The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 17, No. 4, pp.149-154, Aug. 31, 2017. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2017.17.4.149
JIIBC 2017-4-19

가변 샘플링 기법을 이용한 프로그램 성능 개선

Performance Improvement of Application Programs using an Adaptive Sampling Method

조정호*, 서효중**

Jeongho Jo*, Hyo-Joong Suh**

요 약 스마트폰과 같은 모바일 기기는 입력으로부터 결과를 확인하기까지의 반응시간이 짧을수록 기기의 체감성능이 높아진다. 애플리케이션의 반응시간을 짧게 할 수 있다면 이러한 체감성능을 높일 수 있게 된다. 이에 관련하여 온디맨드 거버너는 주파수를 바꿀 때 샘플링 간격으로 인한 약점이 있어 프로그램의 실행을 느리게 할 수 있다. 이에 본 논문에서는 애플리케이션의 실행에 맞춰 온디맨드 거버너의 샘플링 주기를 가변시키는 기법을 제안한다. 기법의 적용에 따라 실행시간과 전력소모량을 비교한 결과, 프로그램의 실행 체감성능에 연관되는 초기실행시간 부분에 대하여 3.34% 개선되는 것을 확인할 수 있었으며, 가변적 샘플링 주기에 따른 기기의 전체적인 전력소모량의 차이는 유의미하게 나타나지 않았다.

Abstract Performance of the mobile devices, such as Smartphones, is sensible by the early-stage of the execution of the applications. To addressing this issue, the dynamic frequency scaling by the ondemand governor has an inherent weakness by the sampling period that may induces some delay in the execution time of the applications. In this paper, we propose an adaptive sampling method that varying the sampling period of the ondemand governor in accordance with the execution of the applications. By the experiment result, the proposed method outperforms 3.34% in early-stage of the execution time that impacts the sensible performance, and exhibits negligible differences in terms of the energy consumption.

Key Words : Smart Devices, Execution Time, Android, Governor, Sampling Period

Ⅰ. 서 론

스마트폰이나 태블릿 PC와 같은 스마트 기기는 다양한 앱을 제공한다. 여러 앱을 사용하므로 앱의 실행시간은 스마트 기기 사용자들에게 가장 민감한 부분이다. 앱의 반응시간을 줄일 수 있다면 스마트 기기는 사용자들에게 성능이 좋다고 인식될 것이다[1].

앱의 실행을 빠르게 하기 위한 간단한 방법은 하드웨어를 고성능화하는 것이다[2]. 예를 들어 2016년 삼성전자와 애플은 내부 저장소를 eMMC(Embedded Multi Media Card)[3]대신 각각 UFS(Universal Flash Storage)[4], NVMe(Non-Volatile Memory Express)[5]를 적용하여 입출력을 개선하였다. 하지만 이러한 접근은 비용이 크다는 단점이 있다.

다른 방법으로는 하드웨어 및 소프트웨어 통합적 접근방법이 있다. 이러한 방법은 하드웨어적 고성능화와 더불어 소프트웨어를 이용하여 필요에 따라 전력소모와 성능을 조절하고 있다. 통합적 접근에서 사용하는 핵심

*정회원, 가톨릭대학교 대학원 컴퓨터공학과
**정회원, 가톨릭대학교 컴퓨터정보공학부(교신저자)
접수일자: 2017년 4월 25일, 수정완료: 2017년 6월 29일
게재확정일자: 2017년 8월 11일

Received: 25 April, 2017 / Revised: 29 June, 2017 /
Accepted: 11 August, 2017
**Corresponding Author: hjsuh@catholic.ac.kr
CSIE Dept. of the Catholic Univ. of Korea

기술로 DVFS(Dynamic Voltage and Frequency)가 있다. DVFS는 동적으로 전압과 클록 주파수를 조절하는 기법이다[6].

리눅스 커널(Linux Kernel)은 CPU에 대하여 DVFS를 적용하고 있는데, 거버너(Governor)를 이용하여 CPU의 클록 주파수를 조절하고 있다. 대표적인 거버너로는 온디맨드(Ondemand) 거버너를 꼽을 수 있는데 온디맨드 거버너는 샘플링을 통해 CPU의 클록 주파수를 조절한다. 따라서 큰 부하를 만나면 바로 클록 주파수를 올리는 것이 아니라 샘플링 후 클록 주파수를 올리게 되어 샘플링 주기만큼의 지연이 발생하게 된다. 이에 본 논문에서는 온디맨드 거버너의 샘플링으로 인한 지연을 줄여 앱의 실행시간을 단축시키고자 한다.

본 논문의 구성을 다음과 같다. 2장에서는 앱의 실행시간을 줄이는 관련 연구들에 대해 서술한다. 3장은 문제점과 제안하는 기법에 대해 기술하며, 4장에서는 제안한 기법에 대한 성능 및 전력소모 비교실험을 다룬다. 마지막 5장에서는 전체적인 결론과 향후 연구에 대해 서술한다.

II. 관련 연구

사용자 체감 위주의 성능개선은 다양하게 시도된 바 있다. 최상위에 있는 인터랙티브 프로세스, 즉 현재 사용하는 앱을 인지하여 CPU의 우선순위를 높이거나 입출력 버퍼를 인위적으로 늘리는 등으로 스마트 기기의 자원을 해당 앱에 많이 할당하여 현재 사용하는 앱의 반응속도를 빠르게 하였고[7], 싱글 스레드 앱을 대상으로 했기 때문에 멀티 스레드 앱 환경에서는 성능향상을 기대하기 어렵다고 지적되기도 하였다[8]. 따라서 사용자가 스마트폰의 화면을 터치하면 해당 앱의 스레드를 파악하고 이와 연관된 스레드들의 우선순위를 연쇄적으로 증가시켜 반응속도를 높이는 방법을 제시하고 있다.

안드로이드에서는 실행을 종료한 앱의 재실행시의 동작성능을 높이기 위해 가능한 메모리에서 앱을 유지하는 방법을 사용하고 있다. 따라서 메모리 관리 체계를 갖추고 있으며, 이와 관련하여 OOMK(Out Of Memory Killer)와 LMK(Low Memory Killer) 두 가지의 메모리 관리 체계를 사용하고 있다. OOMK는 리눅스 커널의 메모리 관리 기법으로 메모리 부족(Out Of Memory)이 발생하면 가장 많은 메모리를 사용하고 있는 프로세스를 종료시켜 메모리 공간을 확보한다. 하지만 OOMK가 어떤 프로세스가 종료시킬지 모르기 때문에 안드로이드에서는 LMK를 도입하여 일반적인 상황에서는 LMK를 수행하게 하고, 만약 LMK를 수행하고 나서도 메모리가 부족한 상황이 되면 OOMK가 작동하도록 만들었다. LMK의 경우 LRU(Least Recently Used) 기반으로 작동하여 가장 최근에 사용된 것을 중점적으로 메모리에 남겨두기 때문에 사용자의 행동 특성과 맞지 않는 부분이 존재한다. 이러한 LRU와 맞지 않는 사용자의 시공간적 특성을 이용하여 LMK의 작동을 보조하여 앱이 메모리에서 해제되는 것을 막는다[9]. 이러한 보조는 앱이 메모리에 적재되어 있어 재실행했을 때 메모리에 적재하는 과정이 줄어 반응속도를 좋게 만든다.

III. 제안하는 기법

1. 안드로이드 거버너와 문제점

AOSP(Android Open Source Project)[10]에 등록되어 있는 거버너는 퍼포먼스(Performance), 파워세이브(Powersave), 온디맨드(Ondemand), 콘서버티브(Conservative), 유저스페이스(Userspace), 인터랙티브(Interactive)이다[11]. 이 중 일반적으로 흔히 사용하는 거버너는 온디맨드 거버너로 일정 기간을 주기로 하여 CPU의 이용률을 샘플링하여 이용률에 따라 CPU의 클록 주파수를 바꾸는 작업을 한다.

CPU의 클록 주파수가 낮은 상태에서 큰 부하의 작업을 할 경우 온디맨드 거버너는 이용률을 샘플링한 후 주파수를 바꿔야하기 때문에 바로 클록 주파수를 올리지 못하는 지연이 발생하게 된다.

2. 가변 샘플링 주기 기법

온디맨드 거버너는 샘플링을 필요로 하므로 샘플링으로 인한 지연을 없애는 것은 불가능하다. 하지만 샘플링 주기를 변경하는 것은 가능하다. 그림 1에서 온디맨드 거버너의 샘플링 주기는 매개변수 중 하나인 sampling_rate가 결정한다[12]. sampling_rate의 값은 양의 정수 값을 가지는데 이는 us 단위의 샘플링 주기로 이 값을 바꾸면 주기가 바뀐다. 예를 들어 sampling_rate가 100,000이면 샘플링 주기는 100,000us(100ms)이다. AOSP에서 기본으로 설정해 놓은 sampling_rate는

100,000이니 100ms 주기로 샘플링을 한다. 만약 50ms로 샘플링 주기를 바꿀 경우 100ms보다 빠르게 클록 주파수를 변화시킨다.

본 논문에서는 앱의 실행을 감지하여 샘플링을 주기를 가변적으로 바꾸는 기법을 제안한다. 예를 들어, 앱을 실행할 때는 100ms에서 50ms로 바꾸어 온디맨드 거버너가 더 빠르게 부하에 반응할 수 있도록 만들고, 실행이 끝나면 다시 100ms로 돌려놓는다.

샘플링 주기가 짧아짐에 따라 전력소모량이 늘어날 것이 우려된다. 이에 대해서는 4장 실험 및 결과에서 자세히 다룬다.

```
root@GT-I9100:/sys/devices/system/cpu/cpufreq/ondemand
# ls
down_differential
flexrate_duration
flexrate_enable
flexrate_forcerate
flexrate_num_effective_usage
flexrate_request
freq_step
ignore_nice_load
io_is_busy
powersave_bias
sampling_down_factor
sampling_rate
sampling_rate_min
up_threshold
```

그림 1. 온디맨드 거버너 매개변수
Fig. 1. Parameters of ondemand governor

Ⅳ. 실험 및 결과

이 장에서는 3장에서 제안한 가변 샘플링 주기 기법을 실제 기기에 적용하여 성능비교 실험을 진행할 것이다. 또한, 샘플링 주기를 줄여 더 빈번한 샘플링을 수행하기 때문에 추가적인 전력소모가 발생할 수 있으므로 전력소모 비교도 제시할 것이다. 표 1은 실험환경을 정리해놓은 것이다.

표 1. 실험환경
Table 1. Experiment environments

Computer	CPU	Intel® CoreTM i3-3220 3.30GHz
	RAM	8GB
	OS	Windows 8.1 64bit
Target	Device	Samsung GALAXY S II
	AP	Samsung Exynos 4210
	Kernel	Linux 3.0.64
	OS	Cyanogenmod 11 (Based on Android 4.4 Kitkat)

대상 기기에 킷캣을 사용한 이유는 실험 당시에 ART(Android Runtime)의 호환성이나 안정성이 좋지 않아 달빅VM(DalvikVM)을 사용하는 마지막 버전인 킷캣으로 진행하였다. 또한, 표 2에서 1/3정도인 29.2%로 상당한 기기가 킷캣을 사용하고 있음을 보여주고 있다. CM(Cyanogenmod)[13]은 AOSP의 소스코드를 수정한 것으로, 실험 기기에 적용되는 킷캣기반의 CM11을 설치하여 실험을 진행하였다.

표 2. 안드로이드 버전별 점유율(2016.12)[14]
Table 2. Android version share(2016.12)[14]

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	1.7%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.6%
4.1.x	Jelly Bean	16	6.0%
4.2.x		17	8.3%
4.3		18	2.4%
4.4	KitKat	19	29.2%
5.0	Lollipop	21	14.1%
5.1		22	21.4%
6.0	Marshmallow	23	15.2%

1. 성능 비교 실험

성능 비교 실험을 위하여 두 가지 앱을 제작하였다. 하나는 앱의 실행을 감지하여 가변적으로 온디맨드 거버너의 샘플링 주기를 변화시키기 위한 백그라운드 서비스이다. 앱이 실행될 때는 대상 기기의 기본 샘플링 주기인

100 ms에서 50 ms로 감소시키고 앱이 종료될 때 다시 100 ms로 돌려놓도록 만들었으며, 제안방법의 핵심 기능을 수행한다.

다른 하나는 성능 부하용으로 사용할 앱으로, CPU 부하를 파라미터로 받아 생성하도록 하였다. 부하의 정도를 지정함에 따라 실행 시 CPU 부하의 시간 길이가 다르게 나타나도록 하였다. 실험에 사용한 대상 기기는 듀얼코어 CPU를 탑재하고 있어 모든 CPU에 부하가 걸리도록 멀티 스레드로 만들었다.

실험은 CPU 부하가 작은 상태에서 실험용 데모 앱을 실행하여 실행부터 화면에 나타날 때까지 걸린 시간(이하 실행시간)을 기록하는 방식으로 진행하였다. 위 과정을 각 반복횟수(부하길이)별로 기법을 적용한 것과 하지 않은 것별로 각각 10회씩 반복하였다.

그림 2에서 막대그래프는 반복횟수에 따라 기법을 적용한 것과 적용하지 않은 것의 실행시간 변화를 비교한 것이고, 꺾은선 그래프는 상대 차이를 나타낸 것이다. 제안 기법을 적용했을 경우 모든 경우에서 실행시간이 2~5% 정도 감소한 것을 확인할 수 있다. 하지만 부하가 지속되는 시간이 길어짐에 따라 초기에 샘플링에 따른 지연이 차지하는 비율이 상대적으로 적어지게 되므로 개선 비율이 줄어드는 것으로 보임을 확인 할 수 있다.

그림 2. 앱의 실행시간
Fig. 2. Execution times of applications

그러므로 안드로이드가 기본적으로 앱을 실행하는 동안 걸리는 시간, 다시 말해 모든 앱을 실행할 때 공통으로 소요하는 초기실행 시간을 '기본실행시간'으로 정의한다. 이 부분은 특히 앱의 초기반응성에 대해 중요한 부분으로써 사용자는 프로그램의 전체 실행시간이 아닌 사용자 인터랙션까지의 초기 실행시간을 체감실행성능으로 느끼게 된다[9]. 앱의 기본실행시간을 비교하기 위하여 초기실행시간을 초과한 임의로 생성한 부하에 걸린 시간을 제거하여 비교한 것이 그림 3이다. 기본실행시간의 경우 어느 앱이든 상관없이 앱을 실행하기 위해 걸리는 시간이기 때문에 그림 2처럼 부하의 정도에 따라 나누지 않고 가변적 샘플링 주기 기법이 적용된 것과 아닌 것으로만 분리하여 정리하였다.

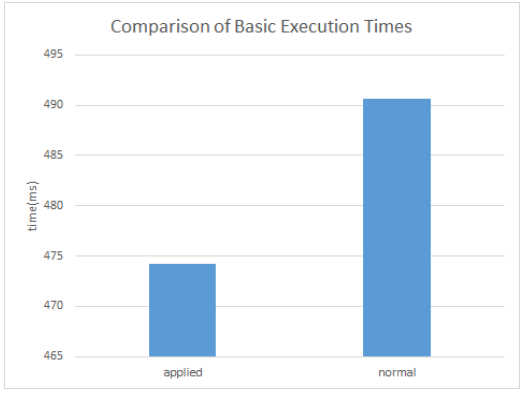

그림 3. 앱의 기본실행시간
Fig. 3. Basic execution times

실험 결과 본 제안기법에 따른 앱의 실행시간 전체의 차이는 그리 크지 않으나, 기본실행시간은 3.34% 개선됨을 관찰할 수 있었으며, 이는 앱 전체의 부하정도에 무관하게 가변샘플링 기법이 특히 초기 실행시간에 효과적임을 나타내고, 이는 특히 사용자 체감성능에 직접 연관됨을 파악할 수 있다.

2. 소비전력 비교

온디맨드 거버너에서 더 잦은 샘플링은 전력소모량에 변화를 나타낼 수 있다. 이는 제한된 전력량을 가진 스마트 기기에 매우 중요한 문제이다.

샘플링 주기변화에 따른 전력소모 변화량은 그림 4와 같은 환경에서 측정되었다. 전력소모 측정은 하드커널(Hardkernel) 사의 오드로이드 스마트파워(ODROID SmartPower, 이하 스마트파워)를 이용하였다. 스마트파워는 전력을 측정해주고, 컴퓨터로 일정 기간의 전력 변화를 저장하게 된다.

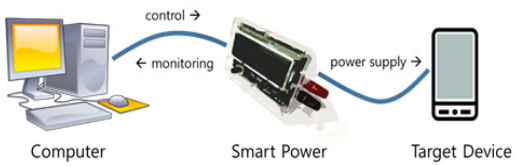

그림 4. 전력소모 측정
Fig. 4. Measurement of power consumption

우선 대상 기기에 별도의 부하를 주지 않은 상태에서 2분 동안의 전력소모량을 측정하는 것을 기본으로 하였다. 이 조건에서 온디맨드 거버너의 기본 샘플링 주기인 100ms를 포함하여 10, 50, 200ms로 각각 변화를 주어 10회 반복하여 전력소모량을 측정하였다. 측정한 2분 동안의 전력소모량 중에서 앞의 1분간은 스마트폰의 조작에 따른 영향을 받게 되기에 제거하고, 뒤쪽 1분 동안의 전력소모량을 비교하였다. 표 3은 샘플링 주기별 10회 실험한 결과의 평균을 정리한 것이다.

표 3. 샘플링 주기별 1분 동안의 평균 전력소모량
Table 3. Average power consumption of 1 min

Sampling rate	10ms	50ms	100ms	200ms
average (mWh)	16.041	16.029	16.126	16.014
difference (mWh)	-0.085	-0.097	*	-0.112
difference(%)	-0.53	-0.60	*	-0.69

실험 결과 샘플링 주기에 따른 전력소모 차이는 0.085 ~ 0.112mWh이고 0.53 ~ 0.69%로 모두 1% 미만이다. 샘플링 주기의 장단에 따른 소모 정도의 선형성은 나타나지 않았으며, 짧은 주기의 샘플링에도 보다 적은 전력소모가 나타나는 등 샘플링 주기에 따른 전력소모 차이보다는 기타 가변적 요소가 전력소모를 결정하고 있음을 확인할 수 있다[15].

하지만 샘플링을 위한 프로세서의 대기상태와 깨움은 다양한 기능 유닛을 내장하고 있는 애플리케이션 프로세서의 내장 유닛에 대한 부가적 처리를 유발할 수 있으며, 따라서 샘플링 주기를 짧게 지속할 경우 입출력시스템에 대한 부하와 전력소모를 유발시킬 수 있다[16].

V. 결론 및 향후 연구

온디맨드 거버너의 경우 CPU 부하에 맞춰 동적으로 클록의 주파수를 변경하기 위한 주기적인 샘플링을 수행한다. 따라서 급격히 부하가 증가하여도 이에 따른 클럭 상승에 최대 100ms의 지연 대응을 나타낼 수 있다. 또한 이러한 지연은 사용자 체감 성능 측면에 있어서 손실을 나타낼 수 있다.

이에 본 논문에서는 앱의 실행을 감지하여 샘플링 주기를 짧아지도록 조절하여 CPU 부하를 더 자주하는 기법을 제안하였다.

제안하는 기법을 이용하여 가변적으로 샘플링 주기를 바꿨을 경우 앱 실행 시 부하의 길이에 따라 2 ~ 5%의 시간이 감소하였으며, 앱의 기본실행시간만 고려했을 경우 3.34%의 시간 단축을 확인할 수 있었다. 또한 샘플링 주기의 순간적 단축은 시스템 전체 전력소모량에 영향이 나타나지 않음을 확인할 수 있었다.

본 논문에서는 다양한 기기에 대해 실험한 결과가 아니며, 샘플링과 어플리케이션 프로세서의 내부 유닛에 대한 대기상태와의 상관관계를 보다 상세히 조사할 필요가 있다. 따라서 보다 다양한 기기 및 어플리케이션 프로세서에 대한 실험을 후속 연구로 준비하고 있다.

References

[1] Jungae, Kim, Eui-young, Cho, "Smartphone Usage Experience of College students", The Journal of The Institute of Internet, Broadcasting and Communication, Vol. 16, No. 3, pp. 187-201, Jun 2016.
DOI: http://dx.doi.org/10.7236/JIIBC.2016.16.3.187

[2] Jongbok Lee. "A Study in the Effects of DRAM on The Microprocessor Performance", The Journal of The Institute of Internet, Broadcasting and Communication, Vol. 17, No. 1, pp. 219-224, Feb 2017.
DOI: https://doi.org/10.7236/JIIBC.2017.17.1.219

[3] Jim Cooke, "The inconvenient truths of NAND flash memory", Flash Memory Summit, 2007.

[4] "Introduction to the Universal Flash Storage

Assocation", Universal Flash Storage Association, 2013.

[5] K. Eshghi, R. Micheloni. "SSD architecture and PCI Express interface", Inside Solid State Drives (SSDs). Springer Netherlands, 2013. 19-45.

[6] J. Pouwelse, K. Langendoen, H. Sips, "Dynamic voltage scaling on a low-power microprocessor", Proc. Of the Intl. Conf. Mobile computing and Networking, pp. 251-259, 2001.

[7] S. Bae, J. Kim and Y. I. Eom, "Enhancing Interactivity in Mobile," Journal of KIISE : Computing Practices and Letters, vol. 18, no. 7, pp. 533-537, 2012.

[8] J. Lee, S. Huh and S. Hong, "Improving Interactivity via Chained Priority Boosting for Android Smartphone," Proceedings of the Korean Society of Computer Information Conference, vol. 21, no. 1, pp. 1-2, 2013.

[9] J. Sung, J. H. Kim, S. Y. Hwang and H.-J. Suh, "Memory Management Policy of Smartphones based on Access Points," Proceedings of Korea Computer Congress 2014, pp. 408-410, 2014.

[10] Android Open Source Project, Available: https://source.android.com/.

[11] W.-Y. Liang, P.-T. Lai, "Design and Implementation of a critical speed-based DVFS mechanism for the android operating system", Embedded and Multimedia Computing (EMC), 2010 5th International Conference on. IEEE, 2010.

[12] V. Pallipadi and A. Starikovskiy, "The Ondemand Governor," Proceedings of 2006 Linux Symposium, vol. 2, pp. 223-238, 2006.

[13] CyanogenMod Android Community Operating System, Available: https://www.cyanogenmod.org/.

[14] Dashboards | Android Developers, Available: https://developer.android.com/about/dashboards/index.html#Platform.

[15] Jeongho Jo, Reducing Execution Time of Smart Device Apps by Varying Sampling Rate, MS Thesis, Catholic Univ. of Korea, 2017.

[16] R. W. Ahmad, A. Gania,S. H. A. Hamid, F. Xiab, M. Shiraza, "A Review on mobile application energy profiling: Taxonomy, state-of-the-art, and open research issues", Journal of Network and Computer Applications vol.58, pp.42-59, 2015.

저자 소개

조 정 호(정회원)

• 2014년 : 가톨릭대학교 컴퓨터정보공학부 졸업(학사). 2017년 가톨릭대학교 컴퓨터공학과 졸업(석사). 관심분야는 임베디드시스템, 사물인터넷(IoT) 등.

서 효 중(정회원)

• 1992년 : 서울대학교 학사
• 1994년 : 서울대학교 석사
• 2000년 : 서울대 컴퓨터공학박사.
• 2003 년 ~ 현재 : 가톨릭대학교 컴퓨터정보공학부 교수. 관심분야는 컴퓨터 시스템, 모바일 및 내장형 시스템

※ 이 논문은 2016년도 정부의 재원으로 한국연구재단의 지원을 받아 수행된 연구사업(2016R1D1A1B01006716)이며, 2017년도 가톨릭대학교 교비연구비의 지원으로 이루어졌음