프로세스 모니터링 기법에 기반한 DVS 시스템

이준희° 차호정
연세대학교 컴퓨터과학과
{jhl,hjcha}@cs.yonsei.ac.kr

A DVS System based on Process Monitoring Technique

Joonhee Lee° Hojung Cha
Dept. of Computer Science, Yonsei University

요 약

본 논문에서는 프로세스 모니터링 기법에 기반한 DVS 시스템을 제안한다. 이상적인 DVS 시스템은 응용프로그램의 수정없이 자동으로 수행되어야 하며 프로세스의 QoS를 고려해야 한다. 본 논문은 이를 위해 본 연구의 이전 논문에서 제시한 Kernel Control Path를 모니터링하여 주기적 프로세스의 QoS관련 정보를 추출할 수 있는 기법을 기반으로 DVS 시스템을 제안한다. 제안한 DVS 시스템은 리눅스 운영체제상에서 실제 구현하였으며 관련 연구와의 비교를 위해 관련연구도 구현하여 실험하였다. 이를 통해 제안한 DVS 시스템이 주기적 프로세스의 QoS를 보장하면서 전력소비를 최소화할 수 있음을 밝힌다.

1. 서론

최근 범용 운영체제상에서 프로세스 상태에 따른 동적전압변경(DVS) 기법 연구가 활발히 이루어지고 있다. 프로세스 상태에 따른 동적전압변경 기법이란 커널내부에서 프로세스 상태를 모니터링하여 프로세서의 전압을 자동으로 변경해주는 기술을 말한다. 이는 기존 응용프로그램의 변경이 필요 없고, 사용자에게 별도의 조작을 요구하지 않는 점에서 큰 장점을 지닌다. 이를 위해서는 운영체제의 커널상에서의 적절한 모니터링 기법과 동적전압변경 알고리즘 개발이 필요하다.

이와 관련된 초기 연구는 프로세서 사용률을 기반으로 한 DVS 기법 개발을 중심으로 수행되었다. 그러나, 초기 연구는 급한 수행이 필요한 프로세스들을 구별하기 어려우며, 프로세스의 QoS에 대한 적절한 고려가 없다는 문제가 있다[1]. 이러한 초기 연구의 문제점을 개선하기 위해 태스크기반 스케쥴링 기법이 제시되었다. 그러나, 이러한 기법은 확률값을 기반으로 하므로 태스크의 데드라인 정보가 정확하지 않으며, 각 태스크가 요구하는 미래의 프로세서 사용률을 알기 어렵다. 이를 해결하기 위해 기존 연구는 응용프로그램의 특성에 따른 프로세서 사용률에 초점을 맞추었다. 범용운영체제의 경우, 응용프로그램에 대한 어떠한 정보도 알 수가 없다. 따라서 범용운영체제상의 기존 연구는 응용프로그램의 특성에 따라 QoS를 보장해야 하는 응용프로그램과 이를 보장할 필요가 없는 응용프로그램으로 나누고 이 둘에 대한 적절한 적응적 DVS 정책을 제공하려 노력하였다.

이 문제를 해결하는 바람직한 방법은 커널상에서 QoS 보장이 필요한 응용프로그램을 자동으로 구별하고 해당 응용프로그램의 QoS 관련정보를 추출하여 DVS 알고리즘을 적용하는 것이다. 이와 관련된 기존 연구는 Vertigo[2]와 PACE[1], GRACE-OS[3]가 있다. 그러나, 기존 연구는 오버헤드가 크고 커널 및 다른 응용프로그램 수행으로 인한 영향을 크게 받아 QoS 보장이 필요한 응용프로그램을 적절히 구별하지 못하며, 데드라인 정보가 정확하지 않아 실제 시스템에 적용하기 어렵다. 본 연구는 기존 연구의 문제점을 해결하기 위해 이전 논문에서 Kernel Control Path를 모니터링하여 자동으로 QoS보장이 필요한 프로세스를 구별하고 해당 프로세스의 QoS관련정보를 추출할 수 있는 기법을 제시하였다[4]. 본 논문에서는 이를 기반으로 한 DVS 시스템을 제시한다. 제시한 DVS 시스템은 응용프로그램의 QoS를 보장하면서 전력소비를 최소화시킬 수 있다.

논문의 구성은 다음과 같다. 2장에서 프로세스 모니터링 기법에 기반한 DVS 시스템에 대해 설명하고, 3장에서 실험을 통해 제시한 DVS 시스템이 기존의 DVS 시스템에 비해 응용프로그램의 수정없이 프로세스의 QoS를 보장하면서 전력소비를 최소화할 수 있음을 실험을 통해 밝힌 후, 4장에서 결론을 맺는다.

2. DVS 시스템

2.1 프로세스 모니터링 구조

본 논문에서 제시하는 DVS 시스템에 기반이 되는 프로세스 모니터링 구조의 특징은 다음과 같다. 첫째, 기존 응용프로그램에 수정을 가할 필요없이 프로세스의 수행상태를 커널에서 모니터링하여 자동으로 프로세스의 주기성과 프로세서 사용률을 확인한다. 둘째, 프로세스 주기성 확인시 멀티프로세싱을 지원하는 범용운영체제상에서 다른 프로세스 및 커널 동작의 영향을 적게 받는다. 셋째, 멀티프로세싱을 지원하는 범용운영체제상에서 QoS 보장을 필요로 하는 프로세스를 자동으로 구별할 수 있다. 넷째, QoS 보장을 필요로 하는 프로세스의 데드라인값과 주기내의 평균수행시간값을 추

• 본 연구는 정보통신연구진흥원에서 지원하는 정보통신기초기술연구지원사업으로 수행하였음 (과제번호 : C1-2003-A1-2000-0240)

출할 수 있다. 제시하는 프로세스 모니터링 구조(PMon)는 그림과 같다.

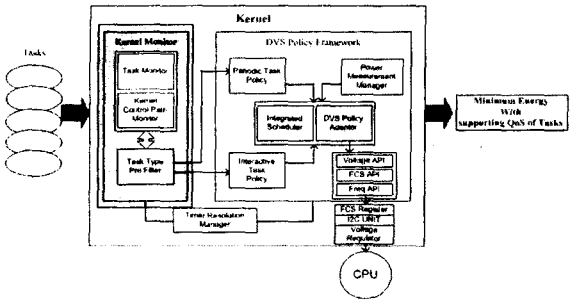

그림 1 : PMon Framework

그림 1에서 PMon은 Task Monitor와 Kernel Control Path Monitor를 가진다. Task Monitor는 프로세스의 프로세서 사용상태를 모니터링하며, Kernel Control Path Monitor는 프로세스가 QoS를 보장하기 위해 커널에 요청한 특정 루틴을 처리하는 Kernel Control Path를 모니터링한다. Task Monitor와 Kernel Control Path Monitor에서 추출된 정보는 Task Type Pre Filter에서 QoS 보장이 필요한 프로세스와 그렇지 않은 프로세스를 구별하고, QoS 보장이 필요한 프로세스의 경우 데드라인값과 주기내의 평균수행시간을 추출하는데 사용된다. Task Type Pre Filter에서 분석된 결과에 따라 분류된 프로세스들은 각각 Periodic Task Policy와 Interactive Task Policy, Integrated Scheduler, DVS Policy Adaptor에서 스케쥴링 기법과 통합되어 실제 시스템에 적용된다. 이와 같이 PMon은 커널상에서 자동으로 프로세스의 프로세서 사용률을 확인하고 QoS 보장이 필요한 프로세스를 구별한 후, 해당 프로세스의 QoS관련 정보를 추출하여 DVS 알고리즘에 적용가능토록 하는 역할을 한다. 보다 구체적인 사항은 저자의 이전 논문에 기술되어 있다[4].

2.2 PMon에 기반한 DVS 알고리즘

PMon을 기반으로 한 DVS 알고리즘은 다음과 같다.

$$F = \{ f_1, f_2, \cdots, f_n \} \text{ (단, } n > 1,\ f_m < f_{m+1}) \quad (1)$$

$$f_m = (freq_m,\ volt_m) \quad (2)$$

if $f_m < f_{m+1}$
then $freq_m < freq_{m+1}$ and $volt_m < volt_{m+1}$

DVS 알고리즘에서 사용하는 프로세서의 동적전압변경 단계의 집합을 F라 정의하고, 각 동적전압단계를 이라 f_m한다. f_m은 다시 프로세서 주파수인 $freq_m$와 이에 적합한 프로세서 전압인 $volt_m$를 속성으로 가진다. PMon은 F를 QoS 보장이 필요한 주기적 프로세스(pe_i)와 QoS 보장이 필요없는 프로세스(ie_i)로 나누고, pe_i의 QoS 조건 $q(pe_i)$을 '데드라인 평균(d_{avg}) < 해당 주기내의 수행시간(t_i)'으로 정의한다. 이를 구현하기 위해 t_i는 해당주기내의 수행완료시점을 명시적으로 알 수 없으므로 과거 수행시간의 평균(t_{avg})으로 계산한다. d_{avg}와 t_{avg}는 다음과 같이 정의한다.

$$d_{avg} = \frac{\sum_{i=n-k+1}^{n} d_i}{k} \quad (3)$$

$$t_{avg} = \frac{\sum_{i=n-k+1}^{n} t_i}{k} \quad (4)$$

k값은 실험결과 5로 설정하였을 때 가장 좋은 성능을 보였다. 이와 같은 정의를 기반으로 PMon에 사용한 DVS 알고리즘은 100ms의 간격마다 다음과 같이 수행한다.

if $\frac{t_{avg}}{d_{avg}} \times 100 > c_1$ then $f_{cur} = f_{cur+1}$

if $c_2 < \frac{t_{avg}}{d_{avg}} \times 100 < c_1$ then $f_{cur} = f_{cur}$

if $\frac{t_{avg}}{d_{avg}} \times 100 < c_2$ then $f_{cur} = f_{cur-1}$

그림 2 : PMon에 기반한 DVS 알고리즘

그림 2에서 f_{cur}는 현재 적용되어 있는 프로세서의 주파수와 전압이다. DVS 알고리즘은 d_{avg}과 t_{avg}의 비율이 c_1을 넘을 경우에는 동적전압단계를 한단계 높여주고, c_2이하일 경우에는 동적전압단계를 한 단계 낮추어준다. 실험에서는 c_1과 c_2 값을 각각 65와 25로 설정하였다. 이를 통해 PMon을 기반으로 한 DVS 알고리즘은 주기적 프로세스의 주기를 넘지 않으면서 주기내의 수행시간을 해당 주기까지 늘려준다. 이상의 PMon과 이에 기반한 DVS 알고리즘으로 구성된 DVS 시스템은 주기적 프로세스의 QoS를 훼손하지 않으면서 전력소비를 최소화할 수 있다.

3. 실험

DVS 시스템과 관련한 실험은 멀티프로세싱을 지원하는 범용 시스템에서 응용프로그램이 정보를 제공하지 않을때 제시한 DVS 시스템이 QoS 보장을 필요로 하는 주기적 프로세스의 QoS를 훼손하지 않으면서도 전력소비를 줄일 수 있음을 보인다. 이를 위해 관련 연구중 본 연구와 가장 비슷한 GRACE-OS[3], Automatic Performance Setting 기법[5], Vertigo[2]를 실제 구현하고 비교실험을 수행하였다.

실험은 Xscale PXA250 프로세서를 장착한 Lubbock Board에서 수행하였다. 응용프로그램은 mpeg_play, splay를 실행시킨 후 각각의 제시한 DVS 알고리즘 및 모니터링 기법에 따라 DVS를 수행, CPU에서 소비한 전력 및 전압을 Fluke 멀티미터를 사용하여 측정하였다. mpeg_play는 QoS 보장여부를 확인하기 위해 1초에 수행해야 하는 프레임수를 지정한 후, mpeg_play 코드를 수정하여 실제 1초마다 지정한 프레임수를 모두 수행하였는지 여부를 확인하여 데드라인 미스율을 확인하였다. 프로세서의 주파수와 전압은 FCS(Frequency Changing Sequence)를 사용하여 100MHz/1.3V, 200MHz/1.4V, 300MHz/1.5V, 400MHz/1.6V의 4단계로 변경하도록 API를 작성하여 사용하였다.

실험에 앞서 Vertigo와 Automatic Performance Setting 기법 및 GRACE-OS는 본 논문의 DVS 시스템과 달리 응용프로그램의 주기성을 적절히 구별하지 못하므로 임의로 해당 프로그램을 구별하도록 커널을 수정하여 실험하였다. mpeg_play 실험에서는 1150Kbps로 인코딩된 MPEG1 데이터를 사용하였으며, splay 실험에서는 44Kbps로 인코딩된 MP3 데이터를 사용하였다.

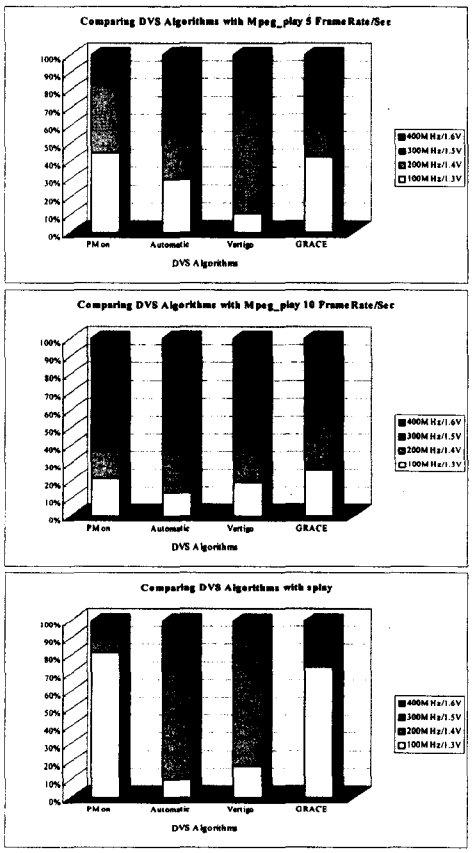

그림 3 : 각 DVS 알고리즘에 따른 주파수 및 전압할당률 비교

그림 3은 각 DVS 알고리즘 및 모니터링 기법을 사용하여 mpeg_play(5 frame, 10 frame), splay를 수행하였을 때의 프로세서 주파수와 전압의 사용비율을 그래프로 나타낸 것이다. 그림 3에서 보듯이 제시한 DVS 시스템은 3가지 경우에 있어서 다른 DVS 시스템에 비해 낮은 주파수 및 전압을 사용한 비율이 더 크며, 소비전력 또한 다른 DVS 시스템에 비해 낮음을 보여준다.

표 1 : Energy Consumption (단위 : Watt X Sec)

	No-DVS	PMon	Auto	Vertigo	GRACE
mpeg_play(5frame)	36.23	26.24(28%)	30.68(16%)	30.57(16%)	31.39(14%)
mpeg_play(10frame)	30.24	21.92(28%)	23.46(23%)	22.70(25%)	23.63(22%)
splay(mp3)	19.51	8.41(56%)	14.68(25%)	11.68(40%)	11.17(43%)

((%)는 DVS를 적용하지 않았을때와 비교한 전력절감률)

표 1은 그림 3의 실험시 Fluke 멀티미터를 사용하여 전력을 측정한 후 에너지 계산 공식에 따라 각 DVS 알고리즘 수행으로 소비된 에너지를 비교한 것이다. 표 1에서 보듯이 제시한 DVS 시스템이 기존의 DVS 시스템에 비해 전력소비가 작다. 다음은 응용프로그램 수행시 각 DVS 시스템 적용에 따른 데드라인 미스율을 표로 작성한 것이다.

표 2 : Deadline Miss Rate

	PMon	Auto	Vertigo	GRACE
mpeg_play(5frame)	0%	3.1%	0.8%	0.8%
mpeg_play(10frame)	1.5%	9.2%	4.6%	12.3%

표 2와 같이 제시한 DVS 시스템은 기존의 3가지 DVS 시스템에 비해 데드라인 미스율이 매우 작다. 이상의 실험결과에서 보듯이 PMon을 기반으로 한 DVS 시스템은 QoS 보장이 필요한 주기적 응용프로그램 수행시 해당 응용프로그램을 자동으로 구별하고 QoS 관련 정보를 추출하여 DVS 알고리즘을 적용하므로 프로세스 QoS를 보장함과 동시에 전력소비를 최소로 할 수 있다.

4. 결론

본 논문에서는 프로세스 모니터링 기법에 기반한 DVS 시스템을 제시한다. 제안한 DVS 시스템은 본 연구의 이전 논문에서 제시한 응용프로그램의 정보없이 자동으로 QoS 보장을 필요로 하는 프로세스를 구별하고 해당 프로세스의 주기와 평균수행시간을 추출할 수 있는 프로세스 모니터링 구조를 기반으로 한다. 이를 기반으로 제시한 DVS 시스템은 프로세스의 QoS를 보장하면서 소비전력을 최소화할 수 있다. 본 연구는 관련연구와 함께 제시한 DVS 시스템을 리눅스에 실제 구현하였다. 그리고 비교실험을 통해 기존의 DVS 시스템보다 제시한 DVS 시스템이 QoS 보장을 필요로 하는 프로세스의 QoS를 훼손하지 않으면서 소비전력을 최소화할 수 있음을 밝혔다. 제안한 DVS 시스템은 임베디드 기기와 같은 저전력을 필요로 하는 기기에 적합하다. 추후에는 DVS 시스템을 스케쥴러와 통합하여 범용운영체제에서 비주기적 프로세스까지 고려한 DVS 시스템을 개발할 예정이다.

참고문헌

[1] Lorch, J. and Smith, A. J. "Using user interface event information in dynamic voltage scaling algorithms," Proc. of the 11th IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, Orlando, FL, pp. 46~55, Oct 2003.

[2] K. Flautner and T. Mudge, "Vertigo: Automatic Performance-Setting for Linux", Proc. of 5th Symp. Operating Systems Design & Implementation, pp. 105-116, Dec 2002.

[3] W. Yuan and K. Nahrstedt, "Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems," Proc. of 19th ACM Symposium on Operating Systems Principles, Bolton Landing, NY, Oct 2003.

[4] 이준희, 김효승, 차호정, "DVS 적용을 위한 프로세스 모니터링 기법," 한국정보과학회 2003년 추계학술발표대회 논문집, 2003년 10월.

[5] K. Flautner, S. Reinhardt, and T. Mudge. "Automatic performance setting for dynamic voltage scaling," Proc. of the 7th Conference on Mobile Computing and Networking, pp. 260~271, July 2002.