JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지(J. Korea Inst. Inf. Commun. Eng.) Vol. 19, No. 9 : 2213~2221 Sep. 2015

온디맨드 거버너 정책에 따른 작업 스케줄링 기법의 성능 평가

탁성우*

Performance Evaluation of Job Scheduling Techniques Incorporating the Ondemand Governor Policy

Sungwoo Tak*

Department of Computer Science and Engineering, Pusan National University, Pusan 609-735, Korea

요 약

안드로이드 기반 스마트폰 플랫폼에서 사용하는 온디맨드 거버너 (Ondemand Governor)는 CPU 사용률 (Utilization)에 따라 CPU 동작 주파수를 조절한다. CPU 사용률의 변화량은 작업 스케줄링에 의해 영향을 받으며, CPU 동작 주파수 증감에 따라 스마트 폰의 전력 소비량도 증감한다. 따라서 작업 스케줄링 방식에 따라 변하는 CPU 사용률 및 동작 주파수는 스마트폰의 전력 소비에 영향을 미친다. 이에 온디맨드 거버너 정책을 작업 스케줄링 기법에 적용시켜 CPU 사용률 및 전력 소비, 그리고 작업 마감시한 측면에서 비교 분석하였다.

ABSTRACT

The ondemand governor used in android-based smartphone platforms is a CPU frequency scaling technique. The ondemand governor sets the CPU operating frequency depending on the CPU utilization rate. Job scheduling affects the CPU utilization rate. The power consumption is proportional to the value of operating frequency. Consequently, CPU frequency scaling and CPU utilization rate have an effect on power consumption in a smartphone. In this paper, we evaluated the performance of job scheduling techniques incorporating the ondemand governor in terms of CPU utilization, power consumption, and job deadline miss ratio.

키워드 : 온디맨드 거버너, 안드로이드, 스마트폰, 전력 소비, 작업 스케줄링

Key word : Ondemand governor, Android, Smartphone, Power consumption, Job scheduling

Received 19 May 2015, **Revised** 26 June 2015, **Accepted** 10 July 2015

*** Corresponding Author** Sungwoo Tak(E-mail:swtak@pusan.ac.kr, Tel:+82-51-510-2387)
Department of Computer Science and Engineering, Pusan National University, Pusan 609-735, Korea

Open Access http://dx.doi.org/10.6109/jkiice.2015.19.9.2213 print ISSN: 2234-4772 online ISSN: 2288-4165

ⓒThis is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License(http://creativecommons.org/li-censes/by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.
Copyright Ⓒ The Korea Institute of Information and Communication Engineering.

Ⅰ. 서 론

본 논문에서는 제한된 배터리 자원을 사용하는 안드로이드 기반 스마트폰 플랫폼에서 온디맨드 거버너를 적용시킨 작업 스케줄링의 성능을 비교 분석하였다. 온디맨드 거버너는 리눅스 기반 다양한 시스템 플랫폼에서 사용되고 있다[1]. 온디맨드 거버너는 주기적으로 관찰한 CPU 사용률 (Utilization)을 사용하여 CPU 동작 주파수를 조절한다. CMOS 기반 CPU 구동 전압은 CPU 동작 주파수 값에 비례하며, CPU 소비 전류량은 CPU 구동 전압에 비례한다[2,3]. 온디맨드 거버너는 CPU 사용률이 임계점 이하가 될 때, CPU 동작 주파수를 20%씩 선형으로 감소시킨다. 그리고 동작 주파수의 감소는 전류 소비량을 감소시켜 스마트폰의 전력 소비를 감소시킨다. 온디맨드 거버너 외에 보존형 (Conservative), 파워세이브 (Powersave), 그리고 퍼포먼스 (Performance) 거버너가 기본적으로 제공하고 있다[4,5].

참고 문헌 [6]에서는, 온디맨드 거버너 정책을 사용하는 스마트폰이 CPU 동작 주파수에 비례하여 전력을 소비하였다. 참고 문헌 [7]과 [8]에서는 CPU 사용률 증가에 따라 CPU 소비 전류량도 선형으로 증가함을 보여주었다. 그러나 참고 문헌 [6]부터 참고 문헌 [8]까지 수행한 온디맨드 거버너 정책의 성능 평가는 단순히 CPU 사용률에 따른 전력 소비량의 분석이었다.

CPU 사용률의 변화량은 작업 스케줄링에 의해 영향을 받는다. CPU 스케줄링 방식은 작업 보존형 (Work Conserving) 및 비작업 보존형 (Non-Work Conserving) 스케줄링, 실시간 (Real-time) 및 비실시간 (None-real-time) 스케줄링, 그리고 선점형 (Preemptive)및 비선점형 (Non-Preemptive)으로 구분된다[9]. 작업 보존형 방식은 처리해야할 작업이 있는 경우, 유휴 시간 없이 해당 작업을 바로 처리한다. 비작업 보존형 방식은 처리해야 할 작업을 바로 처리하지 않고, 일정 시간 동안 유휴 시간을 가진 후 작업을 처리한다. 비실시간 스케줄링과 달리, 실시간 스케줄링 방식은 작업의 실시간 속성을 보장한다. 비선점형 스케줄링과 달리, 선점형 스케줄링 방식에서는 최상위 우선순위를 가진 작업이 CPU 사용권한을 항상 먼저 선점한다.

그림 1은 온디맨드 거버너 성능 평가 과정을 보여준다. 성능 평가 과정의 주요 4개 구성 인자는 (1) 작업 속성, (2) 작업 스케줄링, (3) 스마트폰 CPU 속성, 그리고 (4) 온디맨드 거버너 정책이다. 작업은 실행 요청 시각, 실행 주기, 최악 실행 시간, 그리고 마감 시한 속성을 가진다.

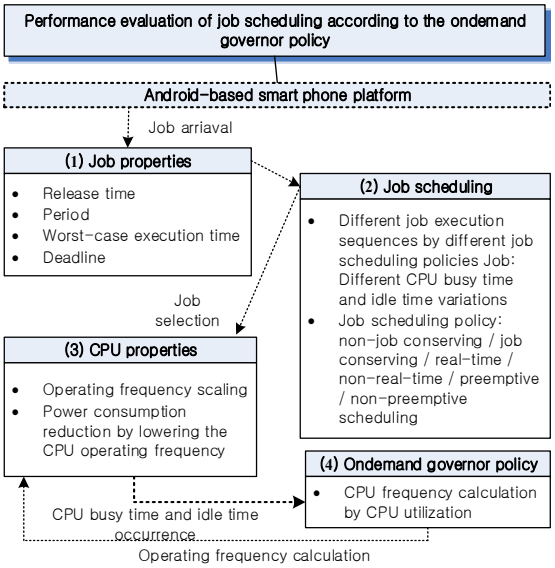

그림 1. 작업 스케줄링의 성능 평가 과정
Fig. 1 Operation procedures for job scheduling performance evaluation

작업 속성 (그림 1-(1))은 성능 평가에 사용되는 작업들의 속성 값이다. 작업은 스케줄링(그림 1-(2))을 통해 실행 순서가 결정되며, CPU (그림 1-(3))에서 실행된다. 그리고 CPU 사용률 변화량에 따라 온디맨드 거버너 정책 (그림 1-(4))이 적용되어 CPU 동작 주파수를 조절한다. 본 논문의 구성은 다음과 같다. II장에서는 온디맨드 거버너 정책과 작업 스케줄링간의 상관관계를 분석하였다. III장에서는 온디맨드 거버너 정책을 적용한 작업 스케줄링의 시뮬레이션 및 성능 평가를 수행하였다. 마지막으로 IV장에서는 결론을 기술하였다.

Ⅱ. 온디맨드 거버너 정책과 작업 스케줄링 기법

온디맨드 거버너 정책의 동작 과정은 표 1과 같다. 본 논문에서는 1GHz Qualcomm QSD8250 Snapdragon ARM 프로세서가 장착된 구글 넥서스 원 (Nexus One)

스마트폰을 사용하였다. 그리고 슈퍼 유저 권한으로 시스템 파일에 접근하기 위하여 루팅을 하였다. 온디맨드 거버너는 샘플링 주기마다 CPU 사용률을 계산하여 CPU 동작 주파수를 증가 혹은 감소시킨다(표 1의 ondemand() 함수 동작 과정). 표 1에서 기술한 변수들은 안드로이드 커널 디렉터리 /sys/devices/ system/cpu/ cpuX/cpufreq/ondemand/에 시스템 파일 형태로 존재한다.

표 1. 온디맨드 거버너 정책의 동작 과정
Table. 1 Operation procedures of the ondemand governor policy

```
1. Initialization
 1.1 cpu_info_transition_latency = 40ms;
 1.2 cpuinfo_min_freq = 100MHz;
 1.3 cpuinfo_max_freq = 1GHz;
 1.4 up_threshold = 80; down_threshold = 20;
 1.5 sampling_rate_min = 10ms;
2. set_sampling_rate() {
 2.1   sampling_rate = cpu_info_transition_latency;
  }
3. ondemand() {
 3.1   CPUfreq = cpuinfo_max_freq;
     while(1) {
 3.2     sampling_rate = max(sampling_rate, sampling_rate_min) ;
 3.3       every sampling_rate:
 3.4         calculate CPU utilization since the last check
 3.5         if (CPU utilization > up_threshold) CPUfreq =
               cpuinfo_max_freq;
 3.6         else if (CPU utilization < down_threshold)
               CPUfreq = max(cpuinfo_min_freq, CPUfreq * 0.2);
     }
  }
```

샘플링 주기는 시스템 파일 cpu_info_transition_latency에 저장된 40ms로 설정하였다 (표 1의 동작 과정 1.1). cpu_info_transition_latency는 CPU가 동작 주파수를 변경하는데 걸리는 지연 시간이다 [10]. 샘플링 주기의 최소 한계값 (sampling_rate_min)은 10ms로 설정하였다(표 1의 동작 과정 1.5). 샘플링 주기 sampling_rate의 최소 한계 값은 sampling_rate_min으로 설정된다 (표 1의 동작 과정 3.2). 온디맨드 거버너의 동작 과정은 다음과 같다. 온디맨드 거버너는 초기 CPU 동작 주파수를 최고 주파수 (cpuinfo_max_freq)로 설정한다 (표 1의 동작 과정 3.1). 샘플링 주기마다 CPU 사용률이 상위 임계치 (up_threshold) 혹은 하위 임계치 (down_threshold) 범위에 있는지 확인한다 (표 1의 동작 과정 3.5와 3.6). 상위 임계값은 CPU 사용률 80%로 설정하였다 (표 1의 동작 과정 1.4). 샘플링 주기 동안 CPU 사용률이 80% 이상이면, CPU 동작 주파수는 최대 주파수 값으로 설정된다 (표 1의 동작 과정 3.5). 하위 임계값은 CPU 사용률 20%로 설정하였다 (표 1의 동작 과정 1.4). 샘플링 주기 동안 CPU 사용률이 20% 이하이면, CPU 동작 주파수를 20%씩 감소시킨다 (표 1의 동작 과정 3.6). 최소 CPU 동작 주파수 (cpuinfo_min_freq) 값은 100MHz로 설정하였다 (표 1의 동작 과정 1.2).

지금까지 살펴본 온디맨드 거버너의 핵심 동작 방식은 CPU 사용률에 따라 동작 주파수를 조절하는 것이다. CPU 사용률의 변화량은 작업 스케줄링 방식에 의해 영향을 받는다. 성능 분석에 적용한 기본 스케줄링 방식은 FCFS (First Come First Service)기반 비실시간 스케줄링과 EDF(Earliest Deadline First)기반 실시간 스케줄링 기법이며, 이를 다음과 같이 변형시켰다.

첫째, EDF는 기본적으로 선점형 (Preemptive) 스케줄링 방식이며, P_EDF (Preemptive EDF)로 표현한다. P_EDF에 대한 변형으로 비선점형 (Non-Preemptive) EDF 스케줄링 방식을 구현하였으며, NP_EDF (Non Preemptive EDF)로 표현한다. NP_EDF 스케줄링은 마감시한이 가장 이른 작업의 실행이 완료되면, 대기 중인 작업들 중에서 마감시한이 가장 이른 작업을 실행할 수 있다. 한편, FCFS는 실행 요청된 순서대로 작업을 처리하는 비선점형 스케줄링 방식이며, NP_FCFS (Non Preemptive FCFS)로 표현한다.

둘째, 작업 실행 시간이 짧은 작업을 먼저 실행하는 JFS (Job First with Shortest Execution Time) 정책을 적용시켜 변형된 FCFS 및 EDF 스케줄링을 구현하였다. JFS 정책을 통해 작업들의 평균 응답 시간을 빠르게 하고 실행 대기 중인 작업들의 마감시한 초과율도 부가적으로 감소시킬 수 있다. 변형된 스케줄링 방식은 NP_FCFS_JFS, P_EDF_JFS, NP_ EDF_JFS 스케줄링이다.

셋째, 비작업 보존형 스케줄링인 작업 그룹기반 스케줄링 (Job Group based Scheduling) 방식을 제안하였다. 첫째와 둘째에서 기술한 스케줄링 방식들은 작업 보존형 스케줄링 방식이다. 구현한 작업 그룹 기반 스케줄링 방식은 JG_NP_FCFS, JG_P_EDF, JG_NP_EDF,

JG_NP_ FCFS_JFS, JG_P_EDF_JFS, 그리고 JG_NP_EDF_JFS이다. 작업 그룹 기반 스케줄링에서는 작업 그룹 상수 *JGC* (Job Group Constant)값을 사용하여 작업 그룹을 형성한다. 그룹을 형성할 때, CPU 실행 유휴 시간을 그룹 내에 존재하도록 한다. 그리고 온디맨드 거버너 정책이 실행 유휴 시간 동안 동작 주파수를 감소시켜 스마트폰의 전력 소비를 감소시킨다. 표 2는 구현된 6개의 작업 그룹 기반 스케줄링 방식에서 JG_NP_EDF 스케줄링의 동작 과정을 보여준다.

표 2. JG_NP_EDF 스케줄링 방식의 동작 과정
Table. 2 Operation procedures of the JG_NP_EDF scheduling scheme

1. Insert_Job():
/* Insert a job Job_i at queue *Job_Ready_Q* according to a sequence of ordered deadlines */
1.1 $Job\_Ready\_Q \leftarrow Job_i$, where $Job(AD)_1 \le Job(AD)_2 \le Job(AD)_i \le ..... \le Job(AD)_n$ in such a way that the resulting sequence of size $n+1$ is also ordered
2. Select_Job():
2.1 if ($Job\_Ready\_Q = \varnothing$) {
Select Job_k, where $Job_k \in Job\_Ready\_Q$ and $Job(AD)_k - MinJob(AD) \le JGC$};
}
2.2 $Job\_Ready\_Q \leftarrow Job\_Ready\_Q - Job_k$;
2.3 Run Job_k;

JG_NP_EDF 스케줄링에서는 작업 그룹 상수 *JGC* 값을 사용하여 작업 그룹을 형성한다. 작업 Job_i의 절대적 마감시한 $Job(AD)_i$는 작업의 실행 요청 시간 $Job(R)_i$와 상대적 마감 시한 $Job(RD)_i$의 합이다. i는 작업 식별자이다. Insert_Job() 함수에서는 절대적 마감 시한이 가장 이른 작업 순서대로 작업 준비 큐 *Job_Ready_Q*에 저장한다 (표 2의 동작 과정 1.1). *Job_Ready_Q*에 저장되어 있는 작업을 선출하는 Select_Job() 함수의 동작 과정은 다음과 같다. 작업 그룹 상수 *JGC* 값과 작업 준비 큐에서 마감시한이 가장 이른 작업의 시한 $Job(AD)_1$ 값을 *MinJob*(*AD*)로 설정한 후, 작업 그룹을 형성한다 (표 2의 동작 과정 2.1). 작업 Job_k의 $Job(AD)_k$와 $Job(AD)_1$ 간의 차이가 작업 그룹 상수 *JGC* 값보다 작은 경우, $Job(AD)_1$ 이 속한 작업 그룹에 포함된다. 해당 작업 그룹의 마감 시한은 $Job(AD)_1$과 *JGC* 값의 합으로 구성된다. 예를 들어, *JGC* 값이 20ms이고 $Job(AD)_1$이 5ms인 경우, 작업 그룹의 마감시한은 25ms가 된다. 마감 시한이 25ms 내에 있는 작업들은 이 작업 그룹에 포함된다. 나머지 작업 그룹 스케줄링 방식은 스케줄링 방식에 따라 표 2의 1.1 과정만 다르며, 나머지 과정은 서로 유사하다.

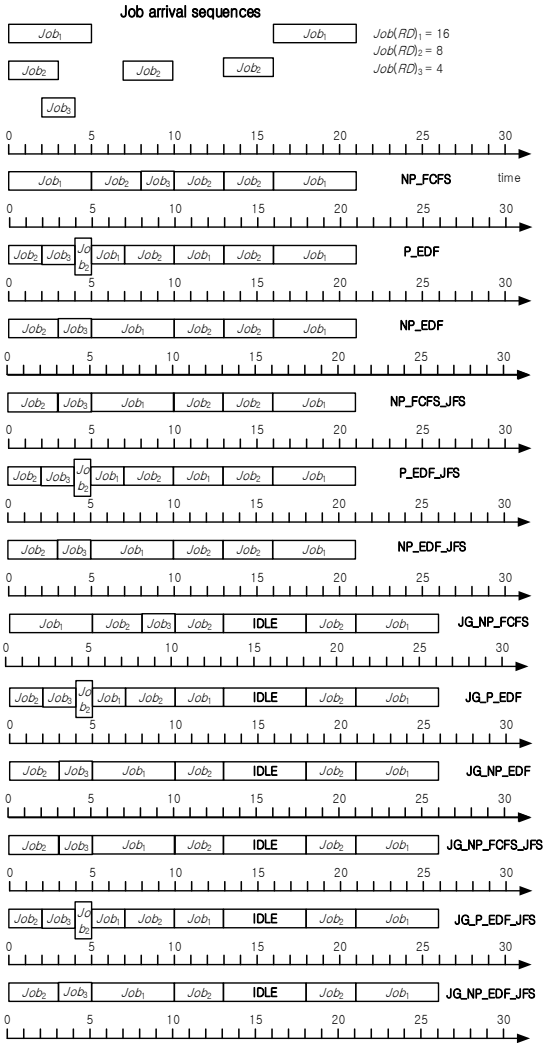

그림 2. 스케줄링 방식별 작업 실행 동작 과정
Fig. 2 Job execution traces according to scheduling schemes

그림 2는 표 3에서 기술한 3개의 작업이 작업 스케줄링들에 의해 처리되는 동작 과정을 보여준다. 표 3에서 $Job(P)_i$는 작업 Job_i의 주기를 나타내며,

$Job(E)_i$는 작업 Job_i의 실행 시간을 나타낸다. $Job(R)_i$와 $Job(RD)_i$는 작업 요청 시간과 상대적 마감 시한을 나타낸다. 작업 FCFS 스케줄링인 경우, 동시에 도착한 작업들이 있을 때 작은 작업 식별자 i를 가진 작업이 먼저 실행된다. 예를 들어, NP_FCFS에서는 시각 0일 때, 동시에 요청된 작업 Job_1과 Job_2 중에서 Job_1을 먼저 실행시킨다. 실행 시간이 짧은 작업을 우선적으로 먼저 실행시키는NP_FCFS_JFS인 경`우, 시각 0에서 요청된 작업 Job_1과 Job_2 중에서 실행 시간이 짧은 Job_2가 먼저 실행된다. 시각 3에서 Job_2가 실행 완료된 후, Job_1과 Job_3 중에서 실행 시간이 짧은 Job_3를 실행한다.

표 3. 태스크 집합
Table. 3 A set of tasks

Job_i	$Job(R)_i$	$Job(P)_i$	$Job(RD)_i$	$Job(E)_i$	
Job_1	0	16	16	5	Periodic
Job_2	0	None	8	3	Aperiodic
Job_3	2	None	4	2	Aperiodic

비선점형 작업 그룹 기반 JG_NP_EDF 스케줄링의 동작 과정을 살펴보면 다음과 같다. 작업 그룹 상수 JGC 값은 10으로 설정하였다. 시각 0일 때, 작업 준비 큐에서 마감시한이 가장 이른 작업의 마감시한 $MinJob(AD)_1$ 값은 Job_2의 마감시한 8 (= $Job(AD)_2$)로 설정되며, 이 값을 기반으로 하여 작업 그룹이 형성된다. 그리고 마감시한이 가장 이른 Job_2가 먼저 실행된다. 비선점형 스케줄링 정책에 의해 Job_3는 Job_2가 실행 완료될 때까지 기다린다. 시각 3에서 Job_2가 실행 완료된 후, 실행 대기 중인 Job_1과 Job_3 중에서 실행 시간이 짧은 Job_3이 먼저 실행 권한을 획득한다. 표 2의 동작 과정 2.1을 사용하여 Job_3이 Job_2가 속한 작업 그룹에 포함되는지를 확인하는 과정은 다음과 같다. 표 2의 동작 과정 2.1에서 $Job(AD)_k$는 $Job(AD)_3$ 값인 6 (= $Job(R)_3$ + $Job(RD)_3$ = 2 + 4) 으로 설정된다. $MinJob(AD)$에는 $Job(AD)_2$ 값인 8 (= $Job(R)_2$ + $Job(RD)_2$ = 0 + 8) 이 삽입된다. 따라서 $Job(AD)_3$ (= 6) – $MinJob(AD)$ (= 8) ≤JGC (= 10)이 만족되어 Job_3은 Job_2가 속한 작업 그룹에 포함되어 실행된다. 시각 10에서 실행된 Job_2의 실행이 완료된 후, 시각 13에서 다시 실행 요청된 Job_2가 작업 그룹에 포함되기 위해서는 표 2의 동작 과정 2.1을 만족해야한다. $Job(AD)_2$ (= $Job(R)_2$ + $Job(RD)_2$ = 13 + 8 = 21) – $MinJob(AD)$ (= 8) ≤JGC (= 10)이 만족되지 않기에 시각 13에서 Job_2는 다음 작업 그룹에 포함된다. 그리고 시각 13부터 시각 18까지는 실행 유휴 시간으로 설정된다. 이외에 다른 작업 그룹 기반 스케줄링 도 이와 유사하게 실행된다. 지금까지 살펴본 작업 그룹 스케줄링 기법의 이론적인 성능 분석은 다음과 같다.

$$Idle_{avrage} = \frac{T - \sum_{k=1}^{K}\sum_{i=1}^{m} Job(E)_{k,i}}{K}, \quad (1)$$
$$where\ MinJob(AD)_{k,1} - Job(AD)_{k,i} \le JGC,$$
$$K = \min\{k \mid T \le MinJob(AD)_{k,1} + JGC\}$$

수식 (1)은 시각 T까지 생성된 K 개의 작업 그룹에서 발생된 실행 유휴 시간의 평균값을 나타낸다. k는 그룹 식별자를 나타낸다. $Job(E)_{k,i}$는 k 번째 그룹에 속한 작업 Job_i의 실행시간을 나타낸다. $MinJob(AD)_{k,1}$ 은 k 번째 그룹에 속한 가장 이른 작업의 마감 시한 값을 나타낸다.

$$\begin{aligned} Idle &= (MinJob(AD)_{k,1} + JGC) - (Job(R)_{k,m} + Job(E)_{k,m}) \\ &= (Job(R)_{k,1} + Job(RD)_{k,1} + JGC) - (Job(R)_{k,m} + Job(E)_{k,m}) \\ &= Job(RD)_{k,1} + JGC - (Job(R)_{k,m} - Job(R)_{k,1}) - Job(E)_{k,m}, \\ &where\ Job(AD)_{k,m} < (MinJob(AD)_{k,1} + JGC) < Job(AD)_{k,m+1} \end{aligned} \quad (2)$$

수식 (2)는 k 번째 그룹의 실행 유휴 시간을 나타낸다. k 번째 그룹은 해당 그룹에서 마감시한이 가장 이른 작업의 $MinJob(AD)_{k,1}$ 값과 작업 그룹 상수 JGC 값을 기반으로 하여 형성된다. 작업 Job_{m+1}의 절대 마감시한 $Job(AD)_{m+1}$이 $MinJob(AD)_{k,1}$과 JGC 값의 합보다 큰 경우, Job_{m+1}의 실행을 지연시키고 다음 작업 그룹에 포함시킨다. 이러한 과정을 통해 작업 그룹 k는 $Idle$만큼의 실행 유휴 시간이 발생한다.

수식 (1)에서 시각 T 동안 작업 그룹 상수 JGC값이 증가하는 경우, 형성되는 작업 그룹의 개수 K는 감소한다. 그리고 K 값의 감소로 인해 평균 실행 유휴시간 $Idle_{average}$ 값은 증가한다. 여기에서 작업 그룹 상수 JGC와 작업 그룹의 개수 K는 반비례 관계임을 알 수 있다. 그러나 JGC 값을 계속 증가시키더라도 실행 유휴시간 $Idle_{average}$ 값이 무한히 계속 증가하지 않는다. 이러한 현상이 발생되는 이유는 다음과 같다. JGC 값을 계속 증가시키면 작업 그룹 k 내에 포함되는 작업의 수가 증가

한다. 즉, 작업 그룹 k에 추가된 마지막 작업 Job_m의 실행 요청 시간 $Job(R)_{k,m}$과 $Job(E)_{k,m}$의 값이 증가하여 수식 (2)의 실행 유휴 시간 *Idle* 값은 감소한다.

실행 유휴 시간은 온디맨드 거버너 정책을 적용한 스마트 폰의 전력 소비량 감소에 영향을 미친다. 그리고 실행 유휴 시간 및 작업 그룹의 형성은 수식 (1)과 수식 (2)에서 기술한 *JGC* 값에 영향을 받는다. 이에 주어진 작업 집합에 대하여 최대 실행 유휴 시간을 생성하면서 작업 마감 시한 초과율을 최소화하는 *JGC* 값을 적용하면, 스마트폰의 전력 소비량 감소 및 실시간 서비스의 품질을 보장할 수 있다.

Ⅲ. 성능 분석

먼저, 성능 분석에 사용되는 작업 집합을 무작위로 생성하는 랜덤 작업 생성 알고리즘을 기술하였다. 작업 생성 알고리즘은 표 4와 같다. CPU 사용률은 개별 작업 마감시한 분의 작업 실행 시간들의 합으로 계산하였다. 총 N개의 작업을 고려하는 경우, $\sum_{i=1}^{N}(Job(RD)_i/Job(P)_i)$로 표현된다. 작업 생성 알고리즘의 입력은 작업 사용률 *TU*이며 출력은 입력된 작업 사용률을 생성하는 작업 집합이다. 작업 생성 알고리즘에서 사용하는 변수는 다음과 같다. i는 작업 식별자이다. 1부터 최대 N개까지의 작업을 생성할 수 있다.

표 4에서 기술한 작업 생성 알고리즘의 실행 순서는 다음과 같다. 먼저, 작업의 실행시간을 랜덤하게 생성한다 (표 4의 동작 과정 1.1). *MinE*는 생성되는 작업 실행시간의 최소값이다. *MaxE*는 생성되는 작업 실행 시간의 최대값이다. *RandGen*(*MinE*, *MaxE*) 함수는 *MinE*부터 *MaxE* 내에서 랜덤 값을 생성한다. *RandGen*() 함수는 표 4의 동작 과정 2에서 기술하였다.

표 4의 동작 과정 1.2 부터 1.6까지는 작업 Job_i에 대한 사용률 $Job(U)_i$, 주기 $Job(P)_i$, 상대적 마감시한 $Job(RD)_i$를 생성하는 과정이다. 표 4의 동작 과정 1.2에서는 작업 Job_i를 제외하고 i보다 큰 식별자 값을 가지는 작업들의 사용률 합을 랜덤하게 생성한다. 생성된 사용률은 *NextTU*에 저장된다. *TU*는 작업 Job_i를 포함하고 i보다 큰 식별자 값을 가지는 작업들의 사용률 합이다. 표 4의 동작 과정 1.3에서는 작업 Job_i의 사용률 $Job(U)_i$를 계산한다. *TU*와 *NextTU*간의 차이 값이 $Job(U)_i$가 된다. *NextTU*가 ε에 근접하면, 더 이상 할당 가능한 작업 사용률이 없기 때문에 작업 생성 알고리즘은 실행을 중지한다 (표 4의 동작 과정 1.4). ε은 0에 가까운 양의 실수 값이며, 본 논문에서는 0.0001로 설정하였다. 표 4의 동작 과정 1.5에서는 동작 과정 1.1에서 생성된 작업 실행 시간 $Job(E)_i$와 동작 과정 1.3에서 생성한 작업 사용률 $Job(U)_i$를 곱하여 작업의 상대적 마감시한 $Job(RD)_i$을 생성한다.

표 4. 작업 생성 알고리즘
Table. 4 Job generation algorithm

```
i ← 1;
N, MinE, MaxE ← Constants given by a user;
TU ← Input value given by a user;
1. while (True) {
   1.1 Job(E)_i ← RandGen(MinE, MaxE);
   1.2 NextTU ← TU × RandGen(0, 1)^{1/(N-1-i)};
   1.3 Job(U)_i ← TU - NextTU;
   1.4 if (NextTU < ε) return sucess;
       else if (NextTU < 0) return failure;
   1.5 Job(RD)_i ← Job(E)_i × Job(U)_i;
   1.6 Job(P)_i ← Job(RD)_i;
   1.7 TU ← NextTU;
   1.8 i ← i + 1;
   }
2. RandGen(MinE, MaxE):
   2.1 rand ← a random number between 0 and 1;
   2.2 return (MinE + rand * MaxE);
```

성능 평가에 사용되는 구글 넥서스 원 (Nexus One) 스마트폰은 1GHz Qualcomm QSD8250 Snapdragon ARM 프로세서를 사용한다. 이 프로세서의 동작 전류량은 CPU 동작 주파수에 따라 변하며, 최대 1GHz에서 240mA/sec, 245MHz에서 70mA/sec, 그리고 19.2MHz에서 25.62mA/sec만큼의 동작 전류량을 소비한다. 동작 주파수는 19.2MHz 단위로 감소하거나 증가하며, 동작 주파수에 따른 전류 소비량은 선형으로 변한다[6,7]. 성능 평가 요소로는 전류 소비량과 작업의 마감시한 초과율, 그리고 실행 유휴 시간을 사용하였다. 표 4에서 제시한 알고리즘을 사용하여 개별 사용률 0.1부터 0.9까지 0.1 단위로 각각 10개의 작업 집합을 생성한 후, 시뮬레이션을 실행하였다. 시뮬레이션 시간은 1시간으로 설정하였다.

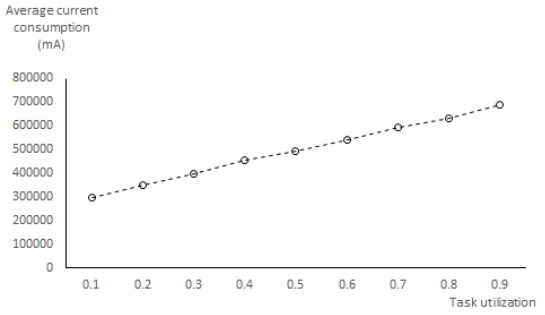

그림 3. 평균 전류 소비량
Fig. 3 Average current consumption

그림 3은 작업 보존형 스케줄링 방식인 NP_FCFS, NP_FCFS_JFS, P_EDF, P_EDF_JFS, 그리고 NP_EDF_JFS의 평균 전류 소비량을 보여준다. 작업 보존형 스케줄링은 처리해야 할 작업이 있으면 작업을 계속 처리해야 하므로, 작업 보존형 스케줄링의 종류와 상관없이 개별 작업 집합의 평균 전류 소비량은 동일하였다. 그리고 사용률 증가에 따라 평균 전류 소비량도 선형으로 증가하였다. 온디맨드 거버너 정책은 주기적으로 CPU 사용률을 샘플링하여 동작 주파수를 선형으로 조절하기 때문에 동작 주파수에 따른 전류 소비량도 선형으로 변한다.

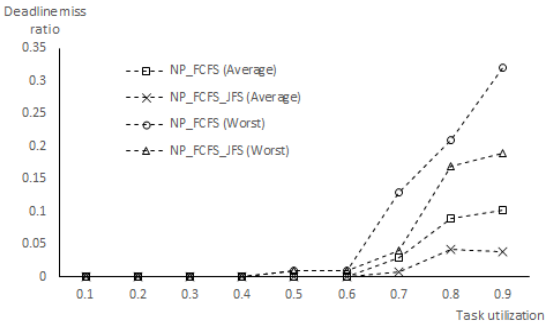

그림 4. NP_FCFS 방식의 마감시한 초과율
Fig. 4 Deadline miss ratio by NP_FCFS schemes

그림 4에서는 실행 시간이 짧은 작업을 우선적으로 처리하는 JFS 방식의 마감시한 초과율 성능을 비교하였다. 마감시한을 고려하는 EDF 방식은 JFS 방식에 영향을 받지 않는다. 따라서 그림 4에서는 작업 보존형 스케줄링 방식인 비선점형 NP_FCFS 스케줄링 방식의 평균 및 최악 마감시한 초과율만을 보여준다. 그림 7에서 보는 바와 같이, 실행 시간이 짧은 작업을 우선적으로 실행하는 NP_FCFS_JFS 스케줄링의 마감시한 초과율이 NP_FCFS 스케줄링보다 낮음을 확인하였다.

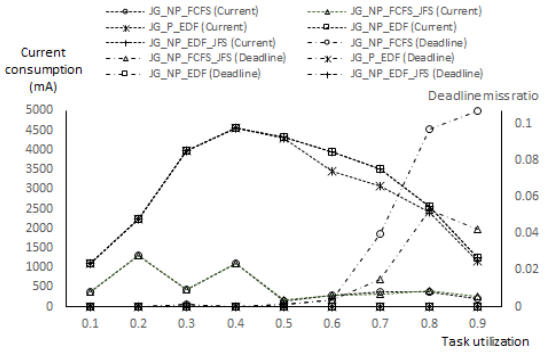

그림 5. 평균 소비 전류 차이와 마감시한 초과율
Fig. 5 Average deadline miss ratio

그림 5는 비작업 보존형 스케줄링 방식이 작업 보존형 스케줄링 방식에 비해 감소된 평균 전류 소비량과 증가된 마감시한 초과율을 보여준다. 전체적으로 비작업 보존형 스케줄링 방식이 작업 보존형 스케줄링 방식보다 낮은 전류량을 소비한다. 그리고 마감시한 초과율 측면에서도 비보존형 작업 스케줄링 방식은 FCFS 방식을 제외하고, 작업 보존형 스케줄링 방식과 매우 유사한 성능을 보여 주었다.

그림 6과 그림 7은 비선점형과 비작업 보존형을 동시에 고려한 스케줄링 기법에서 성능이 가장 우수한 JG_NP_EDF_ JFS 스케줄링 방식의 세부 성능을 보여준다. 성능 분석에 사용한 사용률은 0.4와 0.7이며, 작업 그룹 상수 값에 따라 변하는 평균 소비 전류 및 마감 시한이다. 작업 그룹 상수 25까지는 작업 그룹 상수 값이 증가하면, 실행 유휴 시간이 증가하여 전류 소비량이 감소하는 경향을 보여준다. 그러나 실행 유휴 시간의 증가로 인해 작업의 마감시한 초과율도 증가하는 경향을 보여준다. II장의 식 (2)에서 기술한 바와 같이, 작업 그룹 상수가 증가하는 경우, 작업 그룹 내에 포함되는 작업의 수가 증가하여 실행 유휴 시간이 감소하여 전류 소비량은 증가한다. 그리고 작업 보존형 스케줄링 방식의 소비 전류량 및 마감 시한 초과율에 수렴한다. 또한 작업 그룹 상수가 증가하는 경우, 작업 그룹 내에 포함되는 작업의 수가 증가하기 때문에 해당 작업들의 마감시한 초과율은 감소한다.

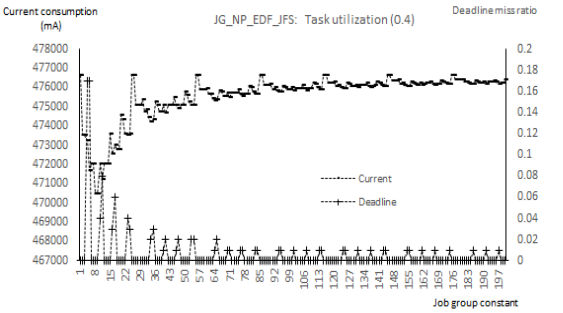

그림 6. 평균 소비 전류와 마감시한 초과율 (사용률: 0.4)
Fig. 6 Average current consumption and deadline miss ratio (Utilization: 0.4)

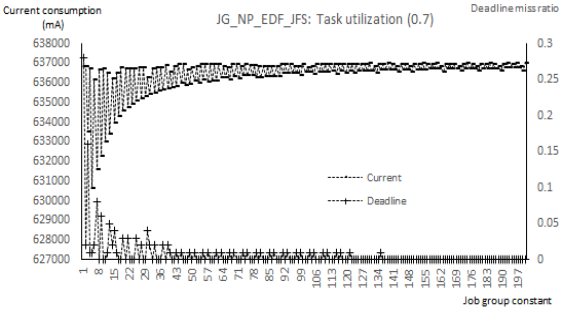

그림 7. 평균 소비 전류와 마감시한 초과율 (사용률: 0.7)
Fig. 7 Average current consumption and deadline miss ratio (Utilization: 0.7)

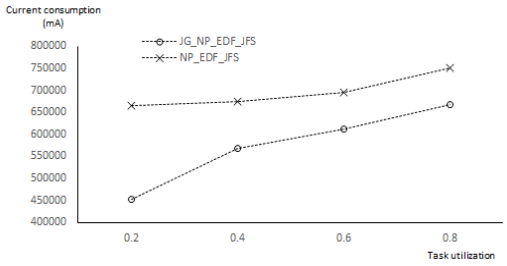

그림 8. 스마트 폰의 전력 소비량
Fig. 8 Current consumption of the smart phone

안드로이드 기반 앱 환경에서는 선점형 스케줄링을 적용하지 못한다. 이에 비선점형에서 성능이 가장 우수한 JG_NP_EDF_JFS 및 NP _EDF_JFS 스케줄링을 스마트 폰에 적용하였다. 그림 8은 이러한 환경에서 소비된 전류량을 보여준다. 스마트폰에 장착된 배터리 용량은 1400mAh이다. 사용한 실험 환경은 리눅스 환경에 TCP 서버를 운영하며, 스마트 폰과 와이파이를 통해 400K바이트 메시지를 30분 동안 계속해서 송수신한다. 각각의 메시지는 마감시한을 가지며 각각의 메시지는 스케줄링에서 처리해야 할 작업으로 대응된다. 작업 그룹 상수 값은 마감시한 초과율이 가장 낮고 전류 소비가 가장 작은 작업 그룹 상수 값을 JG_NP_EDF_JFS에 사용하였다. 비작업 보존형 스케줄링 방식 JG_NP_EDF_JFS에서는 실행 유휴 시간동안 와이파이 네트워크 인터페이스가 수면 모드로 전환되어 전력 소비를 감소시킬 수 있으며, CPU 동작 주파수도 온디맨드 거버너 정책에 의해 감소되기 때문에 전체 전력 소비를 감소시킬 수 있었다.

작업 스케줄링 기법의 성능 분석 결과를 정리하면 다음과 같다. 온디맨드 거버너의 핵심 동작 방식은 샘플링 주기 동안 발생한 CPU 사용 시간을 측정한다. 측정된 CPU 사용 시간을 샘플링 주기로 나누어 계산된 CPU 사용률 (Utilization)에 따라 동작 주파수를 조절하는 것이다. CPU 사용 시간과 CPU 유휴 시간을 포함한 CPU 사용률의 변화량은 작업 스케줄링에 의해 영향을 받는다. 즉 온디맨드 방식과 작업 스케줄링간의 밀접한 연관이 있음을 확인하였다. 그리고 CPU 사용률과 동작 주파수 변화는 스마트폰의 전력 소비량 변화에 영향을 미치는 것을 확인하였다. 제안한 비작업 보존형 스케줄링인 작업 그룹기반 스케줄링 (Job Group based Scheduling) 방식이 온디맨드 거버너 정책을 사용하는 경우 전력 소비와 작업 마감시한 보장 측면에서 우수한 성능을 보여주었다 .

Ⅳ. 결 론

온디맨드 거버너 정책을 사용하는 안드로이드 기반 스마트 플랫폼에서는 실행 유휴 시간이 스마트 폰의 전력 소비를 감소시킬 수 있는 중요한 성능 인자이다. 이에 본 논문에서는 작업들 간에 실행 유휴 시간을 생성하는 비작업 보존형 스케줄링 기법을 제안하여 작업 보존형 스케줄링 기법과 성능을 비교 분석하였다. 성능 분석을 수행한 결과, 비작업 보존형 스케줄링 방식이 전력 소비 및 마감 시한 측면에서 우수한 성능을 제공하였다.

ACKNOWLEDGMENTS

This work was supported by a 2-Year Research Grant of Pusan National University

REFERENCES

[1] V. Pallipadi and A. Starikovskiy, "The Ondemand Governor," in *Proceedings of Linux Symposium,* Ottawa, Canada, pp. 223-228, 2006.

[2] P. Pillai and K.G. Shin, "Real-Time dynamic voltage scaling for low-power embedded operating systems," in *Proceedings of ACM symposium on Operating Systems Principles*, pp. 89-102, 2001.

[3] S. Tak, "Performance evaluation of real-time power-aware scheduling techniques incorporating idle time distribution policies," *Journal of the Korea Institute of Information and Communication Engineering*, vol. 18, no. 7, pp. 1704-1712.

[4] M. Kim, Y. Kim, S. Chung, and C. Kim, "Measuring variance between smartphone energy consumption and battery life," *IEEE Computer Magazine*, vol. 47, no. 7, pp.59-65, 2014.

[5] D. Brodowski and N. Golde, Linux CPUFreq Governors [Internet]. Available: www.kernel.org/doc/Documentation/cpu-freq/governors.txt.

[6] M.J. Johnson, and K.A. Hawick, "Optimizing energy management of mobile computing devices," in *Proceedings of International Conference on Computer Design*, Las Vegas, USA, pp. 1-7, 2012.

[7] R. Murmuria, J. Medsger, A. Stavrou, and J.M.Voas, "Mobile application and device power usage measurements," in *Proceedings of International Conference on Software Security and Reliability*, Gaithersburg:USA, pp. 147-156, 2012.

[8] K. Nagata, S. Yamaguchi, and H. Ogawa, "A Power Saving Method with Consideration of Performance in Android Terminals," in *Proceedings of International Conference on Autonomic & Trusted Computing*, Fukuoka:Japan, pp. 578-585, 2012.

[9] F. Cottet, J. Delacroix, C. Kaiser, and Z. Mammeri, Scheduling in Real-Time Systems, Wiley, 2002.

[10] A. Mazouz, A. Laurent, B. Pradelle, and W. Jalby, "Evaluation of CPU frequency transition latency," *Computer Science - Research and Development*, vol. 29, no. 3-4, pp. 187-195, Aug. 2014.

탁성우(Sungwoo Tak)

2003년 2월 미국미주리주립대학교 Computer Science 박사
2004년 ~ 현재 부산대학교 정보컴퓨터공학부 교수 (부산대 컴퓨터및정보통신연구소, 스마트제어센터 겸임 연구원)
※관심분야 : 유무선 네트워크, 위치인식