여러 가지로 분류해서 수식으로 표현 한다.

a = Prob [i+j<M인 상태 (i,j)일 때 도착사건이 일어나는 경우]

$$=\frac{\lambda}{\lambda+\mu} \tag{6}$$

b = Prob [i+j<M인 상태 (i,j)일 때 정확한 서비스 사건이 일어나는 경우]

$$=\frac{\mu}{\lambda+\mu} \tag{7}$$

c = Prob [i+j=M인 상태 (i,j)일 때 도착 사건이 일어나는 경우]

$$=\frac{\lambda}{(\lambda+\delta_0\mu+t)} \tag{8}$$

d = Prob [i+j=M인 상태 (i,j)일 때 정확한 서비스 사건이 일어나는 경우]

$$=\frac{\delta_0\mu}{(\lambda+\delta_0\mu+t)} \tag{9}$$

e_X = Prob [i+j=M인 상태 (i,j)일 때 부정확한 서비스 사건이 일어나는 경우]

$$=\frac{\{(\delta_X(K+1)\mu)/(K+1-X)\}}{\lambda+\delta_0\mu+t} \tag{10}$$

Ⅵ-6. LCFS에서 시스템 상태(i,j)일 때 부정확한 확률

먼저 상태(i,j)일 때 i위치에 있는 특정 태스크의 부정확한 확률을 구하기 위해서 아래와 같이 FCFS와 똑같은 함수를 정의해 볼 수 있다.

$g_X(i,j)$ = Prob[LCFS에서 상태 (i,j)일 때 i위치의 태스크가 언젠가는 결국 부정확한 계산이 되어 시스템을 나가는 경우]

위에서 정의한 함수 $g_X(i,j)$는 시스템 상태(i,j)일 때 i위치에 있는 특정 태스크의 부정확한 확률을 의미한다. 우리는 여기서 시스템이 여러가지 경우의 상태(i,j)일 때 위치 i번의 특정 태스크가 부정확한 계산이 되는 부정확한 확률들을 구체적으로 구한다.

(1) i = 1인 경우

(1.1) $g_X(1,0), g_X(1,1), \ldots, g_X(1, M-2)$

이러한 상태들은 두가지 사건 즉, 도착사건과 정확한 서비스 사건이 발생할 수 있으며 이 사건들이 발생할 확률은 각각 앞에서 언급한 확률 a, b라 할 수 있다. 이중에서 도착 사건이 발생하는 경우에는 상태가 (1, j)에서 (1, j+1)(j=0, 1, ..., M-2)로 바뀌게 되어 부정확한 계산이 되므로 이러한 경우 부정확한 확률 $g_X(1, j)=a*g_X(1, j+1)(j=0, 1, 2, \ldots, M-2)$가 된다.

(1.2) $g_X(1, M-1)$

이 상태에서는 도착사건과 정확한 서비스사건과 부정확한 서비스사건이 각각 발생할 수 있는데 이런 세가지 사건 중 도착사건이 발생하면 부정확한 계산이 되므로 부정확한 확률 $g_X(1, M-1)=c$이다.

(2) i = 2, 3, ..., M-1일 때의 $g_X(i, j)$

(2.1) i+j<M인 경우

(2.1.1) j=0인 경우

이런 경우에는 도착사건과 정확한 서비스 사건이 발생할 수 있으며 각각의 사건이 발생할 확률은 각각 a, b이다. 도착사건이 발생한 경우에는 상태가 (i, 0)에서 (i, 1)로 바뀌고 정확한 서비스 사건이 발생할 경우에는 상태가 (i, 0)에서 (1, i-2)로 바뀌게 된다. 따라서 이러한 경우의 부정확한 확률 $g_X(i, 0)=a*g_X(i, 1)+b*g_X(1, i-2)$로 쓸 수 있다.

(2.1.2) $j\neq 0$인 경우

이런 경우에는 도착 사건과, 정확한 서비스 사건이 발생할 수 있으며 각각의 사건들이 발생할 확률은 각각 a, b이다. 도착 사건이 발생하는 경우에는 상태가 (i, j)에서 (i, j+1)로 바뀌고 정확한 서비스 사건이 발생할 경우는 상태가 (i, j)에서 (i, j-1)로 바뀌게 된다. 따라서, 이러한 경우의 부정확한 확률 $g_X(i, j)$는 $g_X(i, j)=a*g_X(i, j+1)+b*g_X(i, j-1)$로 쓸 수 있다.

(2.2) i+j=M인 경우

이런 경우에는 도착사건과, 정확한 서비스 사건과, 부정확한 서비스 사건들이 일어날 수 있으며 이들은

확률 c, d, e_X로 발생한다고 볼 수 있다. 따라서 도착사건이 발생하는 경우에는 시스템의 총 태스크수가 (M+1)이므로 현재 서버의 태스크는 부정확한 계산이 되어 시스템을 나가고, 도착하기전 큐의 맨끝 위치의 태스크가 서버로 가서 실행되고 방금 도착한 태스크는 큐의 맨 끝으로 오기 때문에 상태는 (i, j)에서 (i, j)로 바뀌게 된다. 정확한 서비스 사건과 부정확한 서비스 사건의 경우에는 서버에 있는 태스크가 시스템을 나가고, 큐의 맨마지막의 태스크가 서버로 가서 실행되기 때문에 상태가 (i, j)에서 상태 (i, j-1)로 바뀌게 되어 부정확한 확률 $g_X(i, j) = c * g_X(i, j) + d * g_X(i, j-1) + e_X * g_X(i, j-1)$로 쓸 수 있다.

(3) i = M인 경우

이 경우 도착사건, 정확한 서비스사건, 부정확한 서비스사건등이 발생할 수 있으며 이들은 각각 확률 c, d, e_X로 발생할 수 있다. 이때 도착사건이 발생할 때에는 상태가 (M, 0)에서 (1, M-1)로 바뀌며, 정확한 서비스 사건과 부정확한 서비스 사건이 발생할 때에는 상태가 (M, 0)에서 (1, M-2)로 바뀌어 부정확한 확률 $g_X(M, 0) = c * g_X(1, M-1) + d * g_X(1, M-2) + e_X * g_X(1, M-2)$로 쓸 수 있다.

위에서 언급한 모든 경우를 아래의 수식으로 묘사해 볼 수 있다.

$$(1)\ g_X(1, j) = \frac{\lambda}{\lambda+\mu}\ g_X(1, j+1)$$

(j = 1, 2, ..., M-2인 경우)

$$(2)\ g_X(1, M-1) = \frac{\lambda}{\lambda+\delta_0\mu+t}$$

$$(3)\ g_X(i, 0) = \frac{\lambda}{\lambda+\mu}\ g_X(i, 1) + \frac{\mu}{\lambda+\mu}\ g_X(1, i-2)$$

(2≤i≤M-1, j = 0, i+j<M인 경우)

$$(4)\ g_X(1, j) = \frac{\lambda}{\lambda+\mu}\ g_X(i, j+1) + \frac{\mu}{\lambda+\mu}\ g_X(i, j-1)$$

(2≤i≤M-1, j≠0, i+j<M인 경우)

$$(5)\ g_X(i, j) = \frac{\lambda}{\lambda+\delta_0\mu+t}\ g_X(i, j) + \frac{\delta_0\mu}{\lambda+\delta_0\mu+t}\ g_X(i, j-1) + \frac{\{(\delta_X(K+1)\mu)/(K+1-X)\}}{\lambda+\delta_0\mu+t}\ g_X(i, j-1)$$

(2≤i≤M-1, j≠0, i+j = M인 경우)

$$(6)\ g_X(M, 0) = \frac{\lambda}{\lambda+\delta_0\mu+t}\ g_X(1, M-1) + \frac{\delta_0\mu}{\lambda+\delta_0\mu+t}\ g_X(1, M-2) + \frac{\{(\delta_X(K+1)\mu)/(K+1-X)\}}{\lambda+\delta_0\mu+t}\ g_X(1, M-2)$$

Ⅵ-7. LCFS에서의 부정확한 확률

이 논문은 어느 특정 태스크가 시스템에 도착하는 시점부터 시작해서 생각하려고 한다. 어느 특정 태스크가 도착해서 시스템 상태 (i, 0)가 (i = 1, 2, 3, 4, ..., M) 될 확률 P_{i0}는 앞의 Ⅵ-4에서 구한 FCFS방법의 P_{i0}와 같다. 이를 다시 한번 아래에 묘사한다.

$$P_{i0} = \begin{cases} P_{i-1} & (i = 1, 2, \ldots, M-1\text{인 경우}) \\ P_i + P_{i-1} & (i = M\text{인 경우}) \end{cases}$$

따라서, LCFS방법하에서 특정 태스크가 도착하여 초기에 상태(i,0)이 되고 이 상태에서 부정확한 계산이 될 확률을 G라고 할 때 $G = P_{i0} * g_X(i,0)$이 된다. 바로 이 값이 LCFS에서의 우리가 구하고자 하는 부정확한 확률이다.

Ⅶ. 성능 분석

지금까지 두개의 부정확한 스케쥴링 방법 즉, FCFS, LCFS에서 특정 태스크가 시스템의 위치 i에 도착하여 부정확한 계산이 될 부정확한 확률을 구하였으며, 이것을 가시적으로 보이기 위해 지금까지 나온 수식의 해석결과를 그래프로 묘사해 보았다. 이 논문은 시스템 매개변수 M, K, X가 주어졌을 때 특정 태스크가 시스템 위치 i에 도착하여 결국에는 부정확한 계산이 될 확률을 구하여 비교 분석하였다. 우리는 특정 태스크가 위치 i에 도착함을 가정하였는데 그래프의 성능 결과에서는 이 때 위치 i를 구체적으로 M-1로 일률적으로 가정하고 부정확한 확률을 구하였다. 해석 결과 위치 i를 M-1이 아닌 다른 위치로 하더라도 이논문에서 언급하고 있는 부정확한 결과는 비슷한 결과를 이끌어 내어서 i를 M-1로 두어도 그렇게

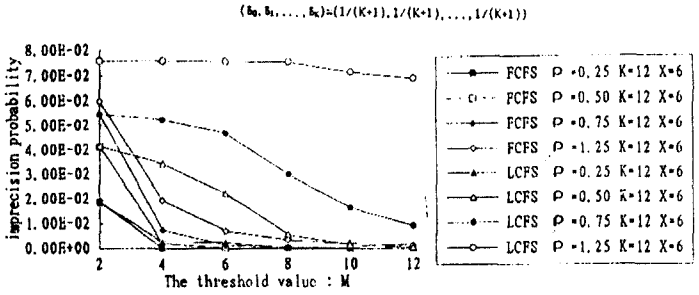
무리가 없다는 결론입니다. 그리고 우리는 현재의 서버의 태스크가 정확한 계산을 할 확률, 부정확한 계산을 할 확률들인(δ_0, δ_1, ..., δ_K)의 값을 여러가지로 구분하여 부정확한 확률을 구하였다. 이해를 돕기 위하여 시스템 매개변수 M, K, X를 다시 한번 더 정의해 보기로 하자.

M → 특정 태스크가 도달한 후 시스템안의 태스크들 수가 M보다 크면 현재 서버에서 실행 시키고 있는 태스크를 부정확한 계산을 실행시키고, M 이하이면 정확한 계산을 하도록 결정하는 매개변수

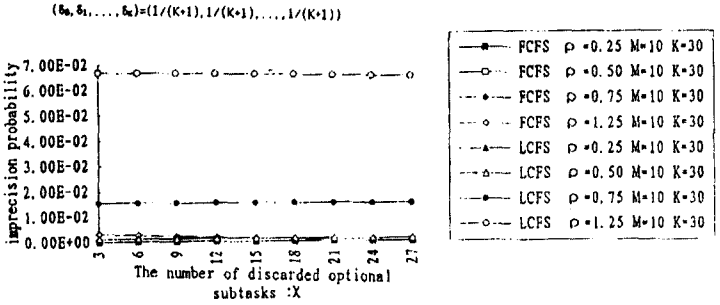
K → 태스크를 한개의 강제적인 서브태스크와 K개의 선택적인 서브태스크들로 분해했을 때 사용되는 매개변수

X → 서버의 태스크가 부정확한 계산을 하게 되는 경우에 자신의 K개의 선택적인 서브태스크들 중 X개의 선택적인 서브태스크를 버리게 될 때 사용되는 매개변수

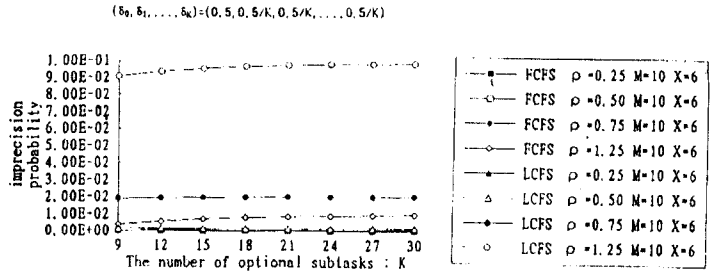
여러 결과들을 (그림 7), (그림 8), (그림 9), (그림 10), (그림 11), (그림 12)에 나타내었다. (그림 7)은 부정확한 확률의 위에서 정의한 M에 대한 민감성을 나타내는 그래프이다. 그림에서 볼 수 있듯이 태스크들의 정확한, 부정확한 계산을 결정하는 매개변수인 M이 커짐에 따라 부정확한 확률이 FCFS, LCFS는 부정확한 확률이 점점 감소하는 경향을 보이고 있다. (그림 8)은 부정확한 확률의 K개의 선택적인 서브태스크들 중 과부하시 버려지는 선택적인 서브태스크들 갯수 X에 대한 민감성을 나타내고 있다. 그림에서 볼 수 있듯이 버려지는 선택적인 서브태스크들 수가 증가할수록 FCFS, LCFS에서 부정확한 확률이 점점 더 감소함을 보이고 있다. 이는 두 방법 다 버려지는 서브태스크들 수가 많아질수록 태스크들이 부정확한 계산을 할 빈도 수가 감소하기 때문이다. (그림 9), (그림 10), (그림 11), (그림 12)은 부정확한 확률의 한 태스크의 선택적인 서브태스크들 갯수인 K에 대한 민감성을 나타내고 있다. 이런 그래프에서는 (δ_0, δ_1, ..., δ_K)의 값을 여러가지로 구분하여 결과를 나타내었다. (그림 9), (그림 10)에서는 일반적으로 K가 커짐에 따라 FCFS, LCFS에서 부정확한 확률이 증가하고 있는 것을 알 수 있다. 이



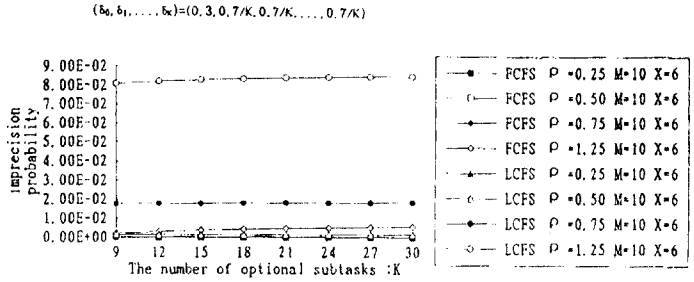
(그림 7) M에 대한 민감성



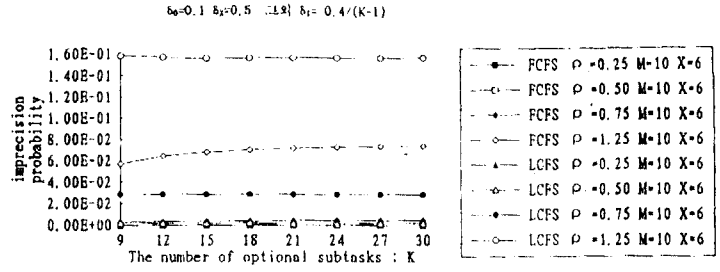
(그림 8) X에 대한 민감성



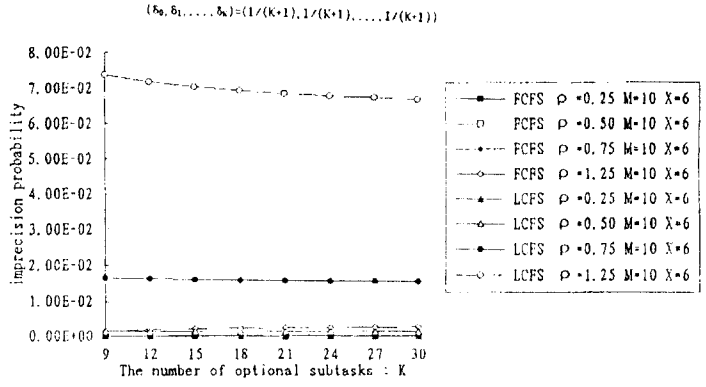
(그림 9) K에 대한 민감성



(그림 10) K에 대한 민감성



(그림 11) K에 대한 민감성



(그림 12) K에 대한 민감성

는 고정된 X에 대해 K가 증가한다는 것은 버려지는 선택적인 서브태스크들 수 X가 적어진다는 것과 같기 때문에 (그림 8)과 같은 맥락에서 K가 증가할수록 평균 부정확한 확률이 증가한다. 그런데, (그림 11), (그림 12)의 LCFS에서는 K가 커짐에 따라 부정확한 확률이 약간씩 감소함을 나타내고 있는데 δ_0의 값이 다른 모든 δ_i의 합에 비해서 적을 때 이런 현상이 나타나고 있음을 알 수 있다.

Ⅷ. 결 론

실시간 시스템의 스케쥴링 유연성을 제공하는 수단으로 등장하고 있는 부정확한 계산기법의 간단한 검토와 함께 지금까지의 부정확한 스케쥴링 문제들을 살펴보았다. 이러한 문제들은 최적의 스케쥴을 구하는 결정적인 공식을 요구하는 것들이었는데 반해 이 논문에서는 큐잉이론에 입각한 비결정적인 공식을 사용하였다. 지금까지 진행되어온 큐잉이론의 방법들은 대부분이 두단계 스케쥴링에 입각한 문제 해결이었으며 이러한 방법에 의해서 주로 다루어온 문제는 태스크들의 평균 결과의 질과 평균 대기시간 사이의 절충(trade-off)이었다. 이 논문은 태스크를 한개의 강제적인 서브태스크와 K개의 선택적인 서브태스크들로 분해하는 다른 단조형 부정확한 시스템을 사용하며, 시스템에 과부하가 발생했을 때 사용자들의 빠른 응답시간을 위해 K개의 선택적인 서브태스크들 중에서 X($X \leq K$)개의 선택적인 서브태스크들을 실행시키지 않음으로써 과부하 발생시 시간 결함을 해결해 주도록 하였다. 이 논문은 FCFS의 부정확한 스케쥴링 방법과 LCFS의 부정확한 스케쥴링 방법하에서 특정 태스크가 시스템에 도착하여 언젠가는 결국 부정확한 계산이 되어서 시스템을 빠져 나갈 확률 즉, 부정확한 확률들을 구하였으며 이러한 확률들이 시스템의 여러 매개변수 M, K, X와 어떤 의존성이 있는 지를 비교 분석해 보았다. 우리는 이 확률을 재귀함수(recursive function)를 정의하여 구하였다. 앞으로의 연구 과제는 MLF(minimum Laxity first) 기법하에서의 이런 부정확한 확률을 구하는 문제에 대한 연구와 이 논문에서 구한 부정확한 확률들을 사용하여 평균대기시간과 부정확한 확률을 절충하는 문제에 대한 연구가 더 진행되어야 한다고 본다.

참 고 문 헌

1. J.W.S.Liu, K.J.Lin, W.K.Shin, A.C.S.Yu, J.Y. Chung, and W.Zhao, "Algorithms for Scheduling Imprecise Computations," Computer, 1991, pp.58-68.
2. E.K.P.Chong and W.Zhao, "Task Scheduling for Imprecise Computer Systems with User Controlled Optimization," Computing and Information, Elsevier Science Publishers, North Holland, 1989.
3. C.L.Lin and J.W.Layland, "Scheduling Algorithms for Multiprogramming in a Hard Read-Time Environment," J.ACM, Vol.20, No.1, Jan. 1973, pp.46-61.
4. J.Y.Chung, J.W.S.Liu, and K.-J.Lin, "Scheduling Periodic Jobs That Allow Imprecise Results," IEEE Trans. Computers, Vol.19, No.9, Sept. 1990, pp.1,156-1,173.
5. R.McNaughton, "Scheduling with Deadlines and Loss Functions," Management Science, Vol.12, No.1, Oct. 1959, pp.1-12.
6. J.F.Kurose and R. Chipalkatti, "Load Sharing in Soft Real-Time Distributed Computer Systems," IEEE Trans. on Computers, Vol.c-36, No.8, August 1987.
7. D.W. Craig and C.M.Woodside, "The Rejection Rate for Tasks with Random Arrivals, and Preemptive Scheduling", IEEE Trans. on Software Engineering, Vol.16, No.10, October 1990.
8. B.T.Doshi and H.Heffes, "Comparision of Control Schemes for a Class of Distributed Systems," in Proc. 21st IEEE Conf. Decision Contr., Orlando, FL, Dec. 8-10, 1982, pp.846-853.
9. B.T.Doshi and H.Heffes, "Overload Performance of Several Processor Disciplines for the M/M/1 Queue," IEEE Trans. Communication Vol.c-35, pp.538-546, June 1986.
10. L. Kleinrock, Queueing Systems, Vol.1 : Theory. New York : Wiley, 1975.
11. L.Kieinrock, Queueing Systems, Vol.2 : Computer Applications. New-York : wiley, 1976.
12. W.Zhao and E.K.P.Chong, "Performance Evaluation of Scheduling Algorithms for Dy-

namic Imprecise Soft Real-TIme Computer Systems," Australian Computer Science Communications 11, 1989, pp.329-340.
13. Lin, K.J., S.Natarajan, J.W.S.Liu and T.Krauskopf, "Concord : A System of Imprecise Computations," Proceedings of the 1987 IEEE Comspac, pp.75-81, October, 1987.
14. Lin, K.J., S.Natarajan, and J.W.S. Liu, "Imprecise Results : Utilizing Partial Computations in Real-Time Systems," Proceedings of the IEEE 8th Real-Time Systems Symposium, December 1987.
15. S. Natarajan and K.J.Lin, "Expressing and Maintaining Timing Constraints in FLEX," Proceedings of 9th IEEE Real-Time Systems Symposium, pp.96-105, 1988.
16. Shih, W.K., J.Y. Chung, J.W.S. Liu and D. W. Gillies, "Scheduling Tasks with Ready Times and Deadlines to Minimize Average Error," ACM Operating Systems Review, pp. 14-28, July 1989.
17. Leung, J.Y-T., T.W.Tam, C.S.Wong and G. H.Wong, "Minimizing Mean Flow Time with Error Constraints," Proceedings of the IEEE 10th Real-Time Systems Symposium, Dec. 1989.

安 貴 任(Gwi Im Ahn) 정회원
1983년 : 부산대학교 계산통계학과 졸업(전산학)
1986년 : 서울대학교 계산통계학과 대학원 이학석사(전산학)
1988년 : 서울대학교 대학원 계산통계학과 박사과정 수료
1989년~현재 : 동의대학교 자연과학대학 전산통계학과 조교수
※주관심분야 : 운영체제, 성능분석, 분산시스템, 실시간시스템.

高 健(Kern Koh) 정회원
서울대학교 공과대학 응용물리학과 졸업
1981년 : 미국 버지니아 대학교 전산학 박사
1974년~1976년 : KIST 연구원
1981년~1983년 : 미국 Bell 연구소 연구원
1983년~현재 : 서울대학교 자연과학대학 계산통계학과 교수
※주관심분야 : 운영체제, 계산기구조, 계산기 성능분석, 분산 및 병렬처리