수정 비례 지분 스케줄러 및 평가법 설계

A Modified Proportional Scheduler and Evaluation Method

김 현 철* 박 정 석**
Hyun-Chul Kim Jeong-Seok Park

요 약

비디오나 오디오 스트림과 같은 처리 요구 조건이 주기적으로 발생하는 데이터들은 전송이나 재생시 시간적인 제약조건을 가진다. 일반적인 실시간 스케줄링 알고리즘은 이러한 연속성을 고려하지 아니하므로 멀티미디어 데이터를 스케줄링하기에 적절하지 않다.

비율조정 비례지분 스케줄러는 멀티미디어 데이터의 연속성을 고려하여 설계된 스케줄링 알고리즘이다. 이 알고리즘은 태스크가 자신의 지분보다 더 많은 자원을 할당 받지 못하도록 제어하기 위해 비율조정기를 사용한다. 그러나 엄격한 비율조정기로 인해 비율조정 비례지분 스케줄러는 자원 할당의 공정성을 보장하지 못하게 된다. 이는 과부하상황에서 시스템의 성능이 급격하게 감소되는 원인이 된다.

본 논문에서 제시하는 수정된 비례지분 스케줄러는 연속성, 시간 제약성과 같은 멀티미디어 데이터의 특성을 고려할 뿐만 아니라 비율조정기를 수정하여 자원 할당의 공정성을 유지하고 과부하시에 성능이 점진적으로 저하되게 한다. 또한 우선권 정책을 사용하여 문맥교환 횟수를 줄인다. 본 논문에서 제시하는 스케줄링 알고리즘의 성능을 측정하기 위해 스케줄링 알고리즘의 유연성을 측정할 수 있는 평가 방법을 제시하고, 평가 결과를 보인다.

Abstract

Since multimedia data such as video and audio data are displayed within a certain time constraint, their computation and manipulation should be handled under limited condition. Traditional real-time scheduling algorithms could not be directly applicable, because they are not suitable for multimedia scheduling applications which support many clients at the same time.

Rate Regulating Proportional Share Scheduling Algorithm is a scheduling algorithm considered the time constraint of the multimedia data. This scheduling algorithm uses a rate regulator which prevents tasks from receiving more resource than its share in a given period. But this algorithm loses fairness, and does not show graceful degradation of performance under overloaded situation.

This paper proposes a new modified algorithm, namely Modified Proportional Share Scheduling Algorithm considering the characteristics of multimedia data such as its continuity and time dependency. Proposed scheduling algorithm shows graceful degradation of performance in overloaded situation and the reduction in the number of context switching. Furthermore, a new evaluation method is proposed which can evaluate the flexibility of scheduling algorithm.

1. 서 론

최근 컴퓨터와 통신 기술이 발전함에 따라 네트워크를 이용한 멀티미디어(multimedia) 응용분야도 급속도로 발전되고 있다. 멀티미디어는 오디오(audio), 비디오(video), 그래픽(graphic), 이미지(image), 애니메이션(animation) 등의 데이터(data)들로 이루어진다. 이러한 데이터들 중 비디오와 오디오는 시간적인 제약조건을 가지므로 연속미디어(continuous media)로 분류된다. 그리고 텍스트, 그래픽, 이미지와 같은 데이터들은 시간적인 제약조건을 가지지 않으므로 이산미디어(discrete media)로 분류된다[1,2]. 연속미디어는 일정 시간마다 연속적으로 데이터 처리 요구가 발생하여

* 경주대학교 컴퓨터전자공학부 교수
kimhc@kyongju.ac.kr
** 국립청주과학대학 컴퓨터과학대학 컴퓨터과학과 조교수
jspark@chongjunc.ac.kr

주어진 시간 안에 처리가 완료되어 재생되어야 하는 특성을 가지므로 이산미디어보다 더 제한된 처리조건을 가지게 된다.

연속미디어를 효과적으로 다루기 위해 RM(Rate Monotonic)과 EDF(Earliest Deadline First) 스케쥴링 알고리즘을 기본으로 하는 경성 실시간 시스템에서의 스케쥴링 알고리즘들이 꾸준히 연구되어 왔다. RM은 주기가 짧은 태스크에게 주기가 긴 태스크보다 높은 우선 순위를 부여해 주는 정적인 스케쥴링 알고리즘이고, EDF는 마감시간에 가장 임박한 태스크에게 가장 높은 우선 순위를 부여해 주는 동적인 스케쥴링 알고리즘이다. 이러한 알고리즘들의 목표는 모든 태스크들의 마감시간을 지키는 것으로 과부하 상황에서는 시스템의 동작을 예측할 수 없다. 따라서 항상 스케쥴링 가능하도록 하기 위해 입장제어(admission control)를 두지만, 이는 자원 활용도를 낮추는 원인이 될 수 있다[3,4].

멀티미디어 시스템에서는 경성 실시간 시스템에서와는 달리 연속미디어의 마감시간을 다소 어기는 것이 치명적인 결과를 초래하지는 않으며, 각 미디어의 서비스 품질을 만족시킬 수 있는 범위 내에서 약간의 마감시간을 어기는 것은 허용되어질 수 있다. 그러므로, 멀티미디어 시스템에서의 연속미디어는 QoS(Quality of Service)의 허용범위 내에서 경성 실시간 시스템에서의 태스크들보다 좀 더 유연하게 스케쥴링될 수 있다[5,6].

연속미디어를 위한 또 다른 스케쥴링 방법으로 비율조정 비례지분 스케쥴링 알고리즘(Rate Regulating Proportional Share Scheduling Algorithm)이 있다. 이하 본 논문에서는 비율조정 비례지분 스케쥴링 알고리즘을 RRPSSA라 칭한다. RRPSSA[7]는 비례지분(proportional share) 방법에 기초를 둔 스트라이드(stride) 알고리즘을 수정한 것이다. 스트라이드 알고리즘은 일반적인 태스크를 위해 설계되었으며, 자원 할당에 있어서 공정성과 시스템의 예측가능성을 유지한다[8,9]. RRPSSA는 연속적인 미디어를 스케쥴링하기 위해 스트라이드 알고리즘을 변형하여 연속미디어의 시간적인 특성, 즉 태스크의 주기와 각 주기 안에서 요구되어지는 실행시간을 반영하였다. 비율조정기는 RRPSSA의 가장 중요한 개념이라고 할 수 있다. 비율조정기는 태스크가 주어진 주기 내에서 자원을 자신의 지분보다 더 많이 할당받지 못하도록 조정한다[7]. 그러나 이 알고리즘은 엄격한 비율조정기로 인하여 스트라이드 알고리즘의 장점인 자원할당에서의 공정성을 유지하지 못하고, 과부하 상태에서 성능을 예측 할 수 없다. 이는 과부하 상태에서 성능이 급격히 저하됨을 의미하고, 연속미디어의 서비스 질을 떨어뜨리게 된다. 따라서 RRPSSA를 이용하여 연속미디어를 스케쥴링하기에는 여전히 부족함이 있다.

이 논문에서는 스트라이드 알고리즘에 연속미디어의 시간적인 특성을 반영한 수정 비례지분 스케쥴링 알고리즘(Modified Proportional Share Scheduling Algorithm) MPSSA를 제시한다. 이 알고리즘은 자원할당에 있어서 스트라이드 알고리즘의 장점인 공정성을 유지하여 과부하 상태에서도 성능이 점진적으로 저하되고, 서비스를 해줄 태스크를 선정하는 과정에서 우선권 정책을 사용하여 문맥교환 회수를 줄인다. 나아가 스케쥴링 알고리즘의 성능을 평가하기 위한 스케쥴링 알고리즘의 유연성을 측정하는 방법을 제시한다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 RRPSSA의 특징 및 장단점을 설명하고 3장에서는 RRPSSA의 단점을 개선할 수 있는 MPSSA 알고리즘을 제안한다. 4장에서는 MPSSA 알고리즘을 실험한 모델 및 실험 결과를 보이고, RRPSSA 알고리즘과 비교한다. 5장에서는 알고리즘의 유연성을 평가하는 방법을 보이고 6장에서 결론을 맺는다.

2. 비율조정 비례지분 스케쥴링 알고리즘 (RRPSSA)

2.1 스케쥴링 파라메타

비율조정 비례지분 스케쥴링 알고리즘은 일반적인 태스크를 위해 설계된 스트라이드 알고리즘

을 기본으로 하여 멀티미디어 데이터를 스케쥴링 하기 위해 설계되었다. RRPSSA 알고리즘은 연속 미디어의 특성을 반영하기 위해 한 쌍의 파라메타 (*P*, *C*)를 도입하였다. *P* 는 태스크의 주기, *C* 는 태스크의 실행시간을 의미한다.

알고리즘의 진행 과정에서 스트라이드 알고리즘과 같이 세 개의 파라메타를 사용한다. 티켓, 스트라이드, 패스가 그것이다. 티켓은 태스크의 자원에 대한 권리를 나타내고 식 2.1과 같은 값을 가진다.

$$Ticket_k = \frac{C_k}{P_k} \qquad (2.1)$$

식 2.1에 사용된 기호는 다음과 같다.

C_k : 태스크 k의 실행시간

P_k : 태스크 k의 주기

스트라이드는 스케쥴러로부터 선택받는 간격을 나타내고 티켓 값의 역수인 값을 가진다. 패스는 스케쥴러의 다음 선택을 위한 가상 시간 인덱스를 나타내고, 초기에 스트라이드와 같은 값을 가진다. 스케쥴러는 패스 값이 가장 작은 태스크를 선택하고, 선택된 태스크의 패스 값은 스트라이드 값만큼 증가하게 된다.

2.2 비율조정기

RRPSSA의 가장 중요한 개념인 비율조정기는 태스크가 한 주기 안에서 자신의 지분보다 더 많은 자원을 할 당 받지 못하도록 해 준다. 비율조정기는 다음 식 2.2와 같다.

$$\frac{\sum_{i=start}^{current} Alloc_k(i)}{\Delta T} \le \frac{C_k}{P_k} \quad (k \in the\ set\ of\ tasks) \qquad (2.2)$$

식 2.2에서 사용한 기호는 다음과 같다.

$\sum_{i=start}^{current} Alloc_k(i)$: 시작에서부터 현재까지 태스크 k가 받은 총 CPU 할당량

ΔT : 시작에서부터 현재까지 시간

C_k : 태스크 k의 실행 시간

P_k : 태스크 k의 주기

2.3 태스크의 선정

스케쥴러는 다음과 같은 과정에 의해 서비스 할 태스크를 선정한다.

1) 스케쥴러는 패스 값이 가장 작은 태스크를 선택하여 비율조정기 만족 여부를 검사한다.
2) 비율조정기를 만족하면, 태스크를 선정하고, 비율조정기를 만족하지 않으면 나머지 태스크들 중에서 1) 과정부터 되풀이 한다.
3) 선정된 태스크의 패스 값을 수정한다.

다음 표 1과 같은 태스크를 가정하자.

(표 1) RRPSSA 스케쥴링 대상 태스크들

태스크 번호	실행 시간	주기
1	1	6
2	1	3
3	1	4

표 1의 태스크들은 다음 표 2와 같은 순서로 실행된다.

2.4 비율조정 비례지분 스케쥴러의 문제점

RRPSSA는 멀티미디어 데이터를 지원하기에는 다음과 같은 문제점이 있다.

첫째, RRPSSA 알고리즘을 이용하는 시스템은 과부하 상태에서 성능이 점진적으로 저하되지 않는다. RRPSSA 알고리즘은 스케쥴러로부터 서비스 받지 못한 시간의 회수 부분에 있어서는 공정성을 유지하는 것처럼 보이지만, 서비스 받아야 할 시간에 대한 서비스 받지 못한 시간의 비율면

(표 2) RRPSSA의 수행 과정

	태스크 1 :	태스크 2 :	태스크 3
티 켓	0.17 :	0.33 :	0.25
스트라이드	6 :	3 :	4

시간	패 스			선택된 태스크
	태스크1	태스크2	태스크3	
1	6	6	4	2
2	6	6	8	3
3	12	6	8	1
4	12	9	8	2
5	12	9	12	3
6	12	9	12	0
7	12	12	12	2
8	18	12	12	1
...	...	...	...	...

에 있어서는 공정성을 유지하지 못한다. 한 주기 동안 실행 시간이 긴 태스크와 실행 시간이 짧은 태스크에게 있어서 서비스 받지 못한 한 퀀텀(quantum)의 시간이 미치는 영향이 같다고 말할 수는 없기 때문이다.

둘째, 이 알고리즘은 엄격한 비율조정기로 인하여 문맥교환에 따르는 오버헤드가 너무 크다. RRPSSA 알고리즘의 비율조정기는 태스크의 지분이 전체 수행시간 동안 효과적으로 분배될 수 있도록 지원해준다. 그러나, 비율조정기를 나타내는 식 2.2에서 보듯이, 매 스케줄 때마다 변하는 값인 ΔT로 CPU 할당량 $\sum_{i=start}^{current} Alloc_k(i)$를 나눔으로써 한번 스케줄 된 태스크는 다음 단계에서 스케줄 받기 어렵게 된다. 즉, 거의 매번 문맥교환이 발생하여 CPU 이용률을 떨어뜨리게 된다.

위에서 언급한 두 가지의 단점을 보완하고 멀티미디어 데이터를 스케줄링 하기 위해 스케줄러 설계 시 다음과 같은 사항들이 고려되어야 한다.

1) 과부하 상태에서도 성능이 점진적으로 저하되어야 한다.
2) 문맥교환 회수를 줄인다.
3) 자원 활용율의 저하를 야기시키는 입장제어를 사용하지 않는다.

3. 수정 비례지분 스케줄링 알고리즘 (MPSSA)

3.1 자원 할당

RRPSSA에서 연속미디어의 특성을 반영하기 위해 사용한 파라메타(P, C)를 MPSSA에서 역시 그대로 이용하여 사용하기로 한다. P 는 태스크의 주기를 나타내고, C 는 태스크의 한 주기 안에서의 실행 시간을 나타낸다.

스케줄링 진행 과정에서 두 가지의 파라메타를 사용한다. RRPSSA에서 티켓, 스트라이드, 패스 세 가지의 파라메타를 사용하여 스케줄링 할 태스크를 선정하였지만, MPSSA에서는 티켓, 패스 두 개의 파라메타만으로도 스케줄링 할 태스크를 선정하기에 부족함이 없다.

티켓은 태스크가 할당받아야 할 자원에 대한 권리를 나타내고 아래 식 3.1과 같이 표현될 수 있다.

$$Ticket_k = \begin{cases} \dfrac{\frac{C_k}{P_k}}{\sum_{i=1}^{n}\frac{C_i}{P_i}} & \text{if } \sum_{i=1}^{n}\frac{C_i}{P_i} > 1 \\ \dfrac{C_k}{P_k} & \text{if } \sum_{i=1}^{n}\frac{C_i}{P_i} \le 1 \end{cases} \quad (3.1)$$

식 3.1에서 사용된 기호는 다음과 같다.

k : 태스크 k
C_k : 태스크 k의 실행 시간
P_k : 태스크 k의 주기

식 3.1에서 보이듯이 과부하 상황에서의 티켓 값과 저부하 상황에서의 티켓 값은 서로 다르다. 식 3.1에서 $\sum_{i=1}^{n}\frac{C_i}{P_i} > 1$ 은 과부하 상황을 의미

하고, $\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$ 은 저부하 상황을 의미한다. 즉, CPU 이용률의 합이 1보다 크면 과부하 상황이고, 1보다 적으면 저부하 상황이다. 저부하 상황에서 태스크는 자신의 주기동안 실행되어야 할 실행 시간의 비율만큼의 지분을 가지지만, 과부하 상황에서는 모든 태스크들의 지분이 저부하 상황에서보다 줄어들게 된다.

패스는 스케쥴러의 다음 선택을 위한 가상 시간 인덱스를 나타낸다. 패스 값은 매 시간 퀀텀마다 다음 식 3.2에 의해 결정된다.

$$Pass_k(t) = Ticket_k - \frac{\sum_{i=start}^{current} Alloc_k(i)}{\triangle T} \quad (3.2)$$

식 3.2의 기호는 다음과 같다.

t : 현재시간

$\sum_{i=start}^{current} Alloc_k(i)$: 시작부터 현재까지 태스크에 자원이 할당된 시간 퀀텀의 수

$\triangle T$: 시작부터 현재까지 시간

스케쥴링이 시작되는 초기에는 현재 할당된 시간의 퀀텀의 수가 0이므로 패스 값은 티켓 값과 동일하다. 스케쥴링이 진행되면서 패스는 태스크에 할당되어야 하는 자원의 양을 나타내는 티켓 값에서 현재까지 할당받은 자원의 양을 뺀 값을 가진다. 즉, 패스는 앞으로 할당받아야 할 자원의 양을 나타낸다.

3.2 태스크 선택

스케쥴러는 앞으로 할당받아야 할 자원의 양이 가장 큰 태스크, 즉, 패스 값이 가장 큰 태스크를 선택하여 서비스 해 주는 것을 기본 원칙으로 한다. 선택하는 과정에서 우선권 정책과 비율조정기가 사용된다.

우선권 정책이란 바로 전 시간의 퀀텀에서 서비스 받은 태스크에게 자원에 대한 우선권을 부여해 주는 것이다. 우선권 정책은 RRPSSA의 스케쥴링 과정에서 빈번하게 발생하는 문맥교환의 회수를 줄인다.

비율조정기는 태스크들이 주어진 기간 내에 자신의 지분보다 더 많은 자원을 할당받지 못하도록 조정해 준다. 스케쥴링 과정에서 어떤 태스크를 서비스해 주기 전에 비율조정기를 만족하는지의 여부를 먼저 검사한다. 비율조정기를 만족할 경우에 태스크를 서비스하고, 만족하지 않는다면 다른 태스크들 중 패스 값이 큰 태스크를 선택하여 다시 비율조정기 만족 여부를 검사하게 된다. 비율조정기는 아래 식 3.3과 같다.

$$Pass_k(t) > 0 \quad (3.3)$$

식 3.3에 사용된 기호는 다음과 같다.

k : 태스크 k

t : 현재 시간

식 3.3에서 패스 값이 0보다 크다는 것은 태스크에게 할당된 자원의 양이 원래 지분보다 작으므로 할당받아야 할 양이 아직 더 남아 있음을 의미한다.

태스크를 선택하는 전체적인 과정은 다음과 같다.

1) 티켓, 패스 값을 설정한다. 티켓 값은 태스크들의 주기와 계산시간으로 부터 계산되고, 패스 값은 초기에 할당받은 자원의 양이 없으므로 티켓값과 동일하다.
2) 스케쥴러는 패스 값이 가장 큰 태스크를 선택한다.
3) 선택한 태스크의 비율조정기 만족여부를 검사한다.
4) 태스크가 비율조정기를 만족하면 서비스를 해주고, 비율조정기를 만족하지 않으면 나머지

태스크들을 대상으로 과정 2)부터 반복한다.

5) 태스크들의 패스 값을 수정하고 우선권을 부여하여 과정 3)부터 반복한다.

본 알고리즘에서는 스케줄링 과정에서 우선권 정책을 이용하여 문맥교환의 회수를 감소시킨다. 또한, 티켓 값 설정 시 과부하 상황을 고려하여 값을 결정하고, 패스 값 수정시에도 선택된 태스크의 패스 값만 조정하는 것이 아니라 모든 태스크들의 패스 값이 상대적으로 조정되므로 과부하 상태에서 성능이 점진적으로 저하되는 것을 기대할 수 있다. 물론, 패스 값을 수정하는 과정에서의 연산회수는 RRPSSA보다 많다. 연산 회수에 따른 overhead에 관한 연구는 앞으로 계속해서 연구되어야 할 것이다.

다음 표 3.1과 같은 태스크를 가정하자.

(표 3) MPSSA 스케줄링 대상 태스크들

태스크	실행 시간	주기
A	1	5
B	2	4
C	2	6

(표 4) 수정된 비례지분 스케줄러의 예

	태스크 A:	태스크 B :	태스크 C
티 켓	0.19 :	0.48 :	0.32
시간	패스 A	패스 B	패스 C	선택된 태스크
1	0.19	0.48	0.32	B
2	0.19	-0.52	0.32	C
3	0.19	-0.02	-0.18	A
4	-0.14	0.15	-0.01	B
5	-0.06	-0.02	0.07	C
6	-0.01	0.08	-0.08	B
7	0.03	-0.02	-0.01	A
8	0.09	0.06	0.04	B
9	-0.06	-0.02	0.07	C
10	-0.03	0.04	-0.01	B
11	-0.01	-0.02	0.02	C
…	…	…	…	…

세 개의 태스크 A, B, C가 스케줄링되는 과정이 다음 표 4에 나타난다. 표 4에서 태스크 A, B, C는 각각의 패스 값에 의해 스케줄러로부터 선택되고, 스케줄러는 자원을 할당하는 데에 있어 공정성을 유지한다.

4. 모의 실험 결과

모의 실험은 저부하 상황과 과부하 상황에서 이루어졌다. 실험 결과를 RRPSSA와 비교하였으며 저부하 상황에서는 문맥교환의 횟수를 비교하였고, 과부하 상황에서는 성능 저하 형태와 문맥교환의 회수로 성능을 평가하였다.

4.1 저부하 상황

다음 표 5는 저부하 상황에서 실험 대상이 된 태스크들의 주기와 실행 시간을 보여준다.

(표 5) 저부하 상황에서 스케줄링 대상 태스크들

태스크	실행 시간	주기
A	32	100
B	17	80
C	23	75

표 5의 태스크들의 CPU 이용률의 합은 0.8392로 1보다 작으므로 저부하상황이다. 이 태스크들을 대상으로 RRPSSA와 MPSSA 스케줄링 알고리즘을 적용한 결과 발생한 문맥교환 회수를 표 6에서 비교하였다. 실험은 1000퀀텀의 시간동안 실시하였다.

(표 6) 1000퀀텀 시간에서의 문맥교환의 회수(저부하)

스케줄러	문맥교환의 수
RRPSSA	720
MPSSA	678

표 6에서 본 논문에서 제시하는 MPSSA 알고리

즘을 적용했을 경우 RRPSSA 알고리즘보다 문맥교환의 회수가 6%정도 감소하였음을 알 수 있다.

4.2 과부하 상황

과부하 상황에서 역시 MPSSA와 RRPSSA를 비교하였다. 문맥교환의 회수와 성능저하의 형태면에서 비교를 하였다.

성능저하 형태는 손실 비율(loss rate)로 알 수 있다. 손실 비율은 태스크의 한 주기동안 스케쥴러로부터 서비스 받지 못한 시간의 비율이며, 다음 식 4.1로 표시되어 질 수 있다.

$$Loss\ Rate_k = \frac{L_k}{C_k} \qquad (4.1)$$

식 4.1에서 사용된 기호는 다음과 같다.

k : 태스크 k

L_k : 태스크 k의 서비스 받지 못한 시간

C_k : 태스크 k의 실행 시간

손실 비율의 범위가 작다는 것은 손실 비율의 분산이 작다는 것을 의미하고, 이는 곧 과부하 상황에서 성능이 점진적으로 저하됨을 의미한다. 과부하 상황에서 성능이 점진적으로 저하될수록 데이터의 서비스 질이 좋아지게 된다.

다음 표 7은 스케쥴링 대상이 된 태스크들의 주기와 실행 시간을 나타낸다.

(표 7) 과부하 상황에서 스케쥴링 대상 태스크들

태스크	실행 시간	주기
A	30	100
B	19	87
C	32	70
D	25	93

표 7의 태스크들의 CPU 이용률의 합은 1.2444로 1보다 크므로 과부하 상황이다. 이 태스크들을 대상으로 RRPSSA와 MPSSA 스케쥴링 알고리즘을 적용한 결과 발생한 문맥교환 회수를 아래 표 8에서 비교하였다. 실험은 1000퀀텀의 시간동안 실시하였다.

(표 8) 1000퀀텀 시간에서의 문맥교환의 회수(과부하)

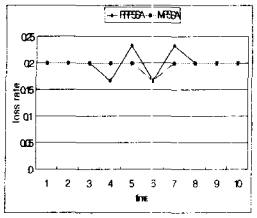
스케쥴러	문맥교환의 수
RRPSSA	992
MPSSA	949

표 8에서 본 논문에서 제시하는 MPSSA 알고리즘을 적용했을 경우 RRPSSA 알고리즘보다 문맥교환의 회수가 5%정도 감소하였음을 알 수 있다. 앞 절의 저부하 상황에서의 실험 결과와 표 8에 나타난 과부하 상황에서의 실험 결과에서 알 수 있듯이 제시한 MPSSA 알고리즘이 RRPSSA 알고리즘보다 문맥교환 회수면에서 보다 나은 성능을 보이고 있다.

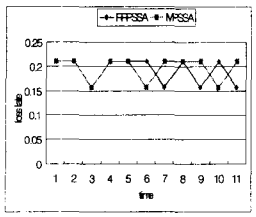
성능저하의 형태는 다음 그림들에서 알 수 있다. 그림 1은 RRPSSA에서의 태스크들의 서비스 받지 못한 시간 퀀텀의 수들의 범위와 MPSSA에서의 태스크들의 서비스 받지 못한 시간 퀀텀의 수들의 범위를 나타낸다. 그림 2에서는 RRPSSA에서의 태스크들의 손실 비율 범위와 MPSSA에서의 태스크들의 손실 비율 범위를 비교할 수 있다.

그림 1의 a, b, c, d 에서 태스크 A, B, C, D의 손실 비율면에서 RRPSSA 알고리즘과 MPSSA 알고리즘을 비교할 수 있다. 그림 1의 a, b, c, d 에서 보이듯이 태스크 A, C, D의 경우에 RRPSSA 알고리즘을 적용했을 때보다 MPSSA 알고리즘을 적용했을 때 손실 비율의 범위가 훨씬 작은 것을 알 수 있다. 손실 비율의 범위가 작다는 것은 훨씬 안정된 상태를 유지한다는 것을 의미한다. 반면, 태스크 B의 경우에는 그림 1.b에서 보이듯이 손실 비율의 범위면에서 RRPSSA 알고리즘과 MPSSA 알고리즘이 차이를 보이지 않는다.

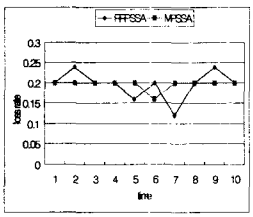
그림 2에서 태스크들의 손실 비율들이 함께 나



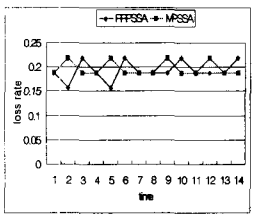
a. 태스크 A



b. 태스크 B

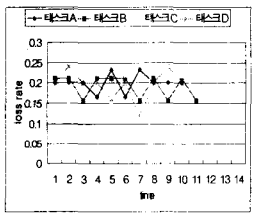


c. 태스크 C

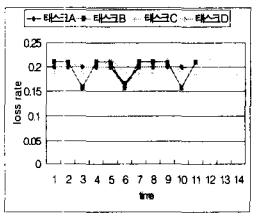


d. 태스크 D

(그림 1) 1000퀀텀동안 태스크별 손실 비율의 비교



a. RRPSSA



b. MPSSA

(그림 2) 1000퀀텀동안 알고리즘별 손실 비율의 비교

타난 그래프를 각각의 알고리즘별로 볼 수 있다. 그림 2.a의 RRPSSA 알고리즘의 경우 태스크 A, B, C, D들의 손실 비율의 전체 범위는 그림 2.b의 MPSSA 알고리즘의 경우보다 훨씬 크다. 특히, 그림 2.a의 시간 7에 나타난 현상은 RRPSSA 알고리즘의 비효율적인 면을 그대로 보여준다. 시간 7에서 태스크 A의 손실 비율이 커지는 반면, 태스크 D의 손실 비율은 작아진다. 이는 같은 시간대에 어떤 한 태스크에게는 자원이 많이 배분되고 있고, 다른 한 태스크에게는 자원이 적게 배분되고 있음을 의미한다. 따라서 자원 할당이 공평하게 이루어지지 않고 있음을 알 수 있다. 그림 2.b의 MPSSA

(표 9) 손실 비율 값

손실 비율		태스크 A	태스크 B	태스크 C	태스크 D
최대값	RRPSSA	0.233	0.211	0.219	0.240
	MPSSA	0.200	0.211	0.219	0.200
최소값	RRPSSA	0.167	0.158	0.156	0.160
	MPSSA	0.167	0.158	0.188	0.160
평균	RRPSSA	0.200	0.192	0.192	0.196
	MPSSA	0.198	0.197	0.197	0.196
분산	RRPSSA	0.000484	0.000715	0.000433	0.001227
	MPSSA	0.000109	0.000613	0.000211	0.000160

알고리즘의 경우에는 그림 2.a의 시간 7에서와 같은 현상이 보이지 않는다. 모든 태스크들의 손실 비율이 같은 방향으로 움직이고 있다. 한 태스크의 손실 비율이 크게 나타날 때 다른 태스크의 손실 비율이 작게 나타나는 현상이 없다. 태스크 A, B, C, D의 손실 비율의 그래프가 마치 하나의 그래프인 것처럼 겹쳐서 나타나게 된다. 이는 모든 태스크들에게 자원이 공평하게 할당되고 있음을 의미한다. 따라서 과부하 상황에서 성능이 급격하게 저하되는 것이 아니라 점진적으로 저하된다는 것을 알 수 있다. 과부하 상황에서 성능이 점진적으로 저하된다는 것은 RRPSSA 알고리즘보다 미디어 서비스 질이 좋다는 것을 의미한다.

표 9는 각 태스크의 손실 비율의 최대 값, 최소 값, 평균, 분산을 나타낸다. 표 9에서 RRPSSA 알고리즘보다 MPSSA 알고리즘의 경우에 분산이 태스크 A는 78%정도, 태스크 B는 15%정도, 태스크 C는 50%정도, 태스크 D는 85%정도 작은 것을 확인할 수 있다. 태스크별로 차이는 나지만 전반적으로 안정된 상태를 유지하는 것을 알 수 있다.

5. 스케줄링 알고리즘의 유연성 평가

5.1 유연성 평가 방법

일반적으로 스케줄링 알고리즘은 여러 가지 항

목으로 평가된다. 전통적인 운영체제에서의 스케쥴링 알고리즘을 평가하기 위해 태스크들의 응답시간이나 대기 시간 등을 비교하기도 하고, 경성 실시간 시스템에서의 스케쥴링 알고리즘은 모든 태스크들을 스케쥴링할 수 있는가에 대한 여부가 대단히 중요한 평가 항목이기도 하다. 제시한 MPSSA 알고리즘의 평가 항목으로 앞 절에서 문맥교환 회수와 과부하 상황에서의 성능 저하 형태면에서 실험한 결과 값을 제시하였다. 이러한 모든 목적을 충족하는 이상적인 스케쥴링 알고리즘이 존재하지 않으므로 응용 용도에 따라 알고리즘의 성능이 다를 수 있다.

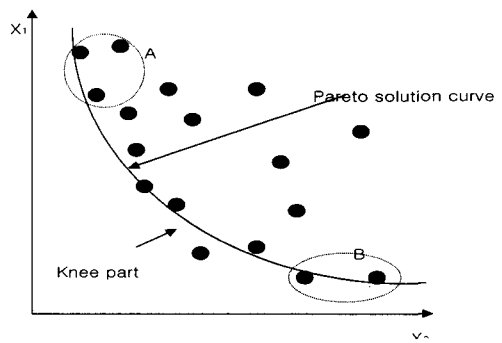
본 논문에서는 이러한 여러 목적을 가지는 스케쥴링 알고리즘을 유연성이라는 측면에서 평가하고자 한다.

여러 목적을 충족해야 하는 경우, 일반적으로 알려진 유연성을 평가하는 방법은 다음과 같다.

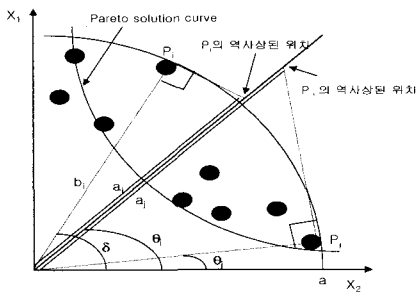
X_1과 X_2라는 두 가지의 목적을 가지는 2차원의 공간에서 스케쥴러의 결과 값들은 그림 3의 점으로 표시 될 수 있다.

그림 3의 Pareto solution curve의 Knee part 부분에 나타난 점들은 X_1과 X_2라는 두 가지의 목적을 충족시켜야 하는 상황에서 유연하게 대처하는 값들이다.

일반적으로 유연성은 "Pareto solution curve에 근접해서 나타나는 값들이 얼마나 많이 있는가"로 평가되어 왔다[10]. 그러나, 그림 3에 점선으로



(그림 3) 2차원 공간에 표시된 스케쥴링 결과 값



(그림 4) 스케쥴링 알고리즘의 적응성 평가

표시되어진 A 부분에 나타난 값들과 B부분에 나타난 값들을 같이 평가하기에는 무리가 있다.

A부분에 나타난 값들은 비록 Pareto solution curve 상에 있고, X_1목적에 근접하지만, X_2목적을 충족시킨다고 보기는 힘들다. 한편, B 부분에 나타난 값들 역시 Pareto solution curve 상에 있고, X_2목적에 근접하지만, X_1목적을 충족시킨다고 보기는 힘들다. 따라서 A 부분에 나타난 값들과 B 부분에 나타난 값들은 다르게 평가되어야 한다.

유연성을 평가하기 위해 그림 4에서 보이는 것처럼 모든 점들을 Pareto solution curve의 Knee part 부분의 방향으로 그어진 선 위로 역사상하였다.

동일한 선 위의 역사상된 점들은 원점으로부터의 거리측정을 쉽게 할 수 있다. 원점으로부터의 거리가 곧 스케쥴러의 유연성이 된다. 역사상을 이용하여 유연성 Z 를 식 5.1과 같이 나타낼 수 있다.

$$Z = \frac{\sqrt{\sum_{i=0}^{m} a_i^2}}{m} \qquad (5.1)$$

식 5.1의 a_i 는 다음과 같다.

$$a_i = \frac{b_i}{\cos(\theta_i - \delta)}$$

식 5.1의 m 은 스케쥴링 결과 값들의 개수이다.

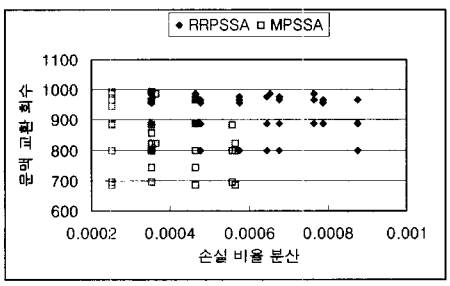
그림 4에서 두 점 Pi, Pj 가 역사상되어 Knee part 방향으로 그어진 선 위에 나타난 위치와 원점까지의 거리는 ai, aj 로 서로 다르다. ai 가 aj 보다 거리가 짧으므로 Pi 값에 대한 평가가 Pj 값에 대한 평가보다 높게 된다. 심지어, Pj 가 Pareto solution curve 상에 있다 하더라도 Pareto solution curve 상에 있지 않은 Pi 보다 낮게 평가된다. Pj 는 X2 의 목적은 잘 반영하고 있지만 X1 과는 거리가 멀다. 따라서 Pj 값은 Pi 보다 덜 유연하게 평가받는다.

5.2 평가 결과

제시한 MPSSA 알고리즘의 유연성을 평가하기 위해 실험을 하였다. RRPSSA 알고리즘과 비교를 하였고, 스케줄링의 여러 가지 목적중 과부하 상황에서의 성능 저하 형태와 문맥교환의 회수 두 가지에 중점을 두었다. 과부하 상황에서의 성능 저하 형태는 태스크들의 손실 비율들의 분산 값으로 측정하였다.

그림 5는 실험 결과이다. 세로 축은 문맥교환의 회수이고 가로축은 태스크들의 손실 비율의 분산 값이다. 각 알고리즘당 50회의 실험을 실시하였다. 50회 모두 과부하 상황으로 설정하였다. 각 실험에서 스케줄링 대상이 된 모든 태스크들의 손실 비율들의 분산 값과 문맥교환의 회수 값이 1회 실험 결과 값의 좌표이다.

그림 5에서 보이듯이 RRPSSA 알고리즘 수행



(그림 5) 알고리즘의 유연성 비교

결과로 나타난 값들은 Knee part에서 멀리 위치한 반면, MPSSA 알고리즘 수행 결과로 나타난 값들은 Knee part의 가까이에 위치한다. 유연성을 계산하기 위해 그림 5에 나타난 좌표 값들을 역사상하고, 거리 값을 측정하기 위해 각각의 축을 0 에서 1사이로 표준화하였다. 표준화한 결과로 나타난 값들의 거리를 측정한 결과 유연성 Z 의 값은 다음과 같이 나타났다.

$$\frac{Z_R}{Z_M} \simeq 1.5 \qquad (5.2)$$

여기서 Z_R은 RRPSSA의 유연성, Z_M은 MPSSA의 유연성이다. RRPSSA의 유연성과 MPSSA의 유연성의 비율 값에서 알 수 있듯이 Z_M 이 훨씬 작은 값을 가진다. 따라서 MPSSA 알고리즘이 유연성면에서 RRPSSA 알고리즘보다 나은 성능을 보인다고 평가 할 수 있다.

6. 결 론

이 논문은 멀티미디어 데이터를 위한 성능이 향상된 비례지분 스케쥴링 알고리즘을 제시하였다. 제시한 알고리즘은 다음과 같은 성능의 향상이 있었다.

1) 모든 태스크들의 손실 비율의 범위가 비슷하며, RRPSSA에 비해 손실 비율의 분산값이 작다. 이는 과부하상태에서 시스템의 성능이 점진적으로 저하됨을 의미한다.
2) 스케쥴링 수행 시 문맥교환 횟수가 RRPSSA보다 작다. 따라서 불필요한 문맥교환에 따르는 오버헤드를 줄일 수 있다.

또한 본 논문에서는 알고리즘의 유연성을 평가할 수 있는 방법을 제시하였다. 제시한 방법에 의해 문맥교환 회수와 손실 비율을 고려한 유연성

을 평가한 결과 제시한 알고리즘이 RRPSSA보다 좋은 결과를 보였다.

그 외에 MPSSA에서는 입장제어를 사용하지 않으므로 자원활용도를 높일 수가 있다.

향후 본 논문에서 제시된 알고리즘의 비례지분 수정을 위한 overhead에 대해 구체적인 값을 제시하고, 알고리즘을 Linux 시스템에 구현하여 실제 연속미디어를 이용한 성능을 평가를 실측 하고자 한다.

참 고 문 헌

[1] Ming-Syan Chen, "Stream Conversion to Support Interactive Video Palyout", Proc. of the IEEE, pp. 51, 1996.

[2] D. P. Anderson, "Metascheduling for continuous media", ACM Operating Systems Review, 25 (4):47, August, 1993.

[3] L. Sha, R. Rajkumar, and S. S. Sathaye, "Generalized rate monotonic scheduling theory", Proc. of the IEEE, pp. 68, 1994.

[4] C. M. Krishna, Kang G. Shin, "Real Time System", The McGraw-Hill Companies, Inc.

[5] D. B. Golub, "Operating system support for coexistence of real-time and conventional schedulings", Technical Report CMU-CS-94-212, Carnegie Mellon Univ., Pittsburgh, Pennsylvania, November, 1994.

[6] R. Govindan, D. P. Anderson, "Scheduling and IPC mechanism for continuous media", In Proceedings of the 13th ACM Symposium on Operating System Principle, pp. 68, October, 1991.

[7] Manhee Kim, Hyogun Lee, JooWon Lee, "A Proportional Share Scheduler for Multimedia Applications", Proc. of the IEEE, pp. 484, 1997.

[8] Carl A. Waldspurger, William E. Weihl, "Stride Scheduling: Deterministic Proportional Share Resource Management", Technical Memorandum MIT/LCS/TM-528, 1995.

[9] Carl A. Waldspurger, "Lottery and Stride Scheduling: Flexible Proportional Share Resource Management", phD thesis, MIT, September, 1995.

[10] J. Koda, et. al., "Estimation of flexibility of Generation planning under Uncertainty", Proceedings of IEEE of japan Power & Energy '90, PE-19, JAPAN, 1990.

◐ 저 자 소 개 ◑

김 현 철

1984년 숭실대학교 전자계산학과 졸업(학사)
1993년 숭실대학교 정보과학대학원 졸업(석사)
1996년 일본 이바라키대학교 대학원 시스템공학과 졸업(박사)
2000년~현재 : 경주대학교 컴퓨터전자공학부 교수
관심분야 : 시스템공학, etc.
E-mail : kimhc@kyongju.ac.kr

박 정 석

1981년 숭실대학교 전자계산학과 졸업(학사)
1983년 숭실대학교 대학원 전자계산학과 졸업(석사)
1996년 충북대학교 대학원 전자계산학과 졸업(박사)
1983년~1996년 한국원자력연구소 선임연구원
1996년~현재 : 국립청주과학대학 컴퓨터과학과 조교수
관심분야 : 시간데이터베이스, Stream 데이터베이스, etc.
E-mail : jspark@chongjunc.ac.kr