유비쿼터스 환경에서의 상황 인지적 우선순위 기반의 실시간 스케줄링 기법*

이효남○, 임성화, 김재훈
아주대학교 정보통신전문대학원
{narsisse○, holyfire, jaikim}@ajou.ac.kr

Real-Time Scheduling Scheme Based on Context-Aware Priority in Ubiquitous Smart Space

Hyo-Nam Lee○, Sung-Hwa Lim, Jai-Hoon Kim
Graduate School of Information and Communication, Ajou University

요 약

유비쿼터스 지능 공간 환경에서 중요한 이슈는 사용자에게 현재 상황에 최적의 서비스를 제공하는 것이며, 이를 위해서 상황 인식(Context Aware) 기법에 대한 연구가 널리 진행되고 있다. 유비쿼터스 지능 공간은 분산 배치된 수많은 애플리케이션 및 장치와 같은 스마트 객체들이 존재하여 사용자에게 최적의 서비스를 제공하는 환경이다. 유비쿼터스 지능 공간에서는 각 스마트 객체가 수집하는 데이터는 무수히 많다. 사용자에게 최적의 서비스를 제공하기 위해선 데이터들을 빠르게 처리하여 서비스를 제공하여야 한다. 현재 연구된 스케줄링 방법은 데이터 처리에만 중점을 두었기 때문에 스스로 상황을 인지하여 예측하여야 하는 유비쿼터스 지능 공간을 위한 미들웨어에는 적합하지 않다. 본 논문은 수집한 데이터를 바탕으로 상황을 인식하고 태스크의 우선순위를 재조정하는 상황 인지형 실시간 스케줄링 기법을 제안한다. 제안하는 상황 인지형 실시간 스케줄링 기법인 U-RM, U-EDF와 기존의 RM, EDF 알고리즘을 비교하여 성능평가를 한 결과 U-RM은 기존의 RM보다 최대 20.7%의 성능 향상을 보였으며, U-EDF의 경우 기존의 EDF보다 최대 26.8%의 성능 향상을 보였다.

1. 서 론

유비쿼터스 컴퓨팅은 일상생활 속에 편재해 있는 컴퓨팅 자원을 이용하여 사용자가 언제 어디서나 동적인 서비스를 받을 수 있는 환경을 제공한다. 조용한 컴퓨팅 보이지 않는 컴퓨팅, 사라지는 컴퓨팅 등의 용어는 유비쿼터스 컴퓨팅에 관한 사용자 인터페이스 관점을 잘 설명해주고 있다[1].

유비쿼터스 지능 공간에서는 사용자에게 최적의 서비스를 제공하기 위하여 수많은 애플리케이션 및 장치와 같은 스마트 객체들이 존재한다. 이 자원들은 사용자에게 서비스하기 위하여 데이터를 수집하는데, 이 데이터들은 수시로 변화하며 다양하다.

유비쿼터스 지능 공간에서 사용자에게 알맞은 서비스를 제공하기 위해서는 주변 환경이나 데이터에 대해 능동적으로 인지하는 능력이 필요하다. 유비쿼터스 지능 공간의 미들웨어와 같이 자율적 수행능력을 갖는 시스템은 이러한 서비스를 제공하는데 있어 효과적인 방법이다. 유비쿼터스 지능 공간에서는 애플리케이션 및 장치와 같은 스마트 객체들의 협업을 하기 때문에 수집되는 데이터도 굉장히 많다. 이때 사용자가 원하는 정보나 서비스를 실시간으로 제공하기 위해서는 수집되는 정보를 빠른 시간 안에 처리할 수 있는 스케줄링 기법이 필요하다. 태스크의 스케줄링은 실시간 시스템에서 태스크들의 실시간성을 만족시키기 위하여 가장 중요한 요소 중의 하나이다.

유비쿼터스 환경에서는 상황에 따라 태스크의 우선순위가 동적으로 변하는 상황이 많이 발생한다. 예를 들면, 유비쿼터스 시스템 환경이 갖춰진 건물이 있다. 이 건물은 많은 사람들이 이용을 하고, 사용자들에게 적합한 서비스를 제공하기 위하여 애플리케이션, 장치와 같은 수많은 스마트 객체들이 존재한다. 유비쿼터스 미들웨어는 수많은 스마트 객체들의 정보를 수집하고 데이터의 우선순위에 맞게 스케줄링하여 처리한다. 정상적인 상황에서는 다른 데이터보다 우선순위가 낮은 온도센서의 데이터를 분석하여 건물의 각 구역의 온도를 조절하게 된다. 온도센서의 데이터는 특정한 상황이 아니면 변화가 크지 않고 다른 서비스보다 중요도가 낮기 때문에 다른 데이터에 비해 우선순위가 낮다. 미들웨어는 스케줄링하여 처리된 데이터는 분석 및 추론을 통하여 현재 상황을 알게 되고 현재 상황을 인지하는 고수준 데이터를 만들어 낸다.

만약 많은 사람이 존재하고, 여러 상황이 발생하여 수많은 데이터가 수집이 되는 환경에서 건물 내 한 구역에서 화재가 났다고 가정한다. 정상적인 상황에서 화재가 발생하는 긴급 상황으로 바뀐 것이다. 건물 내의 수많은 온도 센서 중 한 구역의 온도 센서의 데이터가 급격히

* 본 연구는 21세기 프론티어 연구개발사업의 일환으로 추진되고 있는 지식경제부의 유비쿼터스컴퓨팅및네트워크원천기반기술개발사업의 08B3-S2-10M 과제로 지원된 것임.
* 2007년도 아주대학교 일반교내연구비 지원에 의하여 수행되었음.

증가했다면, 유비쿼터스 미들웨어는 온도 센서의 이상 값을 감지하게 된다. 미들웨어는 센서의 단순한 고장인지, 화재 발생 상황인지 알기 위해 근접 온도 센서의 데이터와 이상 데이터를 보내온 센서의 데이터를 빠르게 처리하여야한다. 이러한 긴급 상황에서는 근접한 온도 센서의 데이터와 보내온 센서의 데이터의 중요도가 급격히 증가하게 된다. 하지만 기존의 스케줄링 기법을 이용한다면 평소와 같이 정해진 우선순위로 인하여 특정 상황에서 중요도가 급격히 증가한 온도 센서의 데이터는 중요도와 우선순위가 다른 데이터에 비해 낮게 처리된다. 유비쿼터스 환경에서는 수많은 데이터가 동적으로 변하고 생성되기 때문에 데이터의 발생이 급격히 증가하는 특정 상황에서 실시간성을 어기는 데이터가 존재하게 된다. 만약 이런 상황에서 온도 센서의 데이터가 우선순위에 밀려 deadline을 지키지 못할 경우에는 화재로 인한 크나큰 손실이 발생하게 된다.

이와 같은 이유로 현재 연구된 스케줄링 방법은 유비쿼터스 지능 공간 미들웨어에 적용하기에는 한계가 있다. 본 논문에서는 현재 연구된 스케줄링 기법을 알아보고, 상황 인지적 우선순위 기반의 실시간 스케줄링 기법을 제안한다. 연구한 상황 인지적 우선순위 기반의 실시간 스케줄링 기법은 단일 프로세서를 가지고 실시간성 보장을 받는 큐와 실시간성을 요하지 않은 큐를 가지는 멀티 큐 시스템의 환경을 가진다.

2. 관련 연구

2.1 실시간 스케줄링

RM(Rate Monotonic) 스케줄링 알고리즘은 상용 실시간 운영체제에서 보편적으로 사용되고 있는 것 중에 하나이다. RM 알고리즘은 하나의 프로세서 상에서 고정된 우선권을 가지는 프로세스들에 대한 선점방식의 스케줄링 알고리즘이다. 태스크의 주기가 짧을수록 높은 우선순위를 할당하는 스케줄링 방식이다[2].

EDF(Earliest Deadline First) 알고리즘은 마감시간이 가까운 태스크를 우선 스케줄링 하는 것이다. EDF는 현재 준비상태에 있는 태스크들 중에서 마감시간이 가장 짧은 태스크를 선택하여 수행한다. 태스크들의 우선순위가 시간이 흐름에 따라 동적으로 변화하므로 동적 우선순위 기반 알고리즘이라고도 불린다. RM 알고리즘의 경우 우선순위가 한번 부여되면 더 이상 변하지 않는 고정 우선순위 방식이므로 이에 비교된다.

EDF의 가장 큰 장점은 이론적으로 총 이용률이 1 이하이기만 하면 항상 스케줄링이 가능하다는 것이다. 즉, EDF 알고리즘은 상기한 태스크 모델에서 최적의 스케줄이며 당연히 RM 알고리즘보다 더 좋은 성능을 낸다. 또한 알고리즘도 RM에 비하여 크게 복잡하지도 않다. 그러나 대부분의 실시간 운영체제들은 RM 알고리즘과 같이 정적 우선순위 기반의 스케줄링 기법을 채용하고 있다. 그 이유는 이론의 가정이 되는 태스크 모델과 실제 수행 환경에서의 태스크들과 차이가 크다는 점이다. 즉, 태스크들의 수행시간과 마감시간, 주기 등을 정확히 예측하여야만 EDF등을 이용하여 총 이용률을 더 향상 시킬 수 있다. 실제로는 태스크 모델을 이론에서와 같이 정확히 가정할 수가 없다[2].

2.2 동적 스케줄링 기법

동적 스케줄링 기법은 정적 스케줄링에서 발생하는 부하 불균형 문제를 해결하기 위한 기법으로 각 반복의 실행 시간에 대한 예측이 불가능한 응용을 처리할 때 각 프로세서에 할당될 작업량을 하나의 중앙 작업 큐를 이용하여 동적으로 조절한다. 따라서 임의의 한 시점에서 프로세서는 하나 또는 그 이상의 반복을 할당받는다.

SS(Self Scheduling) 기법은 모든 반복들의 처리가 완료될 때까지 각 프로세서는 중앙 작업 큐로부터 하나의 반복씩 할당받아 처리한다. 모든 프로세서가 하나의 반복 내에 수행을 완료할 수 있기 때문에 완벽한 부하 평균을 유지할 수 있다. 그러나 하나의 반복 단위로 할당받기 때문에 중앙 작업 큐에 대한 접근이 증가하므로 동기화 오버헤드가 커진다[3].

GSS(Guided Self Scheduling)는 [n/P] 반복의 수를 프로세서에 할당한다. 여기서 n은 남아 있는 반복의 수이고, P는 프로세서의 수를 의미한다. 이 알고리즘은 초기에 큰 수의 반복을 할당하고, 점차적으로 반복의 수를 줄여나간다. 따라서 부하 불균형 문제는 해결되나, 반복의 실행 시간이 크게 변하는 응용일 때 처음 할당된 반복이 너무 커서 나머지 다른 반복이 끝날 때까지 끝나지 않는 경우가 발생한다[4].

기존의 RM 스케줄링 알고리즘은 주기가 짧은 태스크를 우선순위를 높이고, EDF 스케줄링 알고리즘의 경우 Deadline이 가까운 태스크의 우선순위를 높였다. 지금까지 살펴본 실시간 스케줄링 기법은 태스크들의 우선순위가 상황에 따라 동적으로 변해야하는 유비쿼터스 지능 공간 환경에서는 적합하지 않다. 유비쿼터스 미들웨어는 동일한 데이터라도 시점의 변화에 따라 데이터의 우선순위가 바뀔 수 있기 때문이다. 그러므로 유비쿼터스 지능 공간의 미들웨어는 실시간성을 가지는 데이터를 처리할 때 현재 상황을 인지하고, 데이터의 우선순위를 동적으로 바꾸는 기법이 필요하다.

3. 본론

3.1 유비쿼터스 환경의 미들웨어 구조

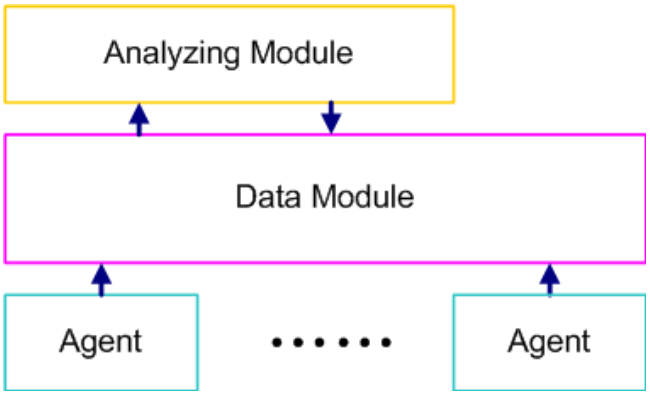

그림 1. 미들웨어의 기본 구조

그림 1은 유비쿼터스 환경을 위한 미들웨어를 단순하게 표시한 구조이다. 분산 배치된 수많은 Agent가 존재하며, 미들웨어는 각각의 Agent는 수집한 데이터를 Data Module에 보내고, Data Module은 Agent에게서 받은 데이터를 Analyzing Module로 보내는 기본 흐름을 가진다. 본 논문에서 기술하는 유비쿼터스 환경의 미들웨어는 단일 프로세서 시스템이며, 멀티 큐를 갖는 구조이다.

표 1은 유비쿼터스 환경의 미들웨어의 각 모듈의 특징을 정리한 표이다.

표 1. 미들웨어의 모듈별 특징

모듈	특징
Agent	• 독립적으로 존재 • 데이터를 수집할 스마트 객체에 분산적 배치
Data Module	• 분산 배치된 Agent를 등록 • 멀티 큐 환경 (스케줄링 기법 적용) • 수집된 데이터의 오류 검사 • Filtering & Aggregation • 데이터를 DB에 저장
Analyzing Module	• 분석 및 추론하여 고수준 데이터 생성 • 상황 인지

3.2 스케줄링 환경

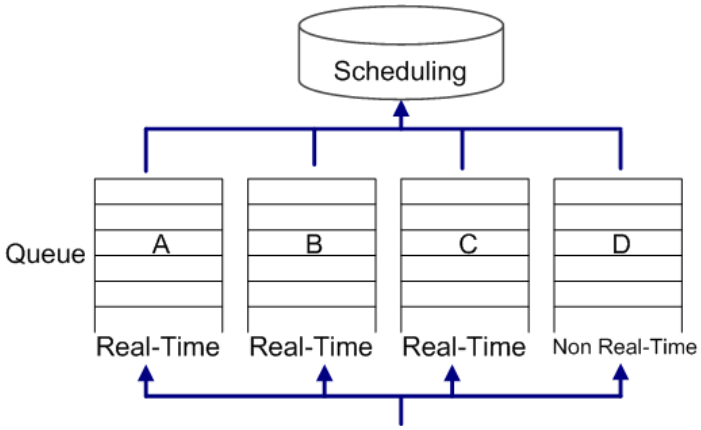

그림 2. 멀티 큐 구조

그림 2는 4개의 멀티 큐를 갖는 부분만을 표시한 그림이다. A, B, C는 실시간 처리를 요하는 큐이고, D는 실시간 처리를 요하지 않는 큐이다. Agent는 미리 정의된 형식에 맞게 데이터를 큐에 전달한다. A, B, C는 실시간 처리를 보장해야한다.

표 2. 각 Queue의 용도

큐 타입	용도	우선순위
A	Application Attribute	1
B	System Resource	2
C	Sensor & Device Attribute	3

표 2는 그림 2의 각각의 큐의 속성을 나타낸 표이다. A, B, C는 실시간 처리를 요하는 큐이다. A, B, C 모두 주기적으로 데이터가 들어오는 주기적 형식이다.

미리 정의된 형식의 데이터가 각 큐에 들어오기 때문에 우리는 A, B, C의 우선순위를 알 수 있기 때문에 표 2와 같이 가정한다.

3.3 상황 인지적 우선순위 기반의 스케줄링 기법

유비쿼터스 환경에서는 여러 상황이 발생함에 따라 데이터의 우선순위가 바뀔 수 있다. 만약 유비쿼터스 미들웨어가 이러한 상황을 미리 예측하고 보다 빠르게 큐의 데이터를 스케줄링하여 처리한다면 사용자에게 제공하는 서비스의 질은 최적에 가까워질 것이다.

표 3. 상황 인지적 우선순위 변화의 예

시나리오	긴급 상황	긴급 상황 시 우선순위가 높은 데이터
현관 감시 카메라	현관 침입 상황	• 카메라의 데이터 • 비전 센서의 데이터
온도 센서	화재	• 주위 온도 센서의 데이터 • 고장이 의심되는 센서의 데이터
스마트 펜던트	어린이 납치	• 스마트 펜던트의 데이터

유비쿼터스 환경에서는 상황에 따라 태스크의 우선순위가 동적으로 변하는 상황이 많이 발생한다. 표 3은 태스크의 우선순위가 동적으로 변하는 몇 가지의 상황을 예를 들어 표시한 표이다. 예를 들면, 유비쿼터스 시스템 환경이 갖춰진 건물이 있다. 이 건물은 많은 사람들이 이용을 하고, 사용자들에게 적합한 서비스를 제공하기 위하여 애플리케이션, 장치와 같은 수많은 스마트 객체들이 존재한다. 유비쿼터스 미들웨어는 수많은 스마트 객체들의 정보를 수집하고 데이터의 우선순위에 맞게 스케줄링하여 처리한다. 표 3의 화재 시나리오를 가정해본다. 정상적인 상황에서는 다른 데이터보다 우선순위가 낮은 온도센서의 데이터를 분석하여 건물의 각 구역의 온도를 조절하게 된다. 온도 센서의 데이터는 특정한 상황이 아니면 변화가 크지 않고 다른 서비스보다 중요도가 낮기 때문에 다른 데이터에 비해 우선순위가 낮다. 미들웨어는 스케줄링하여 처리된 데이터는 분석 및 추론을 통하여 현재 상황을 알게 되고 현재 상황을 인지하는 고수준 데이터를 만들어 낸다.

만약 많은 사람이 존재하고, 여러 상황이 발생하여 수많은 데이터가 수집이 되는 환경에서 건물 내 한 구역에서 화재가 났다고 가정한다. 정상적인 상황에서 화재가 발생하는 긴급 상황으로 바뀐 것이다. 건물 내의 수많은 온도 센서 중 한 구역의 온도 센서의 데이터가 급격히 증가했다면, 유비쿼터스 미들웨어는 온도 센서의 이상값을 감지하게 된다. 미들웨어는 센서의 단순한 고장인지, 화재 발생 상황인지 알기 위해 근접 온도 센서의 데이터와 이상 데이터를 보내온 센서의 데이터를 빠르게

처리하여야한다. 이러한 긴급 상황에서는 근접한 온도 센서의 데이터와 보내온 센서의 데이터의 중요도가 급격히 증가하게 된다. 하지만 기존의 스케줄링 기법을 이용한다면 평소와 같이 정해진 우선순위로 인하여 특정 상황에서 중요도가 급격히 증가한 온도 센서의 데이터는 중요도와 우선순위가 다른 데이터에 비해 낮게 처리된다. 유비쿼터스 환경에서는 수많은 데이터가 동적으로 변하고 생성되기 때문에 데이터의 발생이 급격히 증가하는 특정 상황에서 실시간성을 어기는 데이터가 존재하게 된다. 만약 이런 상황에서 온도 센서의 데이터가 우선순위에 밀려 deadline을 지키지 못할 경우에는 화재로 인한 크나큰 손실이 발생하게 된다.

본 논문에서 제안하는 상황 인지적 우선순위 기반의 스케줄링 기법은 다음과 같다. 특정 온도 센서가 급격히 증가한 데이터를 보내왔다면, Analyzing Module은 데이터 분석을 통하여 이상 상황이라는 것을 감지하고 중요도와 우선순위가 낮았던 온도 센서의 데이터가 중요하다는 것을 감지한다. 미들웨어는 동적으로 온도 센서의 우선순위를 다른 데이터의 우선순위보다 높게 재조정하여 스케줄링을 한다. 제안하는 상황 인지적 우선순위 기반의 실시간 스케줄링 기법을 이용한다면 긴급한 상황을 인지하여 온도 센서의 데이터를 우선 처리한다면 다른 스케줄링 기법을 이용할 때 보다 긴급 상황에서의 데이터 우선순위 기반 데이터 손실률을 최소로 할 수 있게 된다.

이러한 상황을 본다면 유비쿼터스 지능 공간 환경에서는 평소엔 우선순위가 낮은 데이터가 특정한 상황이 되었을 때 우선순위가 높아질 수 있다는 것을 알 수 있다. 유비쿼터스 지능 공간에서는 수많은 애플리케이션 및 장치와 같은 스마트 객체가 존재한다. 그렇기 때문에 수집되는 수많은 데이터가 존재한다. 상황이 변하고 수집되는 데이터가 증가할수록 실시간을 만족시키지 못하는 데이터가 발생할 수 있다. 그러므로 유비쿼터스 지능 공간 환경에서는 현재의 상황을 인지하는 능력이 없는 기존의 스케줄링 기법으로는 한계가 있음을 알 수 있다.

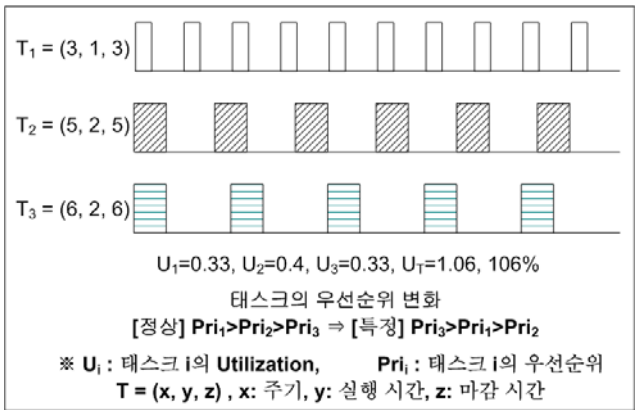

그림 3. 태스크 셋

그림 3과 같이 태스크 셋이 있다고 가정하자. T_1=(3, 1, 3)의 이용률은 33%이고, T_2=(5, 2, 5)의 이용률은 40%, T_3=(6, 2, 6)의 이용률은 33%이다. 그림 4의 태스크 셋의 총 이용률은 106%가 된다. 위 태스크 셋을 스케줄링 할 때 이용률이 100%가 넘기 때문에 실시간성을 만족시키지 못하는 데이터가 존재하게 된다.

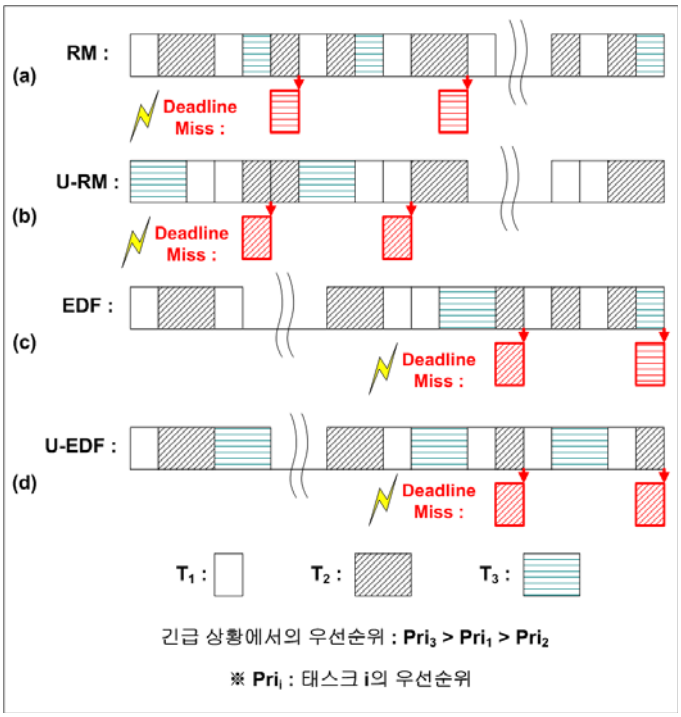

그림 4. 각 기법으로 스케줄링

그림 4는 그림 3의 태스크 셋을 스케줄링을 하여 이용률별로 실시간성을 만족시키지 못하는 데이터의 손실률을 보여준다. 정상 적인 상황의 우선순위가 [Pri_1>Pri_2>Pri_3]이며, 긴급 상황이 되었을 경우에는 [Pri_3>Pri_1>Pri_2]의 우선순위를 가지게 된다.

(a)는 RM(Rate Monotonic) 알고리즘 기법을 이용하여 태스크 셋을 스케줄링하고 실시간성을 만족 시키지 못하는 데이터를 보였다. (b)는 RM 알고리즘 기법에 제안하는 상황 인지적 우선순위 기반의 실시간 스케줄링 기법을 적용시킨 U-RM(Ubiquitous RM) 그림이다. (c)는 EDF(Earliest Deadline First) 알고리즘 기법을 이용하여 태스크 셋을 스케줄링하고 실시간성을 만족 시키지 못하는 데이터를 보였다. (d)는 EDF 알고리즘 기법에 제안하는 상황 인지적 우선순위 기반의 실시간 스케줄링 기법을 적용시킨 U-EDF(Ubiquitous EDF) 그림이다.

표 4. 각 기법의 Penaly 점수

기법	환경 특성	손실 점수
RM	Pri_3 > Pri_1 > Pri_2 $Penalty_1$ = 5, $Penalty_2$ = 3, $Penalty_3$ = 1	10
U-RM		2
EDF		7
U-EDF		2

※ Pri_i : 태스크 i의 우선순위
※ $Penalty_i$: i번째 우선순위를 가지는 태스크의 손실점수

표 4는 그림 4의 각각의 알고리즘으로 스케줄링한 후 Deadline을 만족시키지 못한 손실점수를 표시한 표이다. 긴급 상황이 발생하여 태스크의 우선순위가 [Pri_1>Pri_2>Pri_3]에서 [Pri_3>Pri_1>Pri_2]로 바뀌었다. 우선순위에 따른 손실 점수를 $Penalty_1$=5, $Penalty_2$=3, $Penalty_3$=1로 준다면, 정상 적인 상황에서의 우선순위라면 T_1의 Penalty는 5점, T_2의 Penalty는 3점, T_3의 Penalty는 1점의 손실 점수를 적용할 수 있다. 하지만 본 논문에서는 분석 및 추론에 따른 상황 변화에 따른 우선순위 기반 실시간 스케줄링이기 때문에 우선순위가 바뀌는 환경을 보인다. 긴급 상황으로 인하여 태스크의 데이터 우선순위가 바뀌어 T_3의 Penalty는 5점, T_2의 Penalty는 3점, T_1의 Penalty는 1점의 손실 점수를 받게 된다.

그림 5에 위 손실 점수를 적용하여 데이터 손실률을 알아보면, RM 스케줄링 알고리즘 기법으로는 T_3의 실시간 만족을 못 시키는 잡(Job)이 2번이 발생한다. 긴급 상황에서 T_3의 우선순위는 제일 높기 때문에 (5점×2회)로 10점의 손실을 발생시킨다. 본 논문에서 제안하는 U-RM 스케줄링 알고리즘 기법을 적용한다면 T_3보다 우선순위가 낮은 T_2의 실시간 만족을 못 시키는 잡(Job)이 2번 발생한다. T_2의 손실 점수는 1점으로 (1점×2회)로 2점의 손실을 발생시킨다.

EDF 스케줄링 알고리즘 기법으로는 T_2와 T_3의 실시간 만족을 못 시키는 잡(Job)이 각각 1번 발생한다. T_2의 손실 점수는 1점으로 (1점×1회)로 1점의 손실을 발생시키고, T_3의 손실 점수는 5점으로 (5점×1회)로 5점의 손실을 발생시켜 총 6점의 손실 점수를 발생시킨다. 본 논문에서 제안하는 U-EDF 스케줄링 알고리즘을 적용한다면 T_2의 실시간 성을 만족 못 시키는 잡(Job)이 2번 발생한다. (1점×2회)로 2점의 손실을 발생 시킨다.

종합해보면 위 4개의 스케줄링 기법을 이용한 결과 RM은 10점, EDF는 6점, U-RM은 2점, U-EDF는 2점의 손실을 발생 시킨다. 유비쿼터스 지능 공간 환경과 같은 상황의 변화에 따라 데이터의 우선순위가 바뀌는 환경에서는 RM, EDF와 같은 현재까지 연구된 스케줄링 기법을 적용하는 데에는 문제점이 있다. 본 논문에서는 제안하는 상황 변화를 인지하여 데이터의 우선순위를 동적으로 바꾸는 스케줄링 기법을 적용한다면 동적으로 우선순위가 바뀌는 유비쿼터스 환경에서의 데이터 손실률에 따른 만족도를 향상 시킬 수 있다.

3.4 성능 평가

본 논문에서 제안한 상황 인지적 우선순위 기반의 실시간 스케줄링 기법을 다양한 환경에 적용하여 성능 평가를 하였다.

표 5. Task Set

Task Set	T = (P_i, e_i, D_i)	Task Utilization	Total Utilization
1	T1 : (5, 2, 5) T2 : (7, 1, 7) T3 : (8, 3, 8)	0.4 0.143 0.375	91.8%
2	T1 : (6, 2, 6) T2 : (7, 2, 7) T3 : (8, 3, 8)	0.33 0.286 0.375	99.1%
3	T1 : (3, 1, 3) T2 : (5, 2, 5) T3 : (6, 2, 6)	0.33 0.4 0.33	106%
4	T1 : (4, 1, 4) T2 : (5, 3, 5) T3: (12, 4, 12)	0.25 0.6 0.33	118%
5	T1 : (5, 3, 5) T2: (10, 5, 10) T3: (12, 4, 12)	0.6 0.5 0.33	143%
6	T1 : (4, 3, 4) T2 : (6, 3, 6) T3 : (10, 4, 10)	0.75 0.5 0.4	165%

표 5는 다양한 환경의 태스크 셋이다. 유비쿼터스 환경에서는 사용자에게 서비스하기 위하여 애플리케이션과 장치 같은 수많은 스마트 객체들이 존재한다. 유비쿼터스 환경에는 절대적으로 고정되어 처리되는 데이터도 있지만 사용자 주위 환경이 수도 없이 바뀌기 때문에 데이터 발생 빈도가 일정하지가 않다. 특정한 상황에서는 데이터 발생 빈도가 급격히 높아져 실시간 처리를 만족시키지 못하는 상황이 발생할 수도 있다. 여러 상황이 존재하기 때문에 표 5와 같이 태스크 셋의 이용률을 다양하게 발생 시켜 기존의 연구된 RM 알고리즘, EDF 알고리즘 기법으로 스케줄링하고 본 논문에서 제안하는 U-RM 알고리즘, U-EDF 알고리즘 기법으로 스케줄링하여 만족도를 비교를 하였다.

표 6. 성능 평가 환경

그림	상황	우선순위	손실 점수
그림 5.	정상	Pri_1 > Pri_2 > Pri_3	$Penalty_1$ = 5 $Penalty_2$ = 3 $Penalty_3$ = 1
	긴급	Pri_3 > Pri_1 > Pri_2	
그림 6.	정상	Pri_1 > Pri_2 > Pri_3	$Penalty_1$ = 5 $Penalty_2$ = 3 $Penalty_3$ = 1
	긴급	Pri_3 > Pri_1 > Pri_2	
※ Pri_i : 태스크 i의 우선순위 ※ $Penalty_i$: i번째 우선순위를 가지는 태스크의 손실 점수			

표 6은 성능 평가한 환경을 요약 정리한 표이다. 그림 5는 실시간 처리를 만족 시키지 못한 경우 우선순위에 따라 만족 못 시키는 잡(Job)에 대하여 $Penalty_1$=5, $Penalty_2$=3, $Penalty_3$=1점의 손실 점수를 적용하여 성능 평가를 하였고, 환경에 따라 태스크 마다 중요도가 다를 수 있기 때문에 그림 6은 손실 점수를 $Penalty_1=(3)^2$, $Penalty_2=(2)^2$, $Penalty_3=(1)^2$로 주어 성능 평가를 하였다. 성능 평가한 그래프의 X축은 이용률(%)을 의미하며, Y축은 만족도(Max=100)를 표시한다.

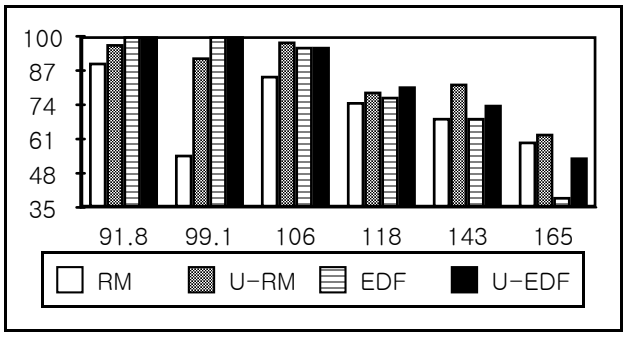

그림 5. 각 스케줄링 기법의 데이터 만족도 비교

그림 5는 표 5의 여러 상황을 RM, EDF 알고리즘과 본 논문에서 제안하는 U-RM, U-EDF 알고리즘으로 스케줄링하여 실시간 처리를 만족 못 시키는 데이터의 손실 점수를 계산하여 만족도를 평가한 그림이다. RM 알고리즘의 경우 태스크가 3개 일 경우 스케줄링을 보장하는 최대의 이용률은 $U_{RM}(n)=n(2^{1/n}-1)$로 총 이용률 78% 안에서는 실시간 스케줄링을 보장한다. 그러므로 그 이후의 이용률에 대해 성능 평가를 하였다. 그림 5를 보면 특정한 상황이 발생하여 데이터의 우선순위가 바뀌었을 때 RM 알고리즘 보다 U-RM 알고리즘이 데이터의 손실이 적다는 것을 볼 수 있다. 총 이용률 100%전과 후 모두 U-RM 알고리즘의 우선순위에 기반 한 데이터 손실률이 적어 만족도가 높음을 보였다. 또한 EDF 알고리즘의 경우 총 이용률 100% 안에서는 실시간 스케줄링을 보장한다. 그림 5를 보면 100%까지는 데이터 손실 없이 만족도 100을 보였다. 하지만 유비쿼터스 지능 공간 환경에서 특정 상황에 따라 데이터의 발생이 급격히 증가함에 따라 실시간 처리를 요하는 태스크들의 총 이용률이 100%가 넘는 상황이 발생할 수 있다. 이와 같은 상황에서 데이터의 우선순위가 변했을 때 우선순위가 높아진 데이터의 손실이 크면 안 된다. 그림 5를 보면 이와 같은 상황에서 데이터의 우선순위를 고려하지 않고 스케줄링 한 EDF 알고리즘이 제안한 U-EDF 알고리즘보다 우선순위 기반의 데이터 손실률이 크다는 것을 알 수 있다. U-EDF 알고리즘은 EDF 알고리즘보다 데이터 손실이 작기 때문에 만족도가 높음을 알 수 있다.

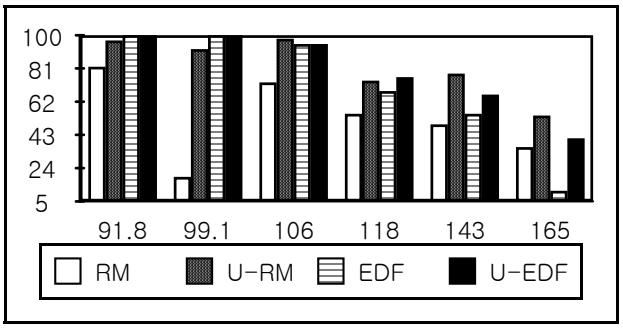

그림 6. 각 스케줄링 기법의 데이터 만족도 비교

유비쿼터스 환경에서는 상황과 환경에 따라 태스크의 중요도가 다를 수 있다. 각각의 중요 점수가 다를 수 있기 때문에 손실 점수 또한 다른 상황이 발생한다. 그림 6은 태스크들의 우선순위 기반의 손실 점수를 다르게 부여하여 평가한 그래프이다. $Penalty_1=(3)^2$, $Penalty_2=(2)^2$, $Penalty_1=(1)^2$의 손실 점수를 주어 평가하였다. 상황에 따라 모든 태스크들의 우선순위에 따른 손실 점수가 다르기 때문에 여러 손실 점수에 따른 성능 평가를 보였는데, 마찬가지로 본 논문에서 제안하는 상황 인지적 우선순위 기반의 실시간 스케줄링 기법이 우선순위 기반 손실률이 작았으며, 만족도가 높임을 보였다. U-RM은 기존의 RM보다 최대 20.7%의 성능 향상을 보였으며, U-EDF의 경우 기존의 EDF보다 최대 26.8%의 성능 향상을 보였다.

4. 결론

유비쿼터스 환경에서는 사용자에게 최적의 서비스를 제공하기 위하여 수많은 스마트 객체들이 존재한다. 수집되는 수많은 데이터들은 특정 상황에 따라 수시로 변하며 다양하다. 상황이 변함에 따라 데이터들의 우선순위가 다양하게 동적으로 바뀐다. 하지만 현재 연구된 알고리즘은 상황에 따라 우선순위가 동적으로 바뀌는 환경에는 적합하지가 않다. A라는 상황에서 T_1의 우선순위가 T_2보다 높았지만, B라는 상황이 발생함에 따라 T_2의 우선순위가 T_1의 우선순위보다 높아지는 상황이 빈번히 발생 할 수 있다. 현재 연구된 RM, EDF 알고리즘 기법을 이용한다면 우선순위 변화를 고려하지 않아 우선순위가 높아진 데이터의 손실률이 커진다. 본 논문에서는 유비쿼터스 미들웨어에서 사용자 주위 상황을 인식한 데이터를 스케줄링 기법에 추가하여 상황 인지적 우선순위 기반 실시간 스케줄링 기법을 제안하였다. 또한, RM, EDF 알고리즘과 제안한 U-RM, U-EDF 기법을 이용하여 데이터 우선순위 기반의 데이터 손실률을 계산하여 성능 측정을 하였다. 유비쿼터스 환경에서와 같이 데이터의 우선순위가 상황에 따라 동적으로 변하는 환경에서 제안한 기법을 이용하였을 경우 손실률이 적었고, 만족도를 높였음을 보였다. U-RM은 기존의 RM보다 최대 20.7%의 성능 향상을 보였으며, U-EDF의 경우 기존의 EDF보다 최대 26.8%의 성능 향상을 보였다.

참고 문헌

[1] Aninid K.Dey, Gregory D. Aborod, and Daniel Salber, “A conceptual Framework and a Toolkit for supporting the Rapid Prototypeing of Context-Aware Application“, Human-Computer International(HCI) Journal, vol.16, pp.97-166, 2001

[2] Jane W.S. Liu, “Real-Time Systems“, Prentice Hall, 2000

[3] Weisong Shi and Zhimin Tang, “Dynamic computation scheduling for load balancing in home-based software DSMs“, IEEE CNF, pp.248-253, 1999

[4] C. D. Polychronopolous, D. J. Kuck, “Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers“, IEEE Transaction on Computers, vol.36, No.12, pp.1425-1439, 1987