양극단 제약을 갖는 비주기, 주기 태스크와 메시지 스케줄링

Scheduling of Sporadic and Periodic Tasks and Messages with End-to-End Constraints

김 형 육*, 오 훈, 박 홍 성
(Hyoung Yuk Kim, Hoon Oh, and Hong Seong Park)

Abstract : The scheduling methods of the distributed real-time systems have been proposed. However, they have some weak points. They did not schedule both sporadic and periodic tasks and messages at the same time or did not consider the end-to-end constraints such as precedence relations between sporadic tasks. This means that system scheduling must guarantee the constraints of practical systems and be applicable to them. This paper proposes a new scheduling method that can be applied to more practical model of distributed real-time systems. System model consists of sporadic and periodic tasks with precedence relations and sporadic and periodic messages and has end-to-end constraints. The proposed method is based on a binary search-based period assignment algorithm, an end-to-end laxity-based priority assignment algorithm, and three kinds of schedulability analysis, node, network, and end-to-end schedulability analysis. In addition, this paper describes the application model of sporadic tasks with precedence constraints in a distributed real-time system, shows that existing scheduling methods such as Rate Monotonic scheduling are not proper to be applied to the system having sporadic tasks with precedence constraints, and proposes an end-to-end laxity-based priority assignment algorithm.

Keywords : distributed real-time systems, end-to-end scheduling, sporadic task, laxity

I. 서론

분산 실시간 시스템을 구성하는 각 노드에서 수행되는 태스크들은 그 응용 모델에 따라 다양한 형태의 태스크들이 수행된다. 예를 들면, 알람 신호와 같이 이벤트에 의해 발생하는 비주기 태스크들, 샘플링이나 제어알고리즘을 수행하는 태스크와 같이 주기적으로 작업을 수행하는 주기 태스크들, 프로그램 혹은 환경설정 데이터를 다운로딩 하는 작업을 수행하는 비실시간 태스크들이 혼재해서 수행되게 된다. 또한 이러한 태스크들간의 통신을 위해 필드버스와 같은 실시간 특성을 갖는 네트워크를 이용하여 노드들을 서로 연결하게 되고 비주기, 주기, 비실시간 형태의 메시지들이 네트워크를 통해 송, 수신되게 된다.

이렇게 다양한 태스크와 메시지로 구성되는 분산 실시간 시스템에는 센서의 샘플링 시점에서부터 구동노드의 제어명령 출력까지의 양극단 시간제약, 어떤 태스크가 기동 된 후에 어떤 메시지가 전송되고 다시 어떤 태스크가 기동 되어야만 하는 태스크와 메시지 간의 선행제약, 두 개 이상의 센서노드로 구성되는 제어루프 경우 각 센서노드의 샘플링 시점 시간차를 어떤 범위 이내로 제한해야 하는 센서 데이터 동기화 문제 등과 같은 양극단 제약사양이 존재하며 시스템 스케줄링 시에 이러한 양극단 제약사양이 모두 만족하도록 고려되어야 한다. 이러한 시스템 제약사양 및 다양한 태스크와 메시지 등의 복잡한 고려사항 들로 인해 분산 실시간 시스템의 스케줄링은 NP-Hard 문제로 알려져 있으며[1], 휴리스틱(Heuristic) 방법을 사용한 분산 실시간 시스템의 sub-optimal 스케줄 결과를 얻는 연구들이 제안되어 왔다. 분산 실시간 시스템에서 양극단을 고려한 스케줄가능성(schedulability) 분석 연구로 주기, 비주기 태스크와 메시지에 대해 어떠한 태스크도 두 개 이상의 메시지를 받지 않는다는 가정 하에서 생산자 태스크의 최악응답시간을 소비자 태스크 또는 메시지의 릴리즈 지터(Release Jitter)로 설정하는 방법을 이용하여 TDMA를 사용하는 분산 실시간 시스템을 위한 전체적인 스케줄가능성 분석(Holistic Schedulability Analysis) 방법이 제안되었다[2]. 그러나 시스템의 스케줄가능성 분석 방법 만이 제안되었으므로 태스크와 메시지의 주기 및 우선순위를 적절히 설정하는 스케줄링 문제가 해결되지 못했으며 태스크가 메시지를 하나 밖에 받을 수 없다는 문제가 있다. 또한 태스크 기반의 스케줄링 방법[3]을 CAN으로 구성되는 분산 실시간 시스템으로 확장한 연구[4]에서는 여러가지 소거법을 통하여 다중 루프를 구성하는 태스크와 메시지의 주기를 할당하여 시스템을 스케줄하는 방법을 제안하였지만 태스크와 메시지의 우선순위 할당 방법은 제시되지 못했고 주기 태스크와 주기 메시지 만을 대상으로 하였다. [5]에서는 분산 실시간 시스템 스케줄링을 클러스터링 알고리즘과 유전적 알고리즘을 사용한 스케줄링 방식을 제안하였지만 태스크간 선행제약을 전혀 고려하지 않았고 특정한 태스크에 대한 특정 노드 할당에 대한 제약이 없다는 가정 하에서 제안되었으므로 각 노드가 수행하는 고유 태스크가 존재하는 실제 분산 실시간 시스템의 환경에 적용하기 어려운 문제가 있다. [6]은 이진검색 알고리즘을 이용하여 다중 루프로 구성되는 분산 실시간 시스템의 메시지 우선순위 설정 방법과 태스크 주기할당 방법

* 책임저자(Corresponding Author)
논문접수 : 2004. 5. 19., 채택확정 : 2004. 12. 23.
김형육, 오훈, 박홍성 : 강원대학교 전기전자정보통신공학부
(petrus@control.kangwon.ac.kr/hoonoh@cc.kangwon.ac.kr/hspark@cc.kangwon.ac.kr)

을 제안하고 양극단에서 스케줄가능성 분석을 수행하였지만 역시 주기 태스크와 주기 메시지만을 대상으로 하였으므로 실제 시스템 상에 존재하는 비주기 태스크를 고려하지 못했다는 단점이 있다. 주기 태스크와 비주기 태스크를 함께 고려한 스케줄가능성 분석 및 스케줄링 연구[7]에서는 네트워크를 통한 지연시간이 고정적이라는 가정 하에 slot-shifting 기법을 이용하여 off-line 스케줄과 on-line 스케줄 방법 결합한 주기 태스크와 비주기 태스크 스케줄링 방법을 제안하였다. 그러나 메시지로 인한 임의 지연시간을 고려하지 않았고, 또한 로컬 노드 내에서 독립적으로 수행되는 비주기 태스크만을 대상으로 하였기 때문에 비주기 태스크들 간의 선행제약을 고려하지 않았다. 비주기 태스크 경우 어떤 노드 A에서 발생한 이벤트를 노드 A에서만 처리하고 끝나는 형태의 태스크도 있지만 시스템 전체에 중대한 영향을 미칠 수 있는 알람 신호와 같은 이벤트는 로컬 노드의 비주기 태스크 뿐만이 아니라 시스템에 분산되어 있는 알람 처리 태스크들에 의해 처리되어야 하며 이러한 태스크들 간에는 어떤 태스크가 다른 태스크를 기동 시키는 선행제약이 존재하게 된다. 그러므로 비주기 태스크의 스케줄링도 선행제약을 고려하여 수행되어야 한다. 본 논문에서는 분산 실시간 시스템에서 비주기 태스크의 응용모델을 살펴보고 이들간에 선행제약이 발생하는 경우를 짚어본다. 비주기 태스크간 선행제약으로 인한 기존 스케줄가능성 분석의 문제점과 양극단 스케줄링 및 스케줄가능성 분석의 필요성을 제시한다. 선행제약을 갖는 비주기 태스크 및 주기 태스크와 비주기, 주기 메시지가 동시에 존재하는 분산 실시간 시스템을 위한 여유도 기반의 태스크와 메시지의 우선순위 할당 방법을 제안한다. 또한 제안된 우선순위 할당 방법과 기존에 제안된 주기할당 알고리즘[6]을 통해 노드 스케줄가능성 분석, 네트워크 스케줄가능성 분석, 양극단 스케줄가능성 분석으로 이루어지는 양극단 제약을 고려한 분산 실시간 시스템의 스케줄링 방법을 제안한다. 최종적으로 제안된 스케줄링 방법을 예제 시스템 적용하여 제안된 알고리즘의 효율성을 입증한다.

앞으로 2절에서 선행제약을 갖는 비주기 태스크의 응용모델과 비주기 태스크 스케줄가능성 분석시 고려사항을 서술한다. 3절에서는 양극단 제약을 고려하여 비주기, 주기 태스크와 비주기, 주기 메시지가 동시에 존재하는 분산 실시간 시스템의 스케줄링 방법을 제안하고 예제 시스템 적용 결과를 서술한다. 마지막으로 4절에서 결론을 맺는다.

II. 비주기 태스크의 선행제약

1. 선행제약을 갖는 비주기 태스크 모델

분산 실시간 시스템에는 제어루프를 구성하여 주기적으로 수행되는 주기 태스크뿐만 아니라 어떤 이벤트에 의해 기동되는 비주기 태스크도 존재한다.

이러한 비주기 태스크들은 I/O 입력 등과 같은 이벤트에 의해 기동 되어 해당 이벤트에 대한 처리 루틴을 로컬 노드에서 수행하고 종료되는 독립적 형태나 이벤트에 의해 기동된 태스크가 원격의 다른 태스크를 기동 시키는 태스크간 선행제약을 갖는 형태로 분류될 수 있다. 비주기 태스크간 선행제약을 가지는 응용모델로서 만약 센서의 샘플링 값이 시스템에 치명적 오류를 유발할 수 있는 범위라면 샘플링 태스크는 이에 대한 알람 신호를 제어노드 또는 구동노드로 전송하여 각 노드에 존재하는 알람 태스크를 기동 시켜 발생한 알람을 처리하게 된다. 또한 분산 실시간 시스템을 구성하는 스마트센서나 구동기는 자신의 오동작 검출 및 복구를 위하여 Watch Dog Timer와 같은 기능을 수행하고 있으며 오동작 시 재부팅을 통한 에러 복구뿐만 아니라 이에 대한 정보를 다른 노드에게 전송, 에러처리 태스크를 기동함으로써 발생한 에러로 인한 전체 시스템 오류를 방지하게 된다. 위에서 언급한 바와 같이 하나의 노드 내에서만 수행을 완료하는 비주기 태스크들도 있지만 시스템 전체에 영향을 미치는 이벤트인 경우에는 이벤트로 인해 기동 된 비주기 태스크가 다른 노드의 비주기 태스크를 기동 시키게 되는 태스크간 선행제약 관계가 발생하게 된다. 이러한 이벤트들은 시스템 전체에 영향을 미치기 때문에 발생한 이벤트에 대한 데드라인 즉, 처리 완료 시간 제약을 처음 기동 되는 비주기 태스크의 기동시점에서 마지막으로 기동 되는 비주기 태스크의 수행완료 시점까지로 보아야 한다.

예를 들어 발생한 알람에 대한 데드라인이 10ms이고 이 알람을 처리하는 비주기 태스크들인 센서노드의 태스크 A, 제어노드의 태스크 B, 구동노드의 태스크 C 가 선행제약을 가지며 순서대로 수행된다고 가정하자. 이때 태스크 A, B, C가 각각 기동 된 시점에서 10ms의 데드라인을 만족하는지를 검사하여 스케줄가능성 분석을 수행하면 안되고 태스크 A가 기동 된 시점-알람이 발생한 시점-에서부터 태스크 C가 완료되는 시점-알람에 대한 처리가 완료되는 시점-까지가 10ms를 만족하는지를 평가하여야 한다.

2. 선행제약으로 인한 비주기 태스크의 스케줄가능성 영향

비주기 태스크간 선행제약이 없는 경우, 즉 비주기 태스크들이 독립적으로 수행되는 경우에 해당하는 간단한 분산 실시간 시스템이 그림 1에 나타나 있다. 원은 태스크를 나타나며 타원은 메시지를 화살표는 태스크간의 생산자-소비자 관계를 나타내며 표기법은 부록에 있다. 대상 시스템은 하나의 센서노드, 제어노드, 구동노드로 구성되며 노드간 통신은 네트워크를 통해 이루어진다. 센서노드에는 두 개의 주기태스크 τ_{S1}^{P1}, τ_{S1}^{P2} 와 하나의 비주기 태스크 τ_{S1}^{S1}, 제어노드에는 두 개의 주기 태스크, τ_{C1}^{P1}, τ_{C1}^{P2} 와 하나의 비주기 태스크 τ_{C1}^{S1}, 구동노드에는 두 개의 주기 태스크 $\tau_{A1}^{P1}, \tau_{A1}^{P2}$ 와 하나의 비주기 태스크 τ_{A1}^{S1} 가 존재한다. 시스템에는 하나의 제어루프가 존재하며 세 개의 주기태스크 τ_{S1}^{P1}, τ_{C1}^{P1}, τ_{A1}^{P1}와 두 개의 메시지 m_1^P, m_2^P 로 구성된다. 모든 노드는 고정 우선순위의 선점형 스케줄러 기반이고 선행제약을 갖는 주기태스크를 제외한 모든 태스크들은 서로 공유하는 자원 없이 수행되고 모든 태스크들이 t0 시점에 동시에 기동 된다고 가정한다. 또한 분석을 간단히 하기 위해서 네트워크를 경유하는 메시지로 인한 전송지연은 없다고 가정한다.

대상 시스템의 요구사양으로 제어루프에 대한 양극단 데드라인이 10ms이고 태스크들에 대한 수행시간(e)과 데드라인(d)이 표 1과 같이 주어졌을 때 요구사양을 만족하기 위한

태스크의 주기와 우선순위를 스케줄하면 다음과 같다. 각 노드에서 독립적으로 수행되는 주기 태스크의 주기는 데드라인과 같게 설정하고 비주기 태스크 경우, 주기를 알 수 없지만 일반적으로 사용되는 비주기 태스크의 최대 발생 빈도 기반의 가상주기를 사용한다면 최대 발생주기가 데드라인보다 짧을 수 없으므로 가상주기는 15ms가 된다. 선행제약을 갖는 제어루프 상의 주기 태스크들은 동기화를 위해 생산자 태스크와 소비자 태스크간에 주기 배수 관계를 갖도록 10ms로 주기를 설정한다. 모든 태스크의 주기가 설정된 후 일반적으로 널리 사용되는 RM(Rate Monotonic) 방법[8]을 이용하여 태스크의 우선순위를 부여하면 최종적인 시스템의 스케줄 결과 및 설정 값은 표 2와 같아진다. 표 1에서 제어루프에 포함된 주기 태스크의 데드라인이 주어지지 않은 것은 일반적으로 시스템 요구사양으로서 제어루프의 양극단 데드라인만이 주어지지 루프에 포함된 각 주기 태스크의 데드라인은 주어지지 않기 때문이다.

표 2의 스케줄 결과 기반으로 동작하는 태스크 타이밍도가 그림 2에 나타나 있다. 제어노드에서 가장 높은 우선순위를 갖는 주기 태스크 τ_{C1}^{P1} 보다 주기 태스크 τ_{C1}^{P2} 가 먼저 수행되는데 이는 센서노드의 주기 태스크 τ_{S1}^{P1} 로부터 메시지를 받을 때까지 주기 태스크 τ_{C1}^{P1} 가 기다려야 하는 선행제약으로 인해서 발생하는 현상이며 구동노드 경우도 마찬가지이다. 제어루프의 양극단 응답시간이 양극단 데드라인 10ms를 만족하며 비주기 태스크 및 다른 태스크들도 각각 데드라인을 만족하며 수행되므로 시스템이 스케줄 가능함을 알 수 있다. 이때 각 노드의 이용률은 센서노드가 63.3%, 제어노드는 66.7%, 구동노드는 63.3%가 된다.

만약 앞에서 사용된 시스템의 비주기 태스크간 선행제약이 생긴다면 현재의 이용률에서 RM 스케줄링 방법을 통해 시스템을 스케줄 할 수 없게 된다. 그림 1의 시스템에 비주기 태스크들간 선행제약을 추가한 모델이 그림 3에 나타나 있다. 이때 비주기 태스크를 기동 시킨 이벤트에 대한 양극단 데드라인은 15ms라고 가정하자. 이 경우 역시 비주기 태스크의 가상주기가 15ms로 설정되기 때문에 RM 방법으로 스케줄 하였을 경우 스케줄 결과는 앞선 결과 표 2와 같게 되며 이때의 시스템 타이밍도가 그림 4와 같이 바뀌게 된다. 이벤트 경로 상의 비주기 태스크들 간의 선행제약으로 인해 양극단 응답시간이 18ms가 되어 데드라인 15ms를 만족시키지 못하게 되었다. 이러한 결과는 비주기 태스크간 선행제약으로 인하여 같은 이용률을 갖고 있는 시스템임에도 불구하고 RM 방법으로는 시스템을 스케줄 할 수 없음을 나타내고 있다. 그림 3의 시스템 이용률을 낮추지 않고 시스템을 스케줄하기 위해서는 새로운 우선순위 할당 방법이 필요하다. 본 논문에서는 RM 방법과는 다른 양극단 데드라인에 대한 태스크 수행 여유도에 기반한 우선순위 할당 방법을 3절에서 제안한다. 양극단 여유도에 기반하여 15ms의 가상주기를 갖는 비주기 태스크들의 우선순위와 15ms보다 짧은 10ms의 주기를 갖지만 상대적으로 데드라인까지의 여유도가 큰, 각 노드에서 독립적으로 수행되는 주기 태스크들의 우선순위를 바꾸어 줌으로써 시스템을 스케줄 가능하게 만들 수 있다.

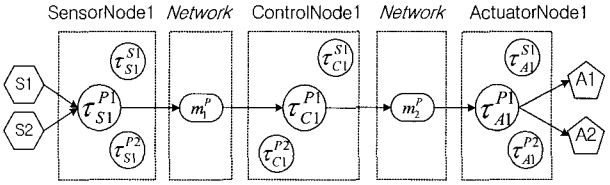


그림 1. 비주기 태스크 간 선행제약이 없는 모델.
Fig. 1. Model of sporadic tasks without precedence.

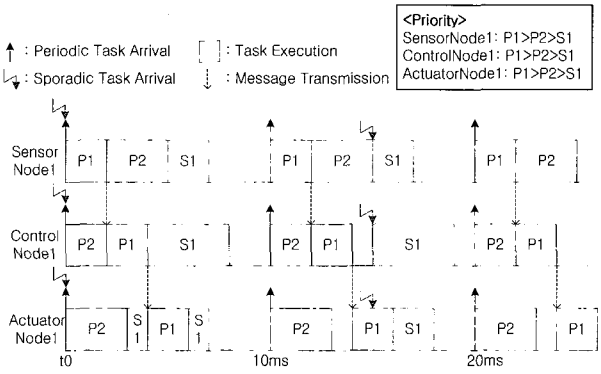


그림 2. 비주기 태스크간 선행제약이 없는 경우.
Fig. 2. Case of sporadic tasks without precedence.

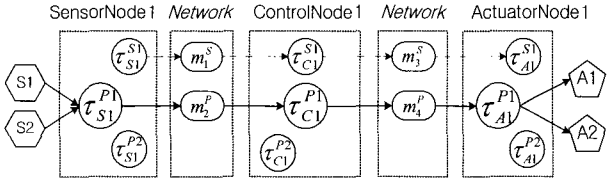


그림 3. 비주기 태스크 간 선행제약이 있는 모델.
Fig. 3. Model of sporadic tasks with precedence.

표 1. 시스템의 요구사양 및 설정 값.
Table 1. System's requirements.

Task	τ_{S1}^{P1}	τ_{S1}^{P2}	τ_{S1}^{S1}	τ_{C1}^{P1}	τ_{C1}^{P2}	τ_{C1}^{S1}	τ_{A1}^{P1}	τ_{A1}^{P2}	τ_{A1}^{S1}
e	2	3	2	2	2	4	2	3	2
d	x	10	15	x	10	15	x	10	15

표 2. RM방법을 통한 스케줄 결과.
Table 2. Schedule result by RM method.

Task	τ_{S1}^{P1}	τ_{S1}^{P2}	τ_{S1}^{S1}	τ_{C1}^{P1}	τ_{C1}^{P2}	τ_{C1}^{S1}	τ_{A1}^{P1}	τ_{A1}^{P2}	τ_{A1}^{S1}
e	2	3	2	2	2	4	2	3	2
d	10	10	15	10	10	15	10	10	15
T	10	10	15	10	10	15	10	10	15
p	1	2	3	1	2	3	1	2	3

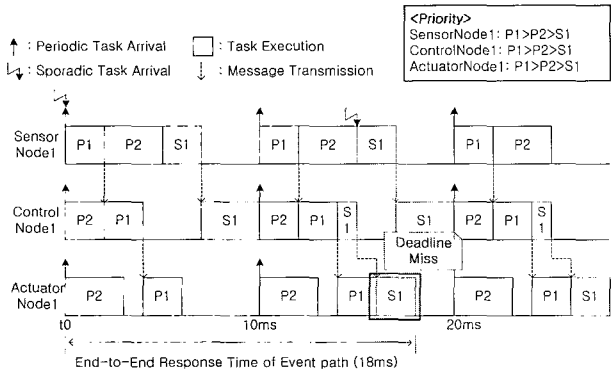


그림 4. 비주기 태스크간 선행제약이 있는 경우.
Fig. 4. Case of sporadic tasks with precedence.

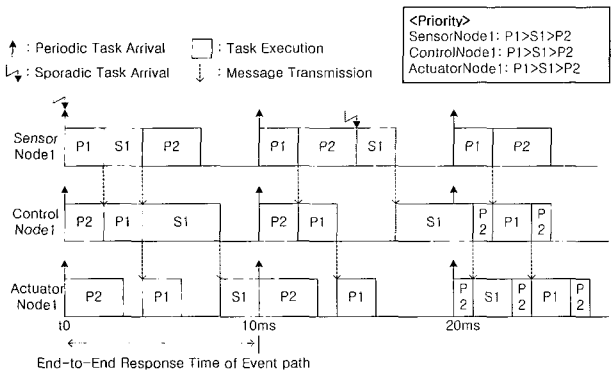


그림 5. 우선순위 조절 후 스케줄 결과.
Fig. 5. Schedule result after adjusting priority.

태스크간 우선순위 재조정 후의 시스템 타이밍도가 그림 5에 나타나 있으며 이러한 결과를 통해 양극단 여유도 기반의 우선순위 할당 방법이 RM 보다 더 높은 시스템 이용률을 보장할 수 있는 방법임을 알 수 있다. 또한 비주기 태스크간 선행제약을 없는 모델에 적용된 스케줄 방법이 비주기 태스크간 선행제약이 있는 모델에 적용될 경우 시스템을 스케줄하지 못할 수도 있음을 알 수 있다.

본 절에서 서술된 스케줄가능성 분석 및 시스템 타이밍 도는 네트워크를 경유한 메시지 전송지연 특성을 고려하지 않고 서술되었다. 만약 태스크간 메시지 전송지연을 고려하게 된다면 메시지의 지연으로 인해 비주기 태스크로 구성되는 이벤트 경로나 제어루프의 양극단 응답시간은 더 길어질 수 있으므로 시스템이 스케줄 불가능해질 수 있다. 그러므로 분산 실시간 시스템의 스케줄링 및 스케줄가능성 분석 시에 태스크들과 함께 메시지도 같이 고려해주어야 한다. 즉, 분산 실시간 시스템의 스케줄링 및 스케줄가능성 분석은 양극단에 걸쳐있는 태스크들과 메시지들의 제약사항을 모두 함께 고려하여야만 한다.

III. 양극단 제약을 고려한 스케줄링

1. 양극단 스케줄링 순서

본 절에서는 양극단 제약을 모두 함께 고려하여 분산 실시간 시스템을 스케줄하기 위한 전체 순서와 태스크와 메시지에 주기와 우선순위를 효율적으로 설정하는 방법을 제안한다. 본 논문에서 제안하는 전체 스케줄링 순서가 그림 6에 나타나 있으며 스케줄링 순서를 따라 논문을 서술해 나간다. 분산 실시간 시스템의 양극단 스케줄링을 위하여 첫 번째로 스케줄 해야 될 시스템에 대한 모델링을 태스크 그래프로 나타내고 각 태스크를 적절하게 노드에 할당하고 각 태스크와 메시지에 시스템 요구사양으로 주어지는 태스크 순수 수행시간 및 메시지 크기를 설정한다. 다음으로 양극단 스케줄가능성 검증 및 태스크와 메시지 주기 설정에 적용되는 제약식을 유도한다. 제약식으로는 각 제어루프와 이벤트 경로에 대해 양극단 응답시간제약, 태스크와 메시지간 또는 태스크간의 선행제약, 생산자 태스크와 소비자 태스크의 주기배수 관계를 나타내는 주기조화 제약을 유도한다. 유도된 제약식을 기반으로 태스크와 메시지의 주기와 우선순위를 설정한다.

태스크와 메시지에 주기와 우선순위가 설정되면 각 노드상의 모든 태스크들이 자신의 데드라인을 만족하면서 수행될 수 있는지, 모든 메시지가 데드라인 이내에 전송될 수 있는지, 그리고 양극단 응답시간을 만족하며 태스크와 메시지가 수행되는지를 검증하는 세 가지 스케줄가능성 분석이 수행된다. 각 스케줄가능성 분석에서 시스템이 스케줄 가능하지 않다면 주기설정 단계로 되돌아가 새로운 주기를 할당한다. 이러한 반복과정에서도 시스템을 스케줄 가능하게 만드는 태스크와 메시지의 주기와 우선순위 집합을 찾지 못한다면 처음 단계로 돌아가 시스템의 요구사양을 변경하여 리모델링을 수행하여야 한다.

Algorithm. End-to-End Scheduling Method
1) System Modeling
- Design the task graph of system and allocate tasks to nodes
- Set parameters of tasks and messages by system requirements
2) Derive End-to-End Constraints
- Derive End-to-End Constraints of control loops
- Derive End-to-End Constraints of event paths
3) Period Assignment
- Task's period: a binary search-based assignment algorithm
- Message's period: inherit from its producer task
4) Priority Assignment
- Task's priority: a laxity-based priority assignment algorithm
- Message's priority: a laxity-based priority assignment algorithm
5) Schedulability Analysis
- Node schedulability: if unschedulable then go to step 3
- Network schedulability: if unschedulable then go to step 3
- End-to-end schedulability: if unschedulable then go to step 3
6) Decide to Reschedule or Finish
- If the end-to-end period of all control loops can be rescheduled to be shorter, then go to step 3
- Set the scheduled results to the parameters of tasks and messages

그림 6. 양극단 스케줄링 방법의 순서도.
Fig. 6. Flow of end-to-end scheduling method.

만약 세 가지 스케줄가능성 분석을 통과한다면 이때 적용된 주기와 우선순위 집합을 저장하고 세 번째 단계로 돌아가 더 짧은 제어루프의 양극단 주기를 적용하여 시스템을 재스케줄 한다. 이러한 반복 과정을 통하여 세 가지 스케줄가능성을 만족하면서 제어루프가 가장 짧은 양극단 주기로 수행되는 태스크와 메시지의 주기와 우선순위 집합을 찾게 된다. 마지막으로 찾아진 태스크와 메시지의 주기와 우선순위 결과값을 태스크와 메시지의 파라미터로 설정하여 시스템을 구동한다.

2. 시스템 모델링

본 논문에서 제안된 스케줄링 방법이 적용될 시스템에 대한 태스크 그래프가 그림 7에 나타나 있다. 대상 시스템은 두 개의 제어루프와 두 개의 이벤트 경로를 구성하는 주기 및 비주기 태스크들과 메시지들, 제어루프와 이벤트 경로에 관계없이 수행되는 기타 주기 및 비주기 태스크들로 구성된다. 네트워크는 CAN을 사용하며 시스템을 구성하는 모든 노드는 글로벌 시간에 동기화되어 있고 에러[14]가 발생하지 않는다고 가정한다.

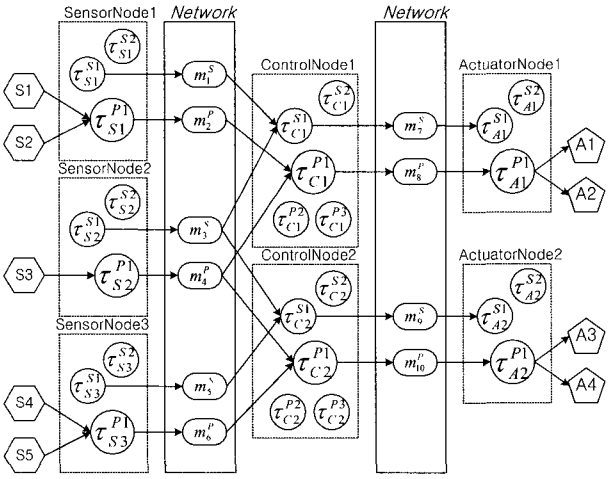


그림 7. 대상 시스템의 태스크 그래프.

Fig. 7. Task graph of example system.

표 3. 시스템 요구사양 및 설정 값.

Table 3. System requirements.

$MADT_1^{Evnt}=15ms$, $MADT_2^{Evnt}=15ms$

$MADT_1^{Ctrl}=50ms$, $MADT_2^{Ctrl}=70ms$

Task	*T*	*e*	*d*	*Task*	*T*	*e*	*d*	*Task*	*T*	*e*	*d*	*Msg.*	*Len*
τ_{S1}^{S1}		2		τ_{C1}^{S2}		4	20	τ_{A1}^{S2}		4	20	m_1^S	8
τ_{S1}^{S2}		4	20	τ_{C1}^{P1}		7		τ_{A1}^{P1}		2		m_2^P	4
τ_{S1}^{P1}		2		τ_{C1}^{P2}		5	50	τ_{A2}^{S1}		2		m_3^S	8
τ_{S2}^{S1}		2		τ_{C1}^{P3}		5	100	τ_{A2}^{S2}		4	20	m_4^P	2
τ_{S2}^{S2}		4	20	τ_{C2}^{S1}		3		τ_{A2}^{P1}		2		m_5^S	8
τ_{S2}^{P1}		2		τ_{C2}^{S2}		4	20					m_6^P	4
τ_{S3}^{S1}		2		τ_{C2}^{P1}		10						m_7^S	8
τ_{S3}^{S2}		4	20	τ_{C2}^{P2}		5	50					m_8^P	4
τ_{S3}^{P1}		2		τ_{C2}^{P3}		5	100					m_9^S	8
τ_{C1}^{S1}		3		τ_{A1}^{S1}		2						m_{10}^P	4

모든 노드는 고정 우선순위의 선점형 스케줄러 기반이고 선행제약을 갖는 제어루프와 이벤트 경로의 주기태스크와 비주기 태스크를 제외한 모든 태스크들은 서로 공유하는 자원 없이 독립적으로 수행된다고 가정한다. 모든 태스크들은 스케줄링 통해 설정되는 초기 시작시간(initial phase time)에 스케줄러에 의해서 기동 된다. 대상 시스템의 요구사양으로 두 개의 제어루프와 두 개의 이벤트 경로에 대한 양극단 데드라인과 각 태스크들의 설정 값들이 표 3에 나타나 있다. 음영으로 표시된 빈칸은 주어진 요구사양을 만족시킬 수 있도록 스케줄링을 통해 얻어야 되는 값들이다.

3. 양극단 제약식 유도

시스템 모델을 태스크 그래프로 나타낸 후 양극단 스케줄가능성 분석을 위한 태스크와 메시지간 선행 제약과 양극단 응답시간 제약을 유도한다. 또한 제어루프를 구성하는 주기 태스크 간 동기화를 위한 주기조화 제약을 유도한다. 우선 두 개의 이벤트 경로에 존재하는 비주기 태스크와 메시지에 대한 선행제약은 다음과 같다.

$$\phi_{\tau_{C1}^{S1}} \geq \max(\phi_{\tau_{S1}^{S1}} + d_{\tau_{S1}^{S1}} + d_{m_1^S}, \phi_{\tau_{S2}^{S1}} + d_{\tau_{S2}^{S1}} + d_{m_3^S})$$

$$\phi_{\tau_{A1}^{S1}} \geq \phi_{\tau_{C1}^{S1}} + d_{\tau_{C1}^{S1}} + d_{m_7^S}$$

$$\phi_{\tau_{C2}^{S1}} \geq \max(\phi_{\tau_{S2}^{S1}} + d_{\tau_{S2}^{S1}} + d_{m_3^S}, \phi_{\tau_{S3}^{S1}} + d_{\tau_{S3}^{S1}} + d_{m_5^S})$$

$$\phi_{\tau_{A2}^{S1}} \geq \phi_{\tau_{C2}^{S1}} + d_{\tau_{C2}^{S1}} + d_{m_9^S}$$

두 개의 제어루프 상의 주기 태스크와 메시지에 대한 선행제약은 다음과 같다.

$$\phi_{\tau_{C1}^{P1}} \geq \max(\phi_{\tau_{S1}^{P1}} + d_{\tau_{S1}^{P1}} + d_{m_2^P}, \phi_{\tau_{S2}^{P1}} + d_{\tau_{S2}^{P1}} + d_{m_4^P})$$

$$\phi_{\tau_{A1}^{P1}} \geq \phi_{\tau_{C1}^{P1}} + d_{\tau_{C1}^{P1}} + d_{m_8^P}$$

$$\phi_{\tau_{C2}^{P1}} \geq \max(\phi_{\tau_{S2}^{P1}} + d_{\tau_{S2}^{P1}} + d_{m_4^P}, \phi_{\tau_{S3}^{P1}} + d_{\tau_{S3}^{P1}} + d_{m_6^P})$$

$$\phi_{\tau_{A2}^{P1}} \geq \phi_{\tau_{C2}^{P1}} + d_{\tau_{C2}^{P1}} + d_{m_{10}^P}$$

두 개의 이벤트 경로에 대한 양극단 응답시간 제약은 아래와 같다.

$$\phi_{\tau_{A1}^{S1}} + d_{\tau_{A1}^{S1}} - \min(\phi_{\tau_{S1}^{S1}}, \phi_{\tau_{S2}^{S1}}) \leq MADT_1^{Evnt.}$$

$$\phi_{\tau_{A2}^{S1}} + d_{\tau_{A2}^{S1}} - \min(\phi_{\tau_{S2}^{S1}}, \phi_{\tau_{S3}^{S1}}) \leq MADT_2^{Evnt.}$$

두 개의 제어루프에 대한 양극단 응답시간 제약은 다음과 같다.

$$\phi_{\tau_{A1}^{P1}} + d_{\tau_{A1}^{P1}} - \min(\phi_{\tau_{S1}^{P1}}, \phi_{\tau_{S2}^{P1}}) \leq MADT_1^{Ctrl.}$$

$$\phi_{\tau_{A2}^{P1}} + d_{\tau_{A2}^{P1}} - \min(\phi_{\tau_{S2}^{P1}}, \phi_{\tau_{S3}^{P1}}) \leq MADT_2^{Ctrl.}$$

주기 태스크 간의 주기 조화 제약은 다음과 같이 유도되며 생산자 태스크와 메시지는 항상 1대1의 관계이므로 메시지의 생산자 태스크의 주기와 같게 된다. 여기서 'A|B'는 B가

A의 배수임을 나타낸다.

$$T_{\tau_{S1}^{P1}} = T_{m_2^P} \mid T_{\tau_{C1}^{P1}} = T_{m_8^P} = T_{\tau_{A1}^{P1}}, \quad T_{\tau_{S2}^{P1}} = T_{m_4^P} \mid T_{\tau_{C1}^{P1}} = T_{m_8^P} = T_{\tau_{A1}^{P1}},$$
$$T_{\tau_{S2}^{P1}} = T_{m_4^P} \mid T_{\tau_{C2}^{P1}} = T_{m_{10}^P} = T_{\tau_{A2}^{P1}}, \quad T_{\tau_{S3}^{P1}} = T_{m_6^P} \mid T_{\tau_{C2}^{P1}} = T_{m_{10}^P} = T_{\tau_{A2}^{P1}}$$

4. 태스크와 메시지 주기설정

제약식 유도 후 태스크와 메시지에 주기를 설정한다. 태스크와 메시지에 대한 주기조화 제약을 태스크와 메시지에 설정할 주기에 대한 규칙으로 사용한다. 우선 태스크의 주기설정으로 비주기 태스크와 주기 태스크의 주기 설정이 있다. 비주기 태스크 경우 태스크의 주기가 존재하지 않으므로 그 주기를 설정할 수 없지만 시스템에 가해질 수 있는 최대 부하를 고려하여 시스템의 스케줄가능성을 분석하므로 비주기 태스크가 기동 될 수 있는 최대 빈도 수를 기준으로 비주기 태스크의 가상주기를 설정하는 방법이 일반적이다. 만약 비주기 태스크가 10ms의 데드라인을 가지고 있다면 태스크 기동 후 10ms 이내에는 다시 기동 되지 않는다는 가정 하에서 이 비주기 태스크의 가상 주기를 10ms로 설정하게 된다. 이벤트 경로에 포함되지 않고 독립적으로 수행되는 비주기 태스크들은 위의 규칙을 적용하는데 문제가 없다. 이벤트 경로에 포함되어 서로 선행제약을 갖는 비주기 태스크들의 경우, 주어지는 요구사양이 양극단 데드라인이기 때문에 비주기 태스크 각각의 데드라인은 알 수가 없다. 그러나 이벤트 경로에 포함되어 있는 모든 비주기 태스크들은 해당 이벤트가 발생할 때마다 순차적으로 기동하므로 이벤트의 최대 발생 빈도를 각각의 비주기 태스크 기동 빈도로 보면 된다. 그러므로 해당 이벤트의 데드라인을 각각의 가상주기로 설정하면 된다. 주기 태스크의 경우, 제어루프와 관계없이 독립적으로 수행되는 주기태스크의 주기는 요구사양으로 주어진 데드라인을 만족하는 주기로 설정한다. 가령, 데드라인이 50ms이라면 주기도 50ms로 설정한다. 제어루프에 포함되어 선행제약을 갖는 주기 태스크의 경우, 주어지는 요구사양은 제어루프의 양극단 데드라인뿐이며 각 주기 태스크의 데드라인은 주어지지 안는다.

또한 발생한 이벤트에 의해 순차적으로 기동 되어 서로 동기화되는 비주기 태스크와는 다르게 주기 태스크들은 생산자 태스크와 소비자 태스크 주기 간에 주기배수 관계를 통하여 동기화 시키게 된다. 이는 다수의 센서로부터의 데이터 동기화 및 두 개 이상의 제어루프가 중첩되었을 경우 주기 태스크간 동기화를 위한 주기설정 제약이다. 그러므로 이벤트 경로 상의 비주기 태스크처럼 제어루프의 양극단 데드라인을 각각의 주기 태스크의 주기로 설정할 수가 없다.

그림 7의 대상 시스템도 3.3절의 주기조화 제약에서와 같이 ControlNode1과 ControlNode2의 제어태스크 τ_{C1}^{P1}와 τ_{C2}^{P1}의 주기는 각각, 센서노드의 τ_{S1}^{P1}와 τ_{S2}^{P1}에, τ_{S2}^{P1}와 τ_{S3}^{P1}의 주기에 배수관계이어야 한다. 또한 Sensor Node2의 샘플링 태스크 τ_{S2}^{P1}의 주기는 제어노드의 τ_{C1}^{P1}와 τ_{C2}^{P1}의 주기에 최대 공약수로 설정되어야 한다. 이러한 주기조화 제약을 만족하면서 시스템의 양극단 제약사양을 만족하는 주기를 찾는 것은 스케줄링에서 중요한 문제 중의 하나이다. 본 논문에서는 주기 태스크의 주기를 [6]에서 제안된 이진탐색 알고리즘에 기반한 주기설정 알고리즘을 이용하여 설정한다. 제안된 알고리즘은 복수 개의 센서노드, 제어노드, 구동노드로 구성되는 제어루프와 이러한 제어루프가 서로 중첩되어 있는 시스템에 적용 가능하다. 또한 제어루프의 양극단 데드라인과 주기 태스크의 주기조화 제약을 만족하면서 가장 짧아질 수 있는 제어루프의 양극단 주기를 찾는 방법이다. 메시지의 주기 설정의 경우, 메시지 생산 및 전송은 생산자 태스크의 주기에 맞추어 수행되므로 메시지의 주기는 생산자 태스크의 주기를 상속 받아 설정한다.

5. 태스크와 메시지 우선순위 설정

본 논문에서 고려하고 있는 시스템은 선점형 고정 우선순위 기반 시스템이므로 off-line 스케줄링 시에 태스크와 메시지에 적절한 우선순위를 부여하여 시스템을 가동하게 된다.

선점형 고정 우선순위 기반 시스템에서 우선순위 설정 방법으로는 태스크의 주기와 데드라인이 같다라는 조건 하에 태스크의 주기에 비례하여 우선순위를 설정하는 RM(Rate Monotonic) 방법과 태스크의 데드라인이 주기와 같지 않다라는 조건 하에 태스크의 데드라인에 비례하게 우선순위를 설정하는 DM(Deadline Monotonic) 방법[1]이 있다.

우선 RM 방법의 경우 2절에서 언급한 바와 같이 선행제약을 갖는 태스크들로 이루어진 시스템에서는 최적의 스케줄 결과를 얻지 못할 수도 있다. 예를 들어 그림 7의 대상 모델에서 SensorNode1의 비주기 태스크 τ_{S1}^{S1}, τ_{S1}^{S2}의 주기는 15ms, 20ms가 되고 주기 할당 알고리즘에 의해 주기 태스크 τ_{S1}^{P1}의 주기와 해당 제어루프의 양극단 주기가 30ms로 설정되었다면 RM 방법에 의한 우선순위 설정은 주기에 비례하므로 τ_{S1}^{S1}, τ_{S1}^{S2}, τ_{S1}^{P1} 순으로 높은 우선순위를 할당하게 된다.

그러나 효율적인 우선순위 할당을 위해 태스크 수행 여유도를 살펴보면 비주기 태스크 τ_{S1}^{S2}는 로컬 노드 내에서 독립적으로 수행을 완료하고 주기가 20ms, 수행시간이 4ms 이므로 데드라인까지의 여유시간이 16ms가 된다. 반면에 태스크 τ_{S1}^{S1}와 τ_{S1}^{P1}는 각각 이벤트 경로와 제어루프에 포함되므로 단순히 데드라인까지의 여유시간을 주기에서 수행시간을 뺀 것으로 생각할 수가 없다. 왜냐하면 비주기 태스크 τ_{S1}^{S1}의 경우 주기가 15ms이지만 알람과 같은 이벤트로 인한 기동시점부터 해당 알람이 시스템 내부에서 처리완료가 될 때까지의 데드라인이 15ms이기 때문이다. 그러므로 이벤트 경로 상의 모든 태스크들의 수행시간과 모든 메시지의 전송시간을 양극단 데드라인에서 빼고 난 나머지 시간을 해당 태스크들과 메시지들에 적절하게 분배하여 얻어지는 값을 비주기 태스크 τ_{S1}^{S1}의 여유시간으로 보아야 한다. 결과적으로 비주기 태스크 τ_{S1}^{S1}의 여유도는 주기 15ms에서 수행시간 2ms를 뺀 13ms가 아니라 상당히 짧은 시간이 된다. 또한 주기 태스크 τ_{S1}^{P1}의 경우, 양극단 제어루프의 주기가 30ms 이므로 해당 제어루프의 모든 태스크 수행과 메시지 전송이 30ms 이내에 완료되어야 하므로 주기태스크 τ_{S1}^{P1}의 여유도도 위와 마찬가

지로 상당히 짧아지게 된다. 이때 만약 SensorNode1의 주기 태스크 τ_{S1}^{P1}의 여유도가 비주기 태스크 τ_{S1}^{S2}의 여유도보다 작다면, 주기가 더 길더라도 주기 태스크 τ_{S1}^{P1}의 우선순위를 비주기 태스크 τ_{S1}^{S2}보다 더 높게 설정하는 것이 더 적절하다.

태스크의 데드라인이 주기와 같지 않다는 조건을 통해 데드라인에 비례하여 우선순위를 설정하는 DM 방법은 위에서 언급된 RM의 비효율성을 제거할 수 있으나 태스크에 적절한 데드라인을 설정해야 되는 추가적인 과정이 필요하다. 로컬 노드에서 선행제약 없이 수행되는 태스크의 경우 주기를 데드라인으로 설정하면 되지만 이벤트 경로나 제어루프에 포함되어 수행되는 태스크의 경우 적절한 데드라인을 찾는 것은 풀어야 될 또 다른 문제이다. 왜냐하면 시스템 요구사양으로 주어지는 것은 단지 이벤트 경로와 제어루프의 데드라인인 양극단 시간제약이기 때문이며 해당 데드라인을 만족할 수 있도록 태스크에 적절한 주기와 데드라인을 설정하는 것은 시스템 설계자가 수행하여야 하기 때문이다.

본 논문에서는 태스크와 메시지 우선순위 설정을 위한 양극단 여유도 기반의 우선순위 할당 방법을 제안한다. 태스크 우선순위 할당 알고리즘이 그림 8에 나타나 있으며 제안된 알고리즘은 모든 태스크에 여유도를 계산하여 여유도가 작을수록 높은 우선순위를 할당하게 된다. 이벤트 경로나 제어루프에 포함되지 않는 태스크의 경우, 자신의 주기에 수행시간을 뺀 나머지 시간을 여유도로 할당한다.

/* Φ : the task set including all tasks in the system */
/* E_j : the set including all tasks and messages in the j-th event path */
/* X_j : the set including all tasks and message in the j-th control loop */
/* Ψ_k : the task set including all tasks in the k-th node */
/* lax_i : the laxity of τ_i */
/* $N(x)$: the number of all tasks and messages included in a set x */
/* $EventPathID(x)$: the event path's index including a task x */
/* $ControlLoopID(x)$: the control loop's index including a task x */

foreach $\forall \tau_i \in \Phi$ do
  if τ_i is included in a control loop or a event path
  then if τ_i is a sporadic task then $j := EventPathID(\tau_i)$

$$lax_i := \left(MADT_j^{Evnt} - \sum_{\forall \tau_k \in E_j} e_i - \sum_{\forall m_k \in E_j} C_i \right) \Big/ N(E_j)$$

  else $j := ControlLoopID(\tau_i)$

$$lax_i := \left(MADT_j^{Ctrl} - \sum_{\forall \tau_k \in X_j} e_i - \sum_{\forall m_k \in X_j} C_i \right) \Big/ N(X_j)$$

  else $lax_i := T_i - e_i$
end

foreach $\forall k$ do
  Set the priority of $\forall \tau_i \in \Psi_k$ according to lax_i
end

그림 8. Laxity 기반 태스크 우선순위 설정 알고리즘.
Fig. 8. Laxity-based task's priority assignment algorithm.

이벤트 경로나 제어루프에 포함되어 수행되는 태스크의 경우는 모든 태스크의 수행시간과 메시지의 전송시간을 양극단 데드라인에서 빼고 이를 다시 태스크와 메시지의 수로 나누어 각각의 태스크 여유도를 할당한다. 모든 태스크에 여유도가 할당되면 이를 기반으로 노드 별로 여유도가 가장 작은 태스크 순으로 높은 우선순위를 할당한다. 메시지의 우선순위 할당은 생산자 태스크의 여유도를 상속 받아서 태스크와 마찬가지로 우선순위를 설정한다.

6. 스케줄가능성 분석

분산 실시간 시스템의 모든 태스크와 메시지에 주기와 우선순위가 설정되면 스케줄된 주기와 우선순위로 시스템이 스케줄 가능한지를 검사한다. 스케줄 가능성 검사는 각 노드에서 모든 태스크가 데드라인을 만족하며 수행되는지를 분석하는 노드 스케줄가능성 분석과 네트워크의 모든 메시지가 데드라인 이내에 전송되는지를 분석하는 네트워크 스케줄가능성 분석, 이벤트 경로나 제어루프가 양극단 시간제약을 만족하면 수행되는지 분석하는 양극단 스케줄가능성 분석을 통해 수행된다. 각 노드가 스케줄 가능한지를 검사하기 위해 노드 별로 모든 태스크의 최악응답시간을 계산하여 최악의 경우에 태스크의 수행완료가 자신의 주기 이내에 끝나는지 검사한다. 만약, 모든 태스크의 주기 이내 수행 완료를 보장하지 못한다면 해당 노드는 스케줄 불가능으로 판정되고 이는 결국 시스템 전체를 스케줄 불가능하게 만들 수 있으므로 그림 6의 step3로 돌아가 다른 주기 집합을 찾는 재스케줄링을 수행한다. 네트워크 스케줄가능성 분석도 태스크와 마찬가지로 모든 메시지의 최악 응답시간을 계산하여 메시지가 최악의 경우에도 주기 이내에 전송이 완료되는지를 검사하여 네트워크의 스케줄가능성을 분석한다. 만약 하나의 메시지라도 주기 이내의 전송을 보장하지 못한다면 스케줄 불가능하므로 그림 6의 step3로 돌아가 역시 재스케줄을 수행한다. 태스크와 메시지의 최악응답시간은 (1)과 (2)을 통해 계산된다. 로컬 노드에서 고정된 우선 순위를 사용하는 선점형 시스템의 경우 태스크 τ_i의 최악응답시간 R_{τ_i}는 다음과 같다[9].

$$R_{\tau_i} = B_{\tau_i} + \sum_{\forall \tau_j \in hp(\tau_i)} \left\lceil \frac{R_{\tau_i}}{T_{\tau_j}} \right\rceil \times e_{\tau_j} + e_{\tau_i} \quad (1)$$

B_{τ_i}는 태스크 τ_i가 자신보다 낮은 우선 순위를 갖는 태스크로 인해 블로킹되는 최대 시간이며 $hp(\tau_i)$는 태스크 τ_i보다 높은 우선 순위를 갖는 태스크들의 집합이다. CAN 상에서 메시지 m_i의 최악응답시간 R_{m_i}는 다음과 같다[10].

$$R_{m_i} = B_{m_i} + \sum_{\forall m_j \in hp(m_i)} \left\lceil \frac{R_{m_i}}{T_{m_j}} \right\rceil \times C_{m_j} + C_{m_i} \quad (2)$$

B_{m_i}는 메시지 m_i가 자신보다 낮은 우선 순위를 갖는 메시지에 의해 지연되는 최대 시간이고 $hp(m_i)$는 메시지 m_i보다 높은 우선 순위 갖는 메시지 집합이다.

양극단 스케줄가능성 분석은 3.3절에서 유도된 태스크와 메시지의 선행 제약식과 양극단 시간제약을 품으로써 수행한다. 태스크와 메시지간 선행제약을 풀기 위해서 데드라인을 설정하여야 하는데 앞에서 언급한 바와 같이 제어루프와 이벤트 경로 상의 비주기, 주기 태스크와 비주기, 주기 메시지의 데드라인이 요구사양으로 주어지지 않기 때문에 적절한 데드라인을 설정하여야 한다. 노드 스케줄가능성과 네트워크 스케줄가능성 분석 시에는 태스크나 메시지가 주기 이내에 수행 또는 전송이 완료되는 것을 보장한다면 노드와 네트워크 입장에서는 스케줄 가능하기 때문에 주기를 데드라인으로 보고 스케줄가능성 분석을 수행하였다. 그러나 양극단 스케줄 가능성 분석 시에 주기를 데드라인으로 설정하여 분석한다면 양극단 응답시간을 길어지게 만드는 문제를 초래하게 된다. 왜냐하면 양극단 스케줄 가능성 분석 시에 생산자 태스크의 데드라인 시점을 소비자 태스크의 기동시점으로 설정하기 때문이다. 태스크나 메시지의 수행 또는 전송완료가 항상 보장되는 주기보다 짧은 시간을 알 수 있음에도 불구하고 주기를 데드라인으로 설정하게 되면 소비자 태스크를 불필요한 시간 동안 일시 정지(suspend) 시키는 결과를 초래한다. 결과적으로 이러한 불필요한 대기시간은 전체 제어루프의 양극단 응답시간을 길어지게 만드는 결과를 초래한다. 그러므로 태스크와 메시지에 적절한 데드라인을 설정해야 한다. 태스크와 메시지의 최악응답시간은 최악의 경우를 고려한 태스크 수행완료와 메시지 전송완료 시간이므로 항상 태스크와 메시지는 최악응답시간 이내에 수행 및 전송이 완료된다고 볼 수 있다. 즉, $R_{\tau_i} \le d_{\tau_i} \le T_{\tau_i}$ 와 $R_{m_i} \le d_{m_i} \le T_{m_i}$ 의 관계가 성립되므로 태스크와 메시지의 데드라인을 최악응답시간으로 설정한다. 이러한 방법으로 태스크와 메시지의 데드라인을 설정하면 불필요한 대기시간을 줄이게 되고 결과적으로 양극단 응답시간을 짧게 만들 수 있는 장점이 있다.

태스크와 메시지의 데드라인을 설정하고 선행 제약식을 풀면 양극단 시간제약이 만족되는지 검사할 수 있다. 만약 양극단 응답시간 제약이 만족되지 않는다면 그림 6의 step3로 돌아가서 새로운 주기 집합을 찾아 재스케줄링을 수행한다. 이러한 반복과정에서도 시스템을 스케줄 가능하게 만드는 태스크와 메시지의 주기와 우선순위 집합을 찾지 못한다면 처음 단계로 돌아가 시스템의 요구사양을 변경하여 리모델링을 수행하여야 한다. 세 가지 스케줄가능성 분석을 통과한 경우라면, 이때 적용된 주기와 우선순위 집합을 저장하고 더 짧은 제어루프의 양극단 주기를 찾을 수 있는 지 검사한다. 만약, 바로 이전에 스케줄이 성공한 제어루프의 양극단 주기가 현재 제어루프의 양극단 주기와 같다면 더 이상 주기가 짧아질 수 없다고 판단하여 스케줄을 마무리하며, 같지 않다면 그림 6의 step3로 돌아가 제어루프의 양극단 주기를 더 짧게 적용하여 시스템을 재스케줄 한다. 이러한 반복 과정을 통하여 세 가지 스케줄가능성을 만족하면서 제어루프의 양극단 주기가 가장 짧게 수행될 수 있는 주기와 우선순위 집합을 얻게 된다. 마지막으로 찾아진 태스크와 메시지의 주기와 우선순위 결과값을 시스템의 태스크와 메시지의 파라미터로 설정하여 시스템을 운영함으로써 양극단 제약을 고려한 스케줄링이 마무리된다.

7. 스케줄링 결과

본 논문에서 제안된 양극단 제약을 갖는 비주기, 주기 태스크의 스케줄링 방법을 그림 7에 적용한 결과가 그림 9에서부터 그림 12에 나타나 있다. 그림 9와 그림 10은 CAN의 속도가 500kbps일 경우 시스템의 스케줄 결과로서 그림 9는 반복적인 스케줄링 시에 설정되는 이벤트 경로와 제어루프의 양극단 주기와 그에 따른 양극단 응답시간이 나타나 있다. 4번의 반복 스케줄을 통하여 제어루프 1과 2는 양극단 주기가 25ms와 30ms로 설정되었으며 이때의 양극단 응답시간은 22ms와 27ms로 데드라인을 만족시키게 되었다. 이벤트 경로 1과 2의 양극단 데드라인은 15ms로 고정되었으며 양극단 응답시간은 각각 10ms와 11ms로 스케줄 되었다. 그림 10은 반복적인 스케줄링 시에 각 노드와 네트워크의 이용률을 나타내고 있다.

최종적인 스케줄 결과 시에 SensorNode1, ControlNode1, ControlNode2의 이용률은 각각 73%, 83%, 88%로 매우 높게 나타났다. 이러한 결과는 제어루프의 양극단 주기를 가능한 짧게 하기 위해 제어루프 상의 주기 태스크들의 주기를 짧게 설정하였기 때문에 나타나는 결과이다. 제안된 스케줄링 방법은 선행제약을 갖는 비주기 태스크들의 스케줄링과 동시에 제어루프의 성능을 위해 양극단 주기를 가능한 짧게 설정하는데 있다. 결과적으로 시스템의 이용률을 높게 유지시키면서 시스템을 스케줄 하게 된다. 다른 스케줄링 방법과 비교해 보면, RM 스케줄링 방법의 시스템 이용률 한도는 태스크 수가 m이라면 $m(2^{1/m}-1)$이 된다(m이 무한대로 가면 69.3%로 수렴)[8]. 그러므로 3개와 6개의 태스크가 존재하는 노드의 이용률 한도는 각각 78%와 72%되며 본 논문에서 제안한 스케줄링 방법이 선행제약이 존재하는 조건에도 불구하고 더 높은 시스템 이용률을 보장할 수 있음을 나타내고 있다.

그림 11과 그림 12는 CAN의 속도가 250kbps일 경우 시스템의 스케줄 결과를 나타내고 있다. 그림 9와 마찬가지로 그림 11은 반복적인 스케줄링 시에 설정되는 이벤트 경로와 제어루프의 양극단 주기와 그에 따른 양극단 응답시간이 나타나 있으며 4번의 반복 스케줄을 통하여 제어루프 1과 2는 양극단 주기가 30ms와 35ms로 설정되었으며 이때의 양극단 응답시간은 26ms와 31ms로 나타났다. 이벤트 경로1과 2의 양극단 데드라인은 역시 15ms로 고정이며 양극단 응답시간은 각각 12ms와 13ms로 스케줄 되었다. 250kbps에서 스케줄링 시의 각 노드와 네트워크의 이용률이 그림 12에 나타나 있으며 최종적인 스케줄 결과에서 SensorNode1, ControlNode1, ControlNode2의 이용률은 각각 73%, 78%, 84%로 나타났다. CAN의 속도가 500kbps일 때 보다 250kbps에서 ControlNode1과 ControlNode2의 이용률이 작아진 이유는 네트워크의 속도가 느려짐에 따라 메시지로 인한 전송지연 시간이 증가하게 되어 제어루프의 양극단 응답시간이 길어져 적용 가능한 제어루프의 양극단 주기가 길어졌고 결과적으로 태스크들의 주기도 길어지게 되어 상대적으로 노드의 이용률이 감소하게 되었다.

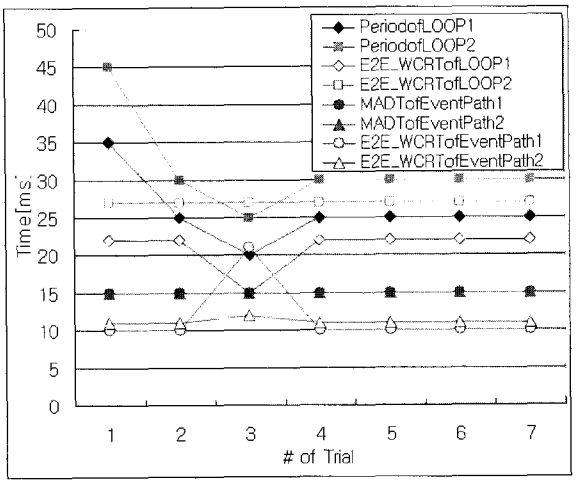


그림 9. 양극단 주기와 응답시간(500kbps).

Fig. 9. End-to-end period and response times(500kbps).

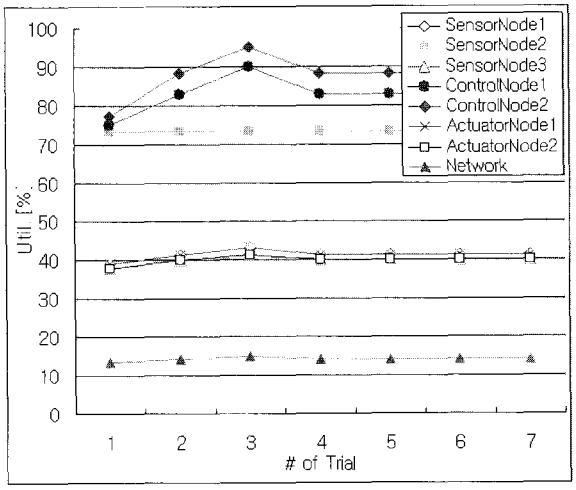


그림 10. 노드와 네트워크 이용률 변화(500kbps).

Fig. 10. Utilization of nodes and network(500kbps).

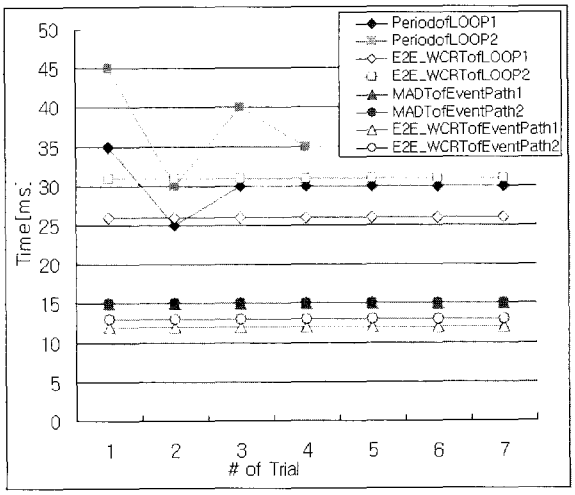


그림 11. 양극단 주기와 응답시간(250kbps).

Fig. 11. End-to-end period and response times(250kbps).

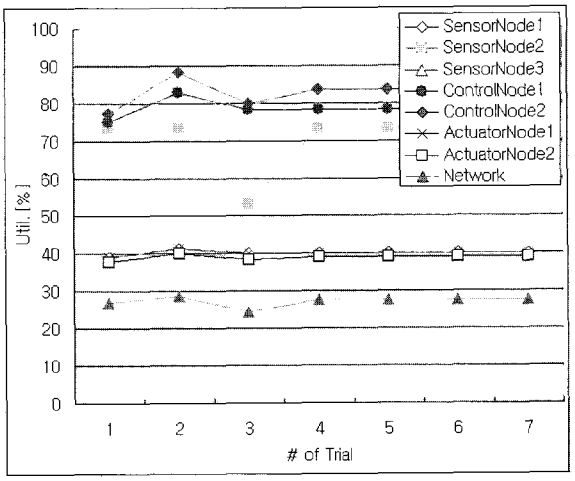


그림 12. 노드와 네트워크 이용률 변화(250kbps).

Fig. 12. Utilization of nodes and network(500kbps).

표 4. 태스크 스케줄링 결과(250kbps).

Table 4. Task schedule result(250kbps).

Task	Node	p	T	e	R	d	ϕ	Task	Node	p	T	e	R	d	ϕ
τ_{S1}^{S1}	1	1	15	2	2	2	0	τ_{C2}^{S1}	5	1	15	3	3	3	5
τ_{S1}^{S2}		3	20	4	8	20	0	τ_{C2}^{S2}		3	20	4	20	20	0
τ_{S1}^{P1}		2	30	2	4	4	0	τ_{C2}^{P1}		2	35	10	13	13	9
τ_{S2}^{S1}	2	1	15	2	2	2	0	τ_{C2}^{P2}		4	50	5	29	50	0
τ_{S2}^{S2}		3	20	4	12	20	0	τ_{C2}^{P3}		5	100	5	59	100	0
τ_{S2}^{P1}		2	5	2	4	4	0	τ_{A1}^{S1}	6	1	15	2	2	2	10
τ_{S3}^{S1}	3	1	15	2	2	2	0	τ_{A1}^{S2}		3	20	4	8	20	0
τ_{S3}^{S2}		3	20	4	8	20	0	τ_{A1}^{P1}		2	30	2	4	4	22
τ_{S3}^{P1}		2	35	2	4	4	0	τ_{A2}^{S1}	7	1	15	2	2	2	11
τ_{C1}^{S1}	4	1	15	3	3	3	4	τ_{A2}^{S2}		3	20	4	8	20	0
τ_{C1}^{S2}		3	20	4	14	20	0	τ_{A2}^{P1}		2	35	2	4	4	27
τ_{C1}^{P1}		2	30	7	10	10	8								
τ_{C1}^{P2}		4	50	5	26	50	0								
τ_{C1}^{P3}		5	100	5	48	100	0								

표 5. 메시지 스케줄링 결과(250kbps).

Table 5. Message schedule result(250kbps).

Msg.	p	T	L	C	R	d	ϕ
m_1^S	2	15	8	0.52	1.56	2	2
m_2^P	7	25	4	0.37	3.63	4	4
m_3^S	1	15	8	0.52	1.04	2	2
m_4^P	6	5	2	0.29	3.26	4	4
m_5^S	3	15	8	0.52	2.08	3	2
m_6^P	9	30	4	0.37	4.36	5	4
m_7^S	4	15	8	0.52	2.6	3	7
m_8^P	8	25	4	0.37	4	4	18
m_9^S	5	15	8	0.52	2.97	3	8
m_{10}^P	10	30	4	0.37	4.36	5	22

SensorNode1의 경우 CAN의 속도가 500kbps와 250kbps에서 이용률의 변화가 없는 이유는 주기 태스크 τ_{S2}^{P1}의 주기가 두 제어루프의 최대 공약수인 5ms로 설정되어 결과적으로 노드 이용률의 변화가 없게 되었다.

CAN의 속도를 125kbsp로 낮추게 되면 양극단 데드라인에 대한 여유도가 충분한 제어루프는 스케줄이 가능하지만 네트워크 이용률이 47%까지 올라감으로 인한 메시지 전송 지연으로 인해 양극단 데드라인에 대한 여유도가 매우 적은 이벤트 경로의 경우 스케줄 가능한 태스크와 메시지의 주기와 우선순위 집합을 찾을 수가 없었다. 그러므로 시스템의 요구사양 인 이벤트 경로의 양극단 데드라인을 늘려주거나 네트워크의 속도를 올려주어야만 한다. 이러한 결과는 메시지의 전송지연이 분산 실시간 시스템의 스케줄가능성에 매우 큰 영향을 미침을 나타내고 있다.

본 논문에서 제안된 양극단 스케줄링의 결과로서 CAN의 속도가 250kbps일 때 시스템의 모든 태스크에 대한 우선순위, 주기, 수행시간, 최악응답시간, 데드라인, 초기시작시간이 표 4에 나타나 있으며 모든 메시지에 대한 우선순위, 주기, 메시지 길이, 전송시간, 최악응답시간, 데드라인, 초기 시작시간이 표 5에 나타나 있다. 흰색 부분은 본 논문에서 스케줄링을 수행하기 전에 시스템 요구사양으로 주어지는 값이며 음영 부분은 제안된 스케줄링 방법에 의해 스케줄 된 태스크와 메시지의 파라미터 값이다.

IV. 결론

분산 실시간 시스템에는 비주기 태스크, 주기 태스크, 비실시간 태스크와 같이 다양한 태스크들과 이러한 태스크간 데이터 교환을 위한 비주기, 주기, 비실시간 메시지들이 존재하게 되므로 시스템의 요구사양을 만족하도록 이러한 태스크들과 메시지들을 스케줄하는 것은 매우 어려운 문제이다. 또한 분산 실시간 시스템을 구성하고 있는 노드와 네트워크에 대한 스케줄가능성뿐 아니라 시스템에 주어지는 양극단 제약을 보장하기 위해서 양극단 스케줄가능성 또한 만족되도록 스케줄 되어야 한다.

본 논문에서는 비주기, 주기 태스크와 비주기, 주기 메시지가 동시에 존재하는 분산 실시간 시스템에 대해서 노드와 네트워크 스케줄가능성뿐 아니라 시스템의 양극단 스케줄가능성을 보장하는 스케줄링 알고리즘을 제안하였다. 선행제약을 갖는 비주기 태스크 모델의 필요성 및 로컬 노드 관점에서만 분석되던 비주기 태스크의 스케줄가능성 분석의 문제점 및 양극단 스케줄가능성 분석의 필요성을 서술하였다. 양극단 제약을 고려하여 분산 실시간 시스템을 스케줄하기 위해 태스크와 메시지의 우선순위 할당을 위한 양극단 여유도 기반 우선순위 할당 방법을 제안하였다. 제안된 알고리즘을 off-line 스케줄에서 많이 사용되는 RM 방법과 이용률을 비교하여 양극단 여유도 기반의 우선순위 할당이 더 효율적인 방법임을 나타내었다. 앞으로 연구할 과제로는 제안된 알고리즘을 좀 더 복잡한 모델에 적용해 보는 것과 on-line 스케줄링 방법으로 적용하는 것이 있다.

참고문헌

[1] J. Y.-T Leung and J. Whitehead, "On complexity of fixed-priority scheduling of periodic real-time tasks", *Performance Evaluation,* 2(4), pp. 237-250, December, 1982.

[2] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems", *Microprocessing and Microprogramming,* vol.40, no. 2, pp. 117-134, Apr., 1994.

[3] R. Gerber and S.S. Hong, "Guaranteeing real- time requirements with resource-based calibration of periodic processes", *IEEE Tr. on Software Engineering,* 21(7), July, 1995.

[4] J. W. Park, Y. S. Kim, S. S. Hong, M. Saksena, S. H. Noh and W. H. Kwon, "Network conscious of distributed real-time systems", *Journal of System Architecture,* pp. 131-156, 1998.

[5] S. Faucou, A.-M. Deplanche and J.-P. Beauvais, "Heuristic techniques for allocating and scheduling communicating", *WFCS-2000,* pp. 257-265, 2000.

[6] H. Y. Kim and H. S. Park, "Period and priority assignment method for DCS Design", *Asian Journal of Control*, vol. 5, no. 3, pp. 422-432, Sept., 2003.

[7] D. Isovic and G. Fohler, "Handling sporadic tasks in off-line scheduled distributed real-time systems", *Proceedings of the 11th Euromicro Conference on Real-Time Systems,* pp. 60-67, June, 1999.

[8] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment", *Journal of the ACM,* vol.20, no. 1, pp. 46-61, 1973.

[9] A. Burns, "Preemptive priority based scheduling: an appropriate engineering approach in principles of real-time systems", *Prentice Hall,* 1994.

[10] K. Tindell, H. Hansson, and A. Wellings, "Analyzing real-time communications: controller area network", *IEEE Real-time Systems Symposium,* 1994.

[11] J. Xu and D. Parnas, "Scheduling processes with release times, deadlines, precedence and exclusion relations", *IEEE Tr. on Software Engineering,* pp. 360-369, March, 1990.

[12] N. Suri, M. M. Hugue, and C. J. Walter, "Synchronization issues in real-time systems", *Proc. of the IEEE,* vol. 82, pp. 41-54, Jan., 1994.

[13] H. S. Park, Y. H. Kim, D. S. Kim, W. H. Kwon, "A scheduling method for network-based control systems", *IEEE Transaction on Control System Technology,* vol. 3. no. 3, pp 318-330, May, 2002.

[14] 김형육, 윤건, 박홍성 "노이즈 환경 하에서 태스크와 메시지 스케줄링", 제어자동화시스템공학 논문지, 제10권, 제4호, pp. 377-384, 2004년 4월.

표기법

τ_β^α : 노드 β 에 위치한 태스크 α. 여기서 $\alpha \in \{Si, Pi\}$ 이고 Si 와 Pi 는 각각 i번째 비주기 태스크와 주기태스크를 의미한다. $\beta \in \{Sj, Cj, Aj\}$ 이고 Sj, Cj, Aj 는 각각 j번째 센서노드, 제어노드, 구동노드를 의미한다.

m_k^γ : γ 형태의 k번째 메시지. 여기서 $\gamma \in \{S, P\}$ 이고 S 와 P 는 각각 주기 메시지와 비주기 메시지를 의미한다.

e_δ : 태스크 δ 의 순수 수행시간. 여기서 $\delta = \tau_\beta^\alpha$ 이다.

C_δ : 메시지 δ 의 순수 전송시간. 여기서 $\delta = m_k^\gamma$ 이다.

T_δ : δ 의 주기. 여기서 $\delta \in \{\tau_\beta^\alpha, m_k^\gamma\}$ 이다.

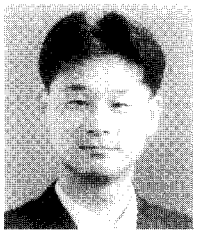
d_δ : δ 의 데드라인. 여기서 $\delta \in \{\tau_\beta^\alpha, m_k^\gamma\}$ 이다.

p_δ : δ 의 주기. 여기서 $\delta \in \{\tau_\beta^\alpha, m_k^\gamma\}$ 이다.

ϕ_δ : δ의 초기시작시간. 여기서 $\delta \in \{\tau_\beta^\alpha, m_k^\gamma\}$ 이다.

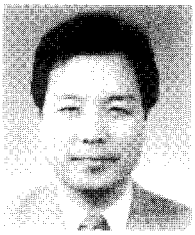
$MADT_i^{Ctrl.}$: i번째 제어루프의 양극단 시간제약 또는 데드라인.

$MADT_i^{Evnt.}$: i번째 이벤트 경로의 양극단 시간제약 또는 데드라인.



김 형 육

1973년 7월 16일생. 1999년 강원대 제어계측공학과 졸업. 동대학원 제어계측공학과 석사(2001). 2001년~현재 동대학원 제어계측공학과 박사과정. 관심분야는 실시간 스케줄링, 필드버스, 무선데이터 통신 등.



박 홍 성

1961년 3월 16일생. 1983년 서울대 제어계측공학과 졸업. 동대학원 제어계측공학과 석사(1986). 동대학원 제어계측공학과 박사(1992). 1992년~현재 강원대 전기전자정보통신공학부 교수. 관심분야는 실시간 네트워크, 무선 네트워크 등.

오 훈

1960년 5월 2일생. 1981년 성균관대 전자공학과 졸업. 1983년~1989년 삼성전자. 1992년 텍사스A&M대 전산학 석사. 동대학원 전산학 박사(1995). 1996년~2000년 삼성전자. 2000년~현재 강원대 BK21 계약 조교수. 관심분야는 이동통신, 실시간 컴퓨팅, 무선통신프로토콜 등.