퍼지 추론을 이용한 사용자 적응적 프로세스 스케줄링

임성수° 조성배
연세대학교 컴퓨터과학과
lss@sclab.yonsei.ac.kr°, sbcho@cs.yonsei.ac.kr

User Adaptive Process Scheduling using Fuzzy Inference

Sungsoo Lim° Sung-Bae Cho
Dept. of Computer Science, Yonsei University

요 약

기존의 운영체제에서는 시스템이 프로세스의 종류를 알지 못하므로, 사용자가 명시하지 않은 서로 다른 종류의 프로세스에 모두 동일한 스케줄링 정책을 적용해 왔다. 따라서 한번 결정된 스케줄링 정책은 변화하는 환경에 적응하지 못한다는 단점이 있다. 본 논문에서는 리눅스 환경에서 프로세스들의 자원사용량을 근거로 각 프로세스를 일괄처리 프로세스, 대화식 프로세스, 실시간 프로세스로 분류하고, 각 분류에 대한 사용자 우선순위를 모델링하여 사용자의 성향에 맞게 프로세스에 우선순위를 부여하는 사용자 적응적 프로세스 스케줄링 기법을 제안한다. 이 방법은 사용자의 성향에 따라서 스케줄링 정책을 결정할 수 있으며, 여러 사용자에게 서로 다른 스케줄링 정책을 적용할 수 있다. 실험 결과 제안하는 방법의 유용성을 확인할 수 있었다.

1. 서론

프로세스 스케줄링의 목적은 다중 프로세스를 CPU가 적절히 배열해 줌으로써 보다 효율적으로 사용자에게 작업결과를 주는 것이다. 따라서 CPU 효율 및 처리율을 최대화하면서 사용자의 관점에서 원하는 작업에 대하여 최소화된 반환시간을 얻을 수 있어야 한다. 모든 시스템의 관점에서 발생하는 모든 프로세스들에게 공정한 CPU 시간을 할당하면서 CPU 사용율을 극대화하는 것이 유용하다. 그러나 세부적으로 일괄처리 프로세스는 높은 처리량과 짧은 반환시간이, 대화식 프로세스는 짧은 응답시간이, 그리고 실시간 프로세스는 주어진 시간 내에 일정량의 연산이 수행될 수 있도록 해야 한다[1].

그러나 이러한 스케줄링의 목표는 짧은 응답시간과 데드라인의 만족 등의 상호 모순이 존재한다. 따라서 기존의 스케줄링 기법들은 특정 스케줄링의 목표를 달성하거나, 특수한 환경에서 효율적으로 동작하는 방향으로 제안되어 왔다. 이는 시스템의 입장에서는 현재 실행중인 프로세스의 종류를 알지 못하기 때문이다.

본 논문에서는 리눅스 환경에서 프로세스의 특징을 추출하여 사용자가 이용하고 있는 프로세스를 일괄처리 프로세스, 대화식 프로세스, 실시간 프로세스로 구분하고, 각 프로세스에 대한 사용자의 우선순위를 모델링하여, 사용자 적응적 프로세스 스케줄링을 제공한다.

2. 프로세스 스케줄링

전통적인 프로세스 스케줄링 알고리즘은 일의 양이 적은 것을 우선적으로 수행하는 SIF(Shortest-Job-First) 스케줄링, 주어진 규칙에 따라서 프로세스의 우선순위를 결정하여 높은 우선순의를 갖는 프로세스를 수행하는 우선순위 스케줄링, 선점형 방식으로 시간 할당량이라고 하는 작은 단위의 시간동안 CPU를 돌아가면서 할당하는 RR(Round Robin) 스케줄링 등이 있다.

근래에는 컴퓨터의 성능과 네트워크 속도 및 대역폭의 증가에 따라서 멀티미디어 태스크를 지원하기 위한 연구들이 진행되어 왔다[2, 3]. Yavatkar는 멀티미디어 태스크의 수행시간을 예측하기 어렵다는 전제하에 비율에 기반한 적응적 우선순위 스케줄링을 제안하였고[2], Nieh와 Monica에 의해서 제안된 SMART(Scheduler for Multimedia And Real-Time application)는 솔라리스 유닉스 환경에서 구현되었다[3]. SMART는 실시간 태스크와 일반적인 태스크가 동시에 수행될 수 있도록 잘 고려되어 있다. 이것은 태스크의 각 특성을 분류하고, 중요도와 긴급성에 따라서 스케줄링을 해주기 때문이다.

3. 제안하는 방법

제안하는 방법은 크게 프로세스 분류단계와 프로세스 우선순위 결정단계로 구분된다. 프로세스 분류단계는 프로세스의 시스템 콜 호출 종류 및 빈도와 CPU 사용량을 가지고 퍼지 규칙을 통해 일괄처리 프로세스, 대화식 프로세스, 실시간 프로세스로 분류한다. 그리고 우선순위 결정단계에서는 사용자 모델링을 통해 앞의 세 종류의 프로세스에 대한 사용자 가중치를 구하고, 프로세스 분류 정보를 이용하여 해당 프로세스의 우선순위를 결정한다. 본 논문에서는 프로세스 분류 및 우선순위 결정을 위해서 퍼지 추론을 이용하였다. 퍼지 추론은 개념적으로 이해하기가 쉽고, 다른 인공지능 기법에 비해 연산량이 적다는 장점이 있다.

3.1 프로세스 분류

프로세스 분류는 그림 1과 같이 퍼지 추론을 통해서 세 종류의 프로세스에 대한 퍼지 소속도를 얻는다. 그림 1에서 사용된 프로세스 로그는 시스템 콜의 종류와 이전 시스템 콜 발생 시점에서 지금의 시스템 콜이 발생할 때까지의 프로세스의 CPU 사용 시간을 하나의 순서쌍으로 하여 구성된다.

일괄처리 프로세스는 컴파일러, 과학계산 등과 같이 CPU 연산을 주로 필요로 하는 프로세스이므로 다른 프로세스에 비해 CPU 사용량이 상대적으로 많으며, 대화식 프로세스는 문서 편

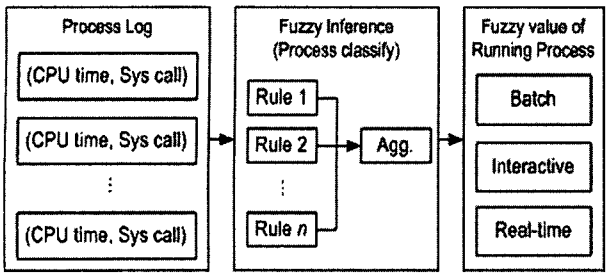

그림 1. 퍼지 프로세스 분류기

집기와 같이 사용자와 빈번하게 상호작용을 하는 프로세스로 다른 프로세스에 비해 입출력에 관련된 시스템 콜이 빈번하게 발생한다. 그리고 실시간 프로세스는 비디오, 오디오 등과 같은 멀티미디어 작업으로 자신의 연산이 끝난 후 다른 프로세스에게 제어를 넘기기 위해서 sleep에 관한 시스템 콜이 많이 발생한다. 따라서 본 논문에서는 표 1과 같은 퍼지 규칙을 정의하여, 프로세스 분류를 위한 퍼지 추론에 사용한다.

표 1. 프로세스 분류를 위한 퍼지 규칙

if pairs of system call {sys_read(), sys_write()} is many, then interactive process if CPU using times(per ticks) is many, then batch process if system call {sys_nanosleep()} is many, then real-time process

3.2 프로세스 우선순위 결정

리눅스에서 대부분의 프로세스는 SCHED_OTHER 스케줄 정책을 사용하며, 스케줄링 우선순위는 task 구조체의 nice값과 밀접한 관계가 있다. 즉, 프로세스의 nice 값이 크면 낮은 우선순위를 나타내며, 반대로 작으면 높은 우선순위를 나타낸다. 본 논문에서는 사용자 모델링을 통한 가중치의 조절로 프로세스의 nice 값을 변경하여 프로세스의 우선순위를 조절한다.

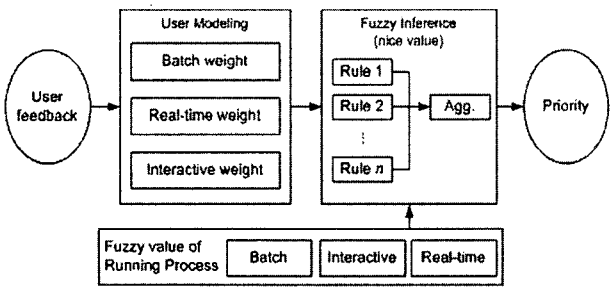

그림 2. 사용자 모델링과 프로세스 우선순위 결정

그림 2는 사용자 모델링과 프로세스 우선순위 결정방법을 보여준다. 사용자 피드백은 사용자가 컴퓨터를 사용하고 로그아웃할 때에 사용자에게 세 종류의 프로세스에 대한 만족도를 0~10점 사이의 점수를 부여함으로써 수행된다. 사용자로부터 피드백이 들어오면 표 2와 같은 수식을 이용하여 사용자의 각 프로세스 분류에 대한 가중치를 갱신한다.

표 2. 피드백을 통한 사용자 가중치 갱신

$$w_{batch} = w_{batch} + w_{batch} \times \frac{f_{batch}}{10}$$

$$w_{inter} = w_{inter} + w_{inter} \times \frac{f_{inter}}{10} \quad (w: \text{가중치}, f: \text{피드백})$$

$$w_{rt} = w_{rt} + w_{rt} \times \frac{f_{rt}}{10}$$

이렇게 얻어진 각 프로세스 분류에 대한 사용자 가중치와 3.1절에서 얻은 프로세스 분류에 대한 퍼지 소속도를 이용하여 퍼지 추론을 통해서 nice 값이 결정된다. 표 3은 nice 값을 결정하기 위한 퍼지 규칙을 보여준다.

표 3. 프로세스 우선순위 결정을 위한 퍼지 규칙

if (process is interactive process) and (user interactive weight is high),	**then** nice is low
if (process is interactive process) and (user interactive weight is middle),	**then** nice is middle
if (process is interactive process) and (user interactive weight is low),	**then** nice is high
if (process is batch process) and (user batch weight is high),	**then** nice is low
if (process is batch process) and (user batch weight is middle),	**then** nice is middle
if (process is batch process) and (user batch weight is low),	**then** nice is high
if (process is real-time process) and (user real-time weight is high),	**then** nice is low
if (process is real-time process) and (user real-time weight is middle),	**then** nice is middle
if (process is real-time process) and (user real-time weight is low),	**then** nice is high

4. 실험 및 결과

본 논문에서는 리눅스 커널 2.4.25를 수정하여 제안하는 방법을 구현했다. 실험은 사용자 그룹을 일괄처리 작업 사용자, 문서편집 사용자, 멀티미디어 사용자의 총 3개로 구분하고, 각 사용자 그룹별로 10명의 피험자들이 일반적인 환경(리눅스 커널 2.4.25)과 제안하는 방법이 적용된 환경에서 주어진 프로세스(일괄처리 프로세스 10개, 대화식 프로세스 1개, 실시간 프로세스 1개)를 실행한 후, 제안하는 방법이 얼마나 자신이 속한 그룹에 적합한지를 1~5점으로 평가하였다. 그리고 정량적인 평가를 위해 단위시간당 CPU 사용량과 CPU 선점 횟수를 비교하였다.

실험을 위해서 실행하는 프로세스는 일괄처리 프로세스 10개와 대화식 프로세스 1개, 실시간 프로세스 1개이다. 일괄처리 프로세스를 많이 실행하는 이유는 CPU 점유율을 높여 모든 프로세스가 원활히 작동되지 않도록 하기 위함이다. 실험에 있어서의 가정은 일괄처리 작업 사용자 그룹은 일괄처리(batch) 프로세스에 대해서 높은 CPU 할당량을 요구하고, 문서편집 사용자 그룹은 대화식 프로세스에 대해서 빠른 답변시간을 요구하며, 멀티미디어 사용자 그룹은 실시간 프로세스에 대해서 QOS의 보장, 즉 높은 CPU 선점 횟수를 요구한다는 것이다.

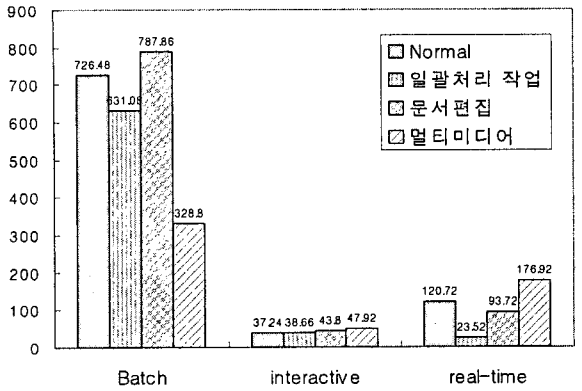

그림 3. 단위시간당 CPU 선점 횟수

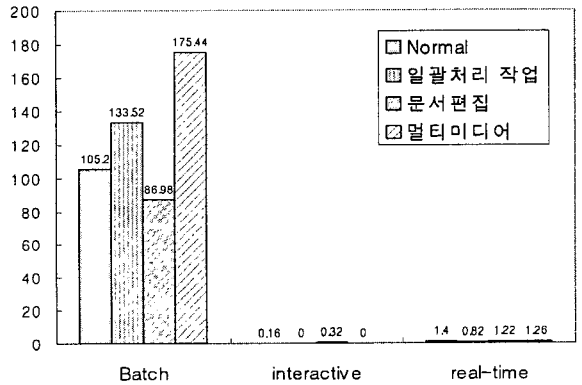

그림 4. 단위시간당 CPU 사용량

그림 3은 단위시간당 CPU 선점 횟수를 보여주고 그림 4는 단위시간당 CPU 사용량을 보여주고 있다. 일괄처리 작업 사용자 그룹(133.52)은 Batch 프로세스에 대해서 일반적인 방법(105.2)보다 높은 CPU 사용량을 나타냈다(그림 5). 그러나 멀티미디어 사용자 그룹(175.44)이 일괄처리 사용자 그룹에 비해서 더욱 높은 CPU 사용량을 보였다. 이는 멀티미디어 사용자 그룹이 전반적으로 CPU 선점 횟수가 다른 그룹에 비해서 월등히 낮아 프로세스 로그를 계산하는데 드는 오버헤드가 적게 들었기 때문이다(프로세스의 로그의 개수는 CPU 선점 횟수에 비례함).

멀티미디어 사용자 그룹(176.92)은 실시간 프로세스에 대해서 다른 그룹에 비해서 높은 CPU 선점 횟수를 나타냈다(그림 4). 이는 실시간 프로세스가 주기적으로 CPU를 선점하여 필요한 연산을 수행하여 다른 그룹에 비해 멀티미디어 자료가 끊기지 않고 수행되었음을 보여준다.

문서편집 사용자 그룹은 대화식 프로세스에 대해서 다른 그룹에 비해 가장 높은 CPU 사용량(0.32)을 보여주고, CPU 선점 횟수에 있어서는 멀티미디어 사용자 그룹(47.92) 다음으로 높은 값(43.8)을 보여준다. 그러나 문서편집과 같은 대화식 프로세스의 경우는 CPU 사용량이 극히 작고, 사용자의 타이핑 속도에 따라 CPU 선점 횟수가 달라지므로 정량적으로 평가하기 어렵다. 그림 5의 사용자 평가 결과를 보면, 제안하는 방법이 좀 더 좋은 평가를 받음을 알 수 있다.

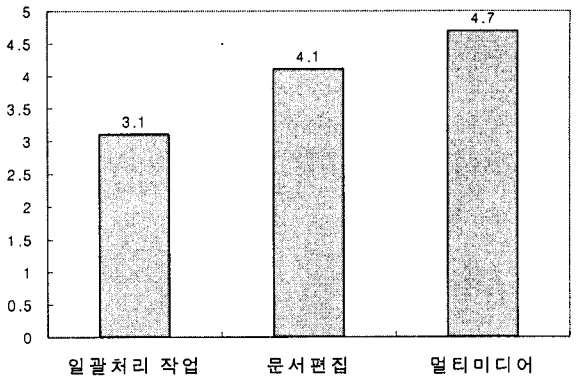

그림 5. 사용자 평가

그림 5는 사용자 평가에 따른 비교분석을 그래프로 보여주고 있다. 평가는 비슷하다고 생각된 경우 3점을 나쁘면 1이나 2점을, 그리고 비교해서 좀 더 빠르게 실행되었다고 생각되면 4, 5점을 주도록 하였다. 분석결과 일괄처리 작업은 3.1점을, 실시간 프로세스는 4.7의 높은 만족도를, 상호작용 프로세스는 4.1의 만족도를 얻을 수 있었다. 일괄처리 작업은 사람이 직관적으로 판단하기 어려웠기 때문에 상대적으로 낮은 만족도를 얻었다.

5. 결론

본 논문에서는 프로세스의 특징을 추출하여 사용자가 이용하고 있는 프로세스를 일괄처리 프로세스, 대화식 프로세스, 실시간 프로세스로 구분하고, 사용자 모델링을 통한 사용자 적응적인 프로세스 스케줄링 기법을 제안했다. 이러한 방법은 기존의 CPU 버스트 또는 프로세스 순서에 따라 일괄적으로 스케줄링하는 방법과는 달리 사용자별로 사용자의 관점에서 스케줄링된다는 장점이 있다. 사용자가 우선적으로 처리해 주기를 바라는 프로세스에 높은 우선순위를 부여함으로써 사용자에 적합한 스케줄링 정책을 제공한다. 이 작업을 위해서는 어떤 사용자인지의 정보를 얻어야 하므로 시스템 로그아웃 도중에 한번은 사용자 정보를 주어야 한다는 제약점도 가지고 있지만, 이 문제는 커널을 컴파일해서 스케줄링 정책을 바꾸어야 하는 것보다는 훨씬 간단하고 쉬운 작업이고 이 간단한 작업으로 사용자별 스케줄링을 제공할 수 있다.

참고 문헌

[1] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel*, O'REILLY, 2003.

[2] R. Yavatkar and K. Lakshman, "A CPU scheduling algorithm for continuous media applications," *Lecture Notes in Computer Science,* vol. 1018, pp. 210-213, 1995.

[3] J. Nieh and Monica S. Lam, "The design, implementation and evaluation of SMART: A scheduler for multimedia applications," *Proc. of 16th ACM Symposium on Operating Systems Principles*, pp. 184-197, 1997.