JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지(J. Korea Inst. Inf. Commun. Eng.) Vol. 20, No. 12 : 2401~2409 Dec. 2016

안드로이드 CPU 거버너의 전력 소비 및 실시간 성능 평가

탁성우*

Evaluating Power Consumption and Real-time Performance of Android CPU Governors

Sungwoo Tak*

School of Electrical and Computer Engineering, Pusan National University, Pusan 46241, Korea

요 약

안드로이드 CPU 거버너는 CPU 주파수를 낮추어 CPU 공급 전압을 감소시키는 DVFS (Dynamic Voltage Frequency Scaling) 기반 전력 관리 기법을 사용한다. 그러나 CPU 주파수의 감소는 태스크의 실행 속도 지연을 유발한다. 이로 인해 태스크의 응답 시간 및 마감 시한 초과율이 증가하여 태스크가 제공하는 서비스의 품질 하락이 발생한다. 이에 본 논문에서는 다양한 안드로이드 CPU 거버너들을 전력 소비와 태스크의 응답성 및 마감 시한 측면에서 분석하였다.

ABSTRACT

Android CPU governors exploit the DVFS (Dynamic Voltage Frequency Scaling) technique. The DVFS is a power management technique where the CPU operating frequency is decreased to allow a corresponding reduction in the CPU supply voltage. The power consumed by a CPU is approximately proportional to the square of the CPU supply voltage. Therefore, lower CPU operating frequency allows the CPU supply voltage to be lowered. This helps to reduce the CPU power consumption. However, lower CPU operating frequency increases a task's execution time. Such an increase in the task's execution time makes the task's response time longer and makes the task's deadline miss occur. This finally leads to degrading the quality of service provided by the task. In this paper, we evaluated the performance of Android CPU governors in terms of the power consumption, tasks's response time and deadline miss ratio.

키워드 : 안드로이드 거버너, 저전력, 실시간, 동적 전압 주파수 조절

Key word : Android governor, Low-power, Real-time, Dynamic voltage frequency scaling

Received 18 July 2016, **Revised** 21 July 2016, **Accepted** 04 August 2016

* **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 46241, Korea

Open Access http://doi.org/10.6109/jkiice.2016.20.12.2401 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

안드로이드 기반 스마트 폰은 전력 관리를 위해 CPU 주파수를 낮추어 CPU 공급 전압을 감소시키는 DVFS (Dynamic Voltage Frequency Scaling) 기반 전력 관리 기법을 사용한다. Conservative 거버너와 Ondemand 거버너는 CPU 주파수를 항상 최대 값으로 설정하는 Performance 거버너에 비해 10%부터 30%까지 전력 소비량을 감소시켰다[1]. 왕복 지연시간이 긴 네트워크 환경에서 Ondemand 거버너를 사용하는 경우, 70% 정도의 전력 소비를 감소시켰다[2]. 상호 작용이 많은 스마트폰 사용 환경에서, Conservative 거버너가 Ondemand 거버너에 비해 약 28%의 전력 소비를 감소시켰다[3]. SSL (Secure Socket Layer)기반 웹 연결에서 Ondemand 거버너가 Performance 거버너에 비해 10% 정도의 전력 소비를 감소시켰다[4]. CPU 사용률이 작은 작업을 처리하는 경우, Ondemand 거버너가 Performance 거버너에 비해 약 30% 정도의 전력 소비를 감소시켰다[5].

CPU 거버너의 성능이 미비함을 기술한 연구는 다음과 같다. 참고 문헌 [6]에서 20명이 250일 동안 사용한 스마트폰의 평균 실행 상태는 50.7% 이었다. 실행 상태에서 CPU의 전력 소비량은 12.7%이었다. 참고 문헌 [4]에서 보여준 Ondemand 거버너의 10% 전력 소비 감소 효과를 참고 문헌 [6]의 CPU 전력 소비량 분석 결과에 적용하면, 감소될 수 있는 CPU 소비전력량은 1.27% (= 0.127 × 0.1 = 0.0127)이다. 이에 참고 문헌 [7]에서는 거버너의 전력 소모 한계를 주장하였다. 이와 반대로 거버너의 성능이 미비하지 않음을 다음과 같이 추정할 수 있다. CMOS 기반 CPU의 전력 소비량은 CPU 공급 전압의 제곱에 비례한다[8]. 삼성 Exynos 5250 프로세스는 실행 유휴 상태에서 CPU 공급 전압은 0.926V이며, 최고 주파수 1.7GHz에서 CPU공급 전압은 1.278V이다[9]. CPU 주파수 1.7GHz 대비 실행 유휴인 경우에 약 47.5% (= 100% * (1.278^2 – 0.926^2) / 1.278^2)의 전력 소비를 감소시킬 수 있다. 이러한 분석 결과는 거버너의 성능이 미비하지 않음을 보여준다.

마지막으로, CPU 주파수의 감소는 태스크의 실행 속도 지연을 유발한다. 이로 인해 태스크의 응답 시간 및 마감 시한 초과율이 증가하여 태스크의 서비스 품질 하락이 발생할 수 있다. 이에 본 논문에서는 전력 소비와 태스크의 응답성 및 마감 시한 측면에서 거버너를 분석하여 거버너의 적합성을 평가하고자 한다.

Ⅱ. 본 론

성능 평가에 사용된 거버너들은 Ondemand, Conservative, LoadBalanced, SmartAssV2, DPM (Dynamic Power Management), 그리고 DPMSampling이다. 이 중에서 LoadBalanced 거버너는 다른 거버너들과 성능 비교를 위하여 본 논문에서 제안한 거버너이다. 거버너에서 사용하는 CPU 주파수 결정 인자들은 결정 시기 (주기 혹은 비주기), 결정 인자 (CPU 부하 및 상태 등), 그리고 결정 방식이다. 결정 방식에서 임계 조절 방식은 CPU 부하 혹은 주파수 사용 시간이 특정 임계 값 이상인 경우에 CPU 주파수를 조절한다. 선형 조절 방식은 CPU 부하 값에 따라 CPU 주파수를 선형으로 조절한다. 특정 조절 방식은 CPU 상태 변화에 따라 특정 CPU 주파수로 설정한다. 모든 거버너의 초기 CPU 주파수와 표 3과 4에서 사용하는 이전 CPU 주파수 (prevCPU freq)는 최고 CPU 주파수로 설정하였다.

Table. 1 Ondemand governor procedures

```
1. procedure ONDEMAND (CPUload):
 1.1  Every governor_timer_rate:
 1.2    Calculate CPU load since the last sampling;
 1.3    if (CPUload > UP_THRESHOLD)
 1.4       CPUfreq = CPUmaxFreq;
 1.5    else if (CPUload < (UP_THRESHOLD –
                DOWN_ DIFFERENTIAL)) {
 1.6            LoadRatio = CPUload / (UP_THRESHOLD –
                DOWN_ DIFFERENTIAL) ;
 1.7            CPUfreq = max(LoadRatio × CPUmaxFreq,
                                CPUminFreq);
       }
```

표 1은 Ondemand 거버너의 동작 과정이다. 결정 시기는 주기 방식이다 (동작 과정 1.1). 결정 인자는 CPU 부하 (CPUload)이다 (동작 과정 1.3과 1.6). CPU 부하는 관찰된 시간 대비 CPU 실행 시간 비율에 100을 곱한 값이다. 결정 방식은 임계 조절 (동작 과정 1.3과 1.4)과 선형 조절 방식 (동작과정 1.7)이다. 동작 과정 1.4에서는 CPU 부하가 상향 임계 (UP_ THRESHOLD)보다 높으면, CPU 주파수를 최고 CPU 주파수 (CPUmaxFreq)

로 설정한다. CPU 부하가 상향 임계와 하향 격차 (DOWN_ DIFFERENTIIAL)간의 차이보다 낮으면, CPU 부하 비율 (LoadRatio)을 최고 CPU 주파수에 곱하여 CPU동작 주파수를 선형으로 감소시킨다. 상향 임계와 하향 격차 값은 각각 80과 10이다.

Table. 2 Conservative governor procedures

```
1. procedure CONSERVATIVE (CPUload):
 1.1 Every governor_timer_rate:
 1.2   Calculate CPU load since the last sampling;
 1.3   if (CPUload > UP_THRESHOLD)
 1.4     CPUfreq = min(CPUfreq + (FREQ_STEP × CPUmaxFreq),
                       CPUmaxFreq);
 1.5   else if (CPUload < (UP_THRESHOLD –
             DOWN_ DIFFERENTIAL))
 1.6           CPUfreq = max(CPUfreq - (FREQ_STEP ×
                       CPUmaxFreq), CPUminFreq);
```

표 2는 Conservative 거버너의 동작 과정이다. 결정 시기는 주기 방식이다 (동작 과정 1.1). 결정 인자는 CPU 부하이다 (동작 과정 1.3과 1.5). 결정 방식은 선형 조절 방식이다 (동작과정 1.4와 1.6). CPU 부하가 상향 임계보다 높으면 (동작 과정 1.3), 최고 CPU 주파수에 FREQ_STEP을 곱한 값을 CPU 주파수에 더한다. 동작 과정 1.6은 동작 과정 1.4와 반대로 CPU 주파수를 선형적으로 감소시킨다. FREQ_STEP의 값은 0.05이다. Conservative 거버너의 상향 임계와 하향 격차 값은 각각 80과 10이다.

Table. 3 LoadBalanced governor procedures

```
1. procedure LoadBalanced (CPUload):
 1.1 Every governor_timer_rate:
 1.2   Calculate CPU load since the last sampling;
       Calculate prevCPUfreqSpendTime since last CPUfreq change;
 1.3   if (CPUload > MAX_CPU_LOAD )
 1.4     CPUfreq = CPUmaxFreq;
 1.5   else  CPUfreq = max((CPUload/100) × CPUmaxFreq,
                       CPUminFreq);
 1.6   if (prevCPUfreq != CPUfreq &&
         prevCPUfreqSpendTime < MIN_SAMPLE_TIME)
 1.7     CPUfreq = prevCPUfreq;
 1.8   prevCPUfreq = CPUfreq;
```

Table. 4 SmartAssV2 governor procedures

```
1. procedure LoadBalanced (CPUload):
1.1  Every governor_timer_rate:
1.2   Calculate CPU utilization since the last sampling;
1.3   Calculate prevCPUfreqSpendTime since last CPUfreq change;
1.4   if (CPUState == WAKEUPfromSLEEP) {
1.5       CPUfreq = SLEEP_WAKEUP_FREQ;
1.6       return;
1.7   } else  if (CPUstate == SLEEP)
1.8     idealFreq = SLEEP_IDEAL_FREQ;
1.9   else if (CPUstate == AWAKE)
1.10    idealFreq = AWAKE_IDEAL_FREQ;
      if (CPUload > MAX_CPU_LOAD && prevCPUfreq <
        CPUmaxFreq &&  (prevCPUfreq < idealFreq ||
1.11    prevCPUfreqSpendTime > UP_TIME))
1.12      rampDir = UP;
      else  if (CPUload < MIN_CPU_LOAD && prevCPUfreq >
        CPUminFreq &&  (prevCPUfreq > idealFreq ||
1.13    prevCPUfreqSpendTime > DOWN_TIME))
1.14      rampDir = DOWN;
1.15  else rampDir = NO_CHANGE;
1.16  if (rampDir == UP) {
1.17    if (prevCPUfreq < idealFreq) CPUfreq = idealFreq;
.       else CPUfreq = min (prevCPUfreq + RAMP_UP_STEP,
1.18                        CPUmaxFreq);
1.19  } else if (rampDir == DOWN) {
1.20    if (prevCPUfreq > idealFreq) CPUfreq = idealFreq;
        else CPUfreq = max (prevCPUfreq -
1.21                        RAMP_DOWN_STEP,   CPUminFreq);
1.22  } else if (rampDir == NO_CHANGE) CPUfreq = prevCPUfreq;
      prevCPUfreq = CPUfreq;
```

표 3은 LoadBalanced 거버너의 동작 과정이다. LoadBalanced 거버너는 CPU 부하에 비례하여 CPU 주파수를 증감한다. 결정 시기는 주기 방식이다 (동작 과정 1.1). 결정 인자는 CPU 부하이다 (동작 과정 1.3). 결정 방식은 임계 조절 (동작 과정 1.3과 1.7)과 선형 조절 (동작 과정 1.6)이다. 동작 과정 1.3에서 CPU 부하가 최고 CPU 부하 (MAX_CPU_LOAD)보다 높으면 최고 CPU 주파수로 설정한다. CPU 부하가 최고 CPU 부하보다 낮으면, 최고 CPU 주파수에 CPU 부하 비율을 곱하여 선형으로 CPU 주파수를 조절한다 (동작 과정 1.6). 최고 CPU 부하 값은 99이다. 동작 과정 1.4와 1.6에서 계산한 CPU 주파수를 사용하기 위해서는, 이전 CPU 주파수 (prevCPUfreq)가 사용한 시간 (prevCPU freqSpend Time)이 최소 샘플 시간 (MIN_SAMPLE_ TIME) 보다 커야 한다 (동작 과정 1.7의 2 번째 조건).

이전 CPU 주파수 사용 시간의 초기 값과 최소 샘플 시간은 20msec이다.

표 4는 SmartAssV2 거버너의 동작 과정이다. 결정 시기는 주기 방식이다 (동작 과정 1.1). 결정 인자는 CPU 부하 (동작 과정 1.10과 1.12)와 CPU 상태 (동작 과정 1.3, 1.6, 그리고 1.8)이다. 결정 방식은 임계 조절 (동작 과정 1.16과 1.19)과 선형 조절 (동작 과정 1.17과 1.20), 그리고 특정 조절 (동작 과정 1.3과 1.21) 방식을 사용한다. 첫 번째 단계에서는 CPU 상태에 따라 설정 가능한 이상적인 주파수 (idealFreq)를 설정한다 (동작 과정 1.3부터 1.9까지). CPU 상태가 휴면 상태에서 활성 상태로 전환되는 경우 (동작 과정 1.3의 WAKEUP fromSLEEP 상태), 최고 CPU 주파수로 설정되어 있는 활성 전이 상태 주파수 (SLEEP_ WAKEUP_ FREQ) 값으로 CPU 주파수를 변경하여 사용자에게 빠른 응답성을 제공한다 (동작 과정 1.4). CPU가 휴면 상태인 경우 (동작 과정 1.6), 이상적인 주파수는 CPU 최고 주파수의 0.24배로 설정된 휴면 상태 주파수 (SLEEP_IDEAL_ FREQ) 값으로 설정된다. CPU가 활성 상태인 경우 (동작 과정 1.8), 이상적인 주파수는 최고 CPU 주파수의 0.63배로 설정된 활성 상태 주파수 (AWAKE_IDEAL_ FREQ) 값으로 설정된다.

두 번째 단계에서는 CPU 주파수의 증감 방향을 결정한다. 현재 CPU 부하가 최고 CPU 부하보다 크고, 이전 CPU 주파수 (prevCPUfreq)가 최고 CPU 주파수보다 작은 경우를 먼저 고려한다. 이러한 조건에서 이전 CPU 주파수가 이상적인 주파수보다 작거나 이전 CPU 주파수의 사용 시간이 상향 시간 (UP_TIME)보다 크면, CPU 주파수 설정 방향을 상향 (UP)으로 설정한다 (동작 과정 1.10과 1.11). 이전 CPU 주파수가 이상적인 주파수보다 크거나 이전 CPU 주파수의 사용 시간이 하향 시간 (DOWN_TIME)보다 크면, CPU 주파수 설정 방향을 하향 (DOWN)으로 설정한다 (동작 과정 1.12와 1.13). 최고 CPU 부하와 최저 CPU 부하 (MIN_CPU_ LOAD) 값은 50과 25이다. 상향 시간과 하향 시간 값은 각각 99msec와 48msec이다.

마지막 단계에서는 임계 및 선형 조절 방식을 사용한다. 상향 단계 (RAMP_UP_STEP)와 하향 단계 (RAMP_DOWN_STEP) 값은 동일하게 최고 주파수의 0.24배를 곱한 값이다. 먼저 CPU 주파수의 증감 방향 (동작 과정 1.15와 1.18)에 따라 이상적인 주파수까지는 빠르게 증감한다 (동작 과정 1.16과 1.19). 그 이후부터는 선형 조절 방식을 사용하여 상향 혹은 하향 단계만큼 이전 CPU 주파수를 증가 혹은 감소시킨다 (동작 과정 1.17과 동작 과정 1.20).

Table. 5 DPM governor procedures

```
1. procedure DPM:
1.1   if (CPUstate == SLEEP)
1.2     CPUfreq = CPUminFreq;
1.3   else if (CPUstate == AWAKE)
1.4     CPUfreq = CPUmaxFreq;
```

표 5는 DPM 거버너의 동작 과정이다. 결정 인자는 CPU 상태이다. 결정 시기는 CPU 상태 (동작 과정 1.1 및 동작 과정 1.3)에 따라 CPU 주파수를 변경하는 비주기 방식이다. 결정 방식은 특정 조절 방식을 사용하여 CPU가 활성 상태 혹은 휴면 상태일 때 최고 혹은 최저 CPU 주파수로 설정한다 (동작 과정 1.2와 1.4).

Table. 6 DPMSampling governor procedures

```
1. procedure DPMSampling:
1.1   Every governor_timer_rate:
1.2    if (CPUstate == AWAKE)
1.3       CPUfreq  = CPUmaxFreq;
1.4    else if (CPUstate == SLEEP &&
                CPUsleepTime > MIN_SLEEP_TIME)
1.5             CPUfreq = CPUminFreq;
```

표 6은 DPMSampling 거버너의 동작 과정을 보여준다. 결정 시기는 주기 방식이다 (동작 과정 1.1). 결정 인자는 CPU 상태이다. CPU가 실행 상태 (동작 과정 1.2)일 때는 DPM 거버너와 유사하게 특정 조절 방식을 사용하여 최고 CPU 주파수로 설정한다 (동작 과정 1.3). CPU의 휴면 시간이 최소 휴면 시간 (MIN_SLEEP_ TIME)보다 큰 경우에만 CPU 주파수를 최저 CPU 주파수로 변경한다. 이는 CPU 주파수의 감소 시점을 더디게 한다. 최소 휴면 시간 값은 20ms이다.

한편, DVFS기반 실시간 스케줄링은 태스크의 실행우선 순위에 따른 CPU의 선점 권한을 제어하여 저전력 및 실시간성을 제공한다. 본 논문에서는 많이 알려져 있는 SVS (Static Voltage Scaling)와 CC-DVS

(Cycling-Conserve Dynamic Voltage Scaling) 기반 EDF (Earliest Deadline First) 기반 스케줄링을 사용하였다 [10]. 태스크 속성을 기술하는 변수는 다음과 같다. i는 태스크 식별자이다. $Task(D)_i$는 마감시한, $Task(WCET)_i$는 최악 실행시간 (WCET: Worst-Case Execution Time), 그리고 $Task(ACET)_i$는 실제 실행시간 (ACET: Actual-Case Execution Time)을 나타낸다.

Table. 7 SVS procedures

1. procedure Select_Frequency():

1.1 CPUavailFreq[ ] = {CPUminFreq < ..... < CPUmaxFreq};
1.2 CPUutil = Task(WCET)$_1$ / Task(D)$_1$ + ...
+ Task(WCET)$_n$ / Task(D)$_n$;
1.3 CPUfreq = CPUmaxFreq;
1.4 foreach i in CPUavailFreq
1.5 if (CPUutil ≤ CPUavailFreq[i] / CPUmaxFreq)
1.6 CPUfreq = min (CPUFreq, CPUavailFreq[i]);
1.7 return CPUfreq;

표 7은 SVS 기법의 동작 과정을 보여준다. n은 태스크의 총 개수를 나타낸다. 먼저, 사용 가능한 CPU 주파수를 오름 순서로 배열 CPUavailFreq[]에 저장한다 (동작 과정 1.1). 태스크의 최악 실행시간 대비 마감시한 비율로 계산되는 CPU 이용률 (동작 과정 1.2)은 수식 (1)과 같다.

$$CPUutil = \sum_{i=1}^{n} \frac{Task(WCET)_i}{Task(D)_i} \quad (1)$$

사용가능한 CPU 주파수 (동작 과정 1.1의 CPUavailFreq[]) 중에서 $\frac{Task(WCET)_i}{Task(D)_i} \times CPUmaxFreq$만큼 선형적으로 감소시켜도 태스크 i의 마감시한을 만족시킬 수 있는 CPU 주파수를 선택한다. 만약 전체 태스크를 고려하는 경우, 전체 태스크의 마감 시한을 만족시킬 수 있는 CPU 주파수는 수식 (2)와 같이 계산할 수 있다.

$$CPUavailFreq = \sum_{i=1}^{n} \frac{Task(WCET)_i}{Task(D)_i} \times CPUmaxFreq \quad (2)$$

$$CPUfreq = \min(CPUmaxFreq, CPUavailFreq) \quad (3)$$

수식 (1)과 (2)를 사용하여 수식 (3)을 유도할 수 있다. 수식(1) 부터 (3)을 사용하여 전체 태스크의 마감 시한을 만족시킬 수 있는 최적의 CPU 주파수를 계산한다 (동작 과정 1.1부터 1.7까지).

Table. 8 CC-DVS procedures

1. Initialization:

1.1 CPUavailFreq[ ] = {CPUminFreq < ..... < CPUmaxFreq};
1.2 CPUutil[ ] = {Task(WCET)$_1$ / Task(D)$_1$ ...
Task(WCET)$_n$ / Task(D)$_n$};

2. procedure Select_Frequency():

2.1 CPUfreq = CPUmaxFreq;
2.2 totalCPUutil = CPUutil[1] + ... + CPUutil[n];
2.3 foreach i in CPUavailFreq
2.4 if (totalCPUutil ≤ CPUavailFreq[i] / CPUmaxFreq)
2.5 CPUfreq = min (CPUfreq, CPUavailfreq[i]);
2.6 return CPUfreq;

3. procedure Task_Release(Task$_i$):

3.1 CPUutil[i] = Task(WCET)$_i$ / Task(D)$_i$;
3.2 CPUfreq = Select_Frequency();

4. procedure Task_Completion(Task$_i$):

4.1 CPUutil[i] = Task(ACET)$_i$ / Task(D)$_i$;
4.2 CPUfreq = Select_Frequency();

표 8은 CC-DVS 기법의 동작 과정을 보여준다. 개별 태스크의 CPU 이용률을 배열 CPUutil[]에 저장한다. 태스크의 실행 요청이 있는 경우, SVS 기법과 유사하게 최악 실행 시간을 사용하여 태스크 이용률을 계산한 후 (동작 과정 3.1), Select_Frequency() 함수를 통해 CPU 주파수를 재설정한다 (동작 과정 3.2). 태스크의 실행이 완료되면, 태스크의 실제 실행 시간을 사용하여 태스크의 이용률을 재계산한 후 (동작 과정 4.1), Select_Frequency() 함수를 통해 CPU 주파수를 재설정한다 (동작 과정 4.2). CC-DVS는 태스크의 실제 실행 시간도 고려하기 때문에 최악 실행시간만을 고려하는 SVS 기법보다 더 낮은 CPU 주파수를 사용할 수 있다.

Ⅲ. 성능 분석

성능 평가를 위해 C언어 기반의 시뮬레이션을 구현하였다. 성능 평가에서 고려한 CPU 환경은 Qualcomm

QSD8250 CPU이다. QSD8250 CPU는 최고 동작 주파수 1GHz에서 240mA/sec, 245MHz에서 70mA/sec, 그리고 최저 동작 주파수 19.2MHz에서 25.62mA/sec만큼의 동작 전류량을 소비한다[11]. 개별 CPU 이용률마다 100개의 태스크 집합을 생성하였다. 각 개별 태스크 집합은 최대 20개 이하의 주기 태스크들로 랜덤하게 구성된다. 각 개별 태스크 집합의 시뮬레이션 시간은 7000초로 설정하였다. 태스크의 최악 실행 시간과 마감 시한은 최고 CPU 주파수 1GHz를 기준으로 하여 설정하였다. 성능 분석에 적용한 스케줄링 방식은 FCFS (First Come First Service) 기반 비실시간 스케줄링과 EDF 기반 실시간 스케줄링이다. 그리고 작업 실행 시간이 짧은 작업을 먼저 실행하는 TFS (Task First with Shortest execution time) 정책을 적용시켰다[12]. TFS 정책을 통해 작업들의 평균 응답 시간을 빠르게 하고 실행 대기중인 작업들의 마감시한 초과율을 감소시키고자 하였다. 성능 분석에 사용되는 스케줄링 방식은 TFS 정책을 사용하지 않는 FCFS와 TFS 정책을 사용하는 FCFS_TFS, 그리고 EDF이다.

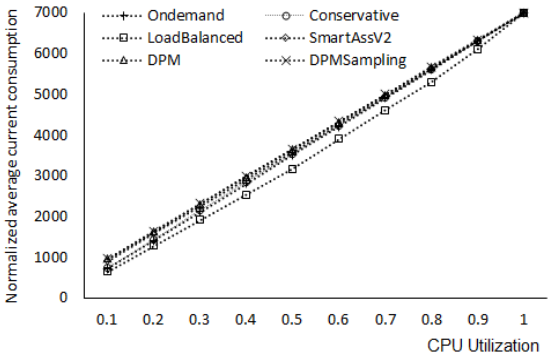

Fig. 1 Average current consumption by FCFS

그림 1은 FCFS 스케줄링이 적용된 거버너들의 평균 전류 소비량을 보여준다. 평균 전류 소비량은 시뮬레이션 시간 7000초를 기준으로 하여 0에서 부터 7000사이의 값으로 정규화하였다. 그림 1부터 그림 4까지의 실험 환경에서 태스크의 최악 실행 시간과 실제 실행 시간은 동일하게 설정하였다. LoadBalanced 거버너는 CPU 부하가 99로 설정된 최고 CPU 부하에 도달하기 전까지 CPU 부하에 비례하여 동작 주파수를 조절하기 때문에 전류 소비량이 가장 적다. Ondemand 거버너와 Conservative 거버너인 경우, CPU 부하가 80으로 설정된 상향 임계보다 크면 바로 최고 CPU 주파수로 동작한다. 따라서 LoadBalanced 거버너보다 CPU 증가 속도가 빠르다. 또한 하향 격차 방식으로 인하여 CPU 주파수의 감소 속도가 LoadBalanced 보다 느려 전류 소비량이 많다. SmartAssV2 거버너인 경우, CPU 부하가 50으로 설정된 최고 CPU 부하보다 크면, CPU 주파수는 최고 CPU 주파수의 0.63배로 설정된 활성 상태 주파수로 빠르게 변경된다. 따라서 Ondemand와 Conservative 거버너보다 다소 많은 전류를 소비한다. DPM 및 DPMSampling 거버너인 경우, CPU 활성 상태에서 최고 CPU 주파수로 즉시 동작하기 때문에 다른 거버너들보다 많은 전류량을 소비한다. 특히 DPMSampling 거버너는 최소 휴면 시간이 지나야 최저 CPU 주파수로 변경할 수 있기 때문에 DPM보다 더 많은 전류량을 소비한다. FCFS_TFS 및 EDF 스케줄링은 FCFS 스케줄링 대비 최대 0.00015% 더 많은 전류를 소비하였다.

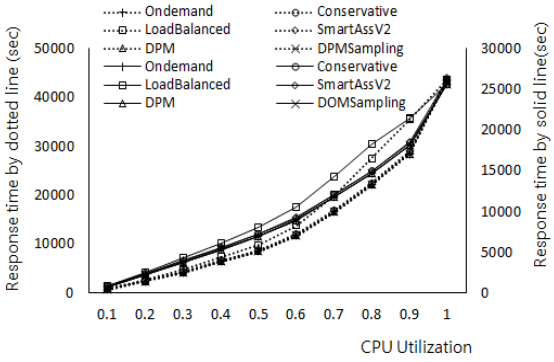

Fig. 2 Average response time of tasks by FCFS and FCFS_TFS

그림 2는 FCFS와 FCFS_TFS 스케줄링에서 태스크의 평균 총 응답 시간을 보여준다. FCFS 스케줄링은 점선을 사용하고, 왼쪽 세로축에 결과를 표시하였다. FCFS_TFS 스케줄링은 실선으로 사용하고, 오른쪽 세로축에 결과를 표시하였다. FCFS 스케줄링에서 LoadBalanced 거버너를 사용한 경우, 다른 거버너보다 응답 시간이 느렸다. 이는 그림 1에서 기술한 바와 같이 LoadBalanced 거버너에서 설정되는 CPU 주파수가 다른 거버너에 비해 낮아 태스크의 실행 완료 속도가 느려졌기 때문이다. FCFS_TFS 스케줄링을 사용한 경우,

모든 거버너에서 태스크의 평균 응답 시간은 전체적으로 감소하였다. LoadBalanced 거버너인 경우, CPU이용률 1에서 약 41% 의 성능 향상을 보여주었다.

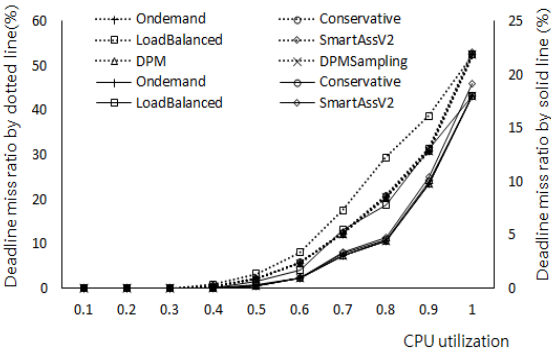

Fig. 3 Average deadline miss ratio of tasks by FCFS and FCFS_TFS

그림 3은 FCFS와 FCFS_TFS 스케줄링이 적용된 거버너들의 평균 태스크 마감시한 초과율을 보여준다. 그림 3의 그래프 표시선과 세로축의 표시 방식은 그림 2와 동일하다. FCFS 스케줄링인 경우, CPU 이용률 0.5 이상부터 모든 거버너에서 마감 시한 초과가 발생하였다. 다른 거버너에 비해 낮은 CPU 주파수를 설정하는 LoadBalanced 거버너는 다른 거버너에 비해 태스크의 마감시한 초과율이 높다. FCFS_TFS 스케줄링을 사용한 경우, 전체적으로 태스크의 마감 시한 초과는 많이 감소하였다. 특히, LoadBalanced 거버너는 CPU 이용률 1에서 FCFS 스케줄링 대비 약 34%의 성능 향상을 보여주었다.

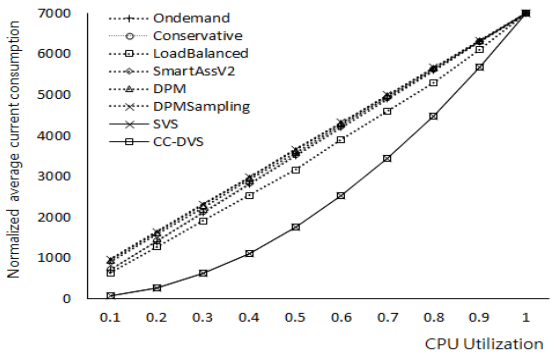

Fig. 4 Average current consumption by EDF

그림 4는 EDF 스케줄링이 적용된 거버너들의 평균 전류 소비량을 보여준다. EDF와 FCFS_TFS 스케줄링 간의 총 전류 소비량 차이는 ±0.05 정도로 매우 미미하였다. 태스크의 실제 실행 시간과 최악 실행 시간을 동일하게 설정하였기에 SVS와 CC-DVS의 성능은 동일하였다. SVS와 CC-DVS는 CPU 이용률에 따라 동작 주파수를 비선형으로 설정한다. 따라서 그림 4에서 보는 바와 같이, 전류 소비량의 변화도 비선형 형태로 나타난다. CPU 이용률 0.9까지는 CC-DVS, SVS, LoadBalanced, Ondemand, Conservative, SmartAssV2, DPM, 그리고 DPMSampling 거버너 순으로 전류를 많이 소비한다. 따라서 CC-DVS와 SVS가 설정하는 CPU 주파수가 다른 거버너보다 낮음을 알 수 있다. CPU 이용률이 1인 경우, 모든 거버너와 SVS 및 CC-DVS는 최고 CPU 주파수로 설정하기에 전류 소비량은 동일하였다. EDF 스케줄링인 경우, 모든 CPU 이용률에서 태스크의 마감 시한 초과는 발생하지 않았다.

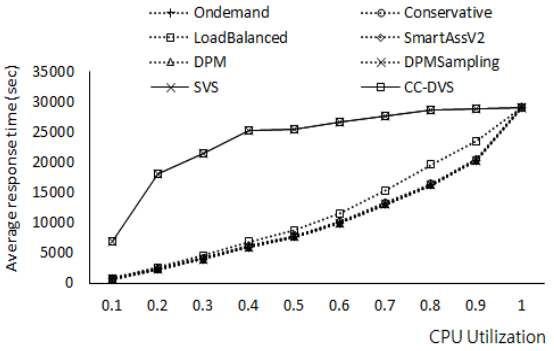

Fig. 5 Average response time of tasks by EDF

그림 5는 EDF 스케줄링에서 태스크의 평균 총 응답 시간을 보여준다. SVS와 CC-DVS는 태스크의 마감시한을 초과하지 않는 범위 내에서 CPU 주파수를 최대한 낮추어 설정한다. 이로 인해 태스크의 실행 시간이 길어져 태스크의 응답 시간도 길어진다.

그림 6과 7에서 태스크의 실제 실행시간은 최악 실행 시간 범위 내에서 랜덤하게 생성하였다. 이러한 실험 환경에서 마감 시한 초과율이 발생한 거버너는 없었다. 그림 6에서 전체 태스크의 최악 실행 시간 대비 마감 시한 비율 0.5까지는 SVS와 CC-DVS가 낮은 전류 소비 전류량을 보여준다. 그러나 전체 태스크의 최악 실행

시간 대비 마감 시한 비율이 0.6이상인 경우, 다른 거버너들 보다 많은 소비 전류량을 보여준다. SVS인 경우, 태스크의 최악 실행 시간만을 기준으로 CPU 주파수를 설정하고 실제 실행 시간을 CPU 주파수 설정에 반영하지 않기 때문이다 (표 7의 동작 과정 1.2). CC-DVS인 경우, 실행 초기 혹은 실행 대기 중인 태스크가 없는 경우에는 태스크의 최악 실행 시간만을 고려하기 때문에 CPU 거버너에 비해 많은 전류 소비량을 보여준다. 그러나 Ondemand를 포함한 CPU 거버너들은 실제 CPU 부하를 반영하여 CPU 주파수를 동적으로 조절하기 때문에 SVS와 CC-DVS 보다 낮은 전류 소비량을 보여준다.

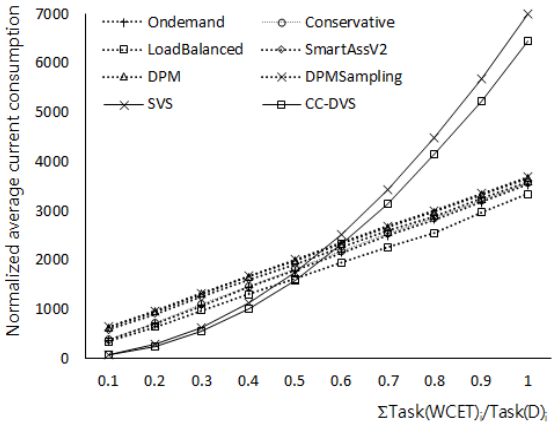

Fig. 6 Average current consumption by EDF over varying actual execution times

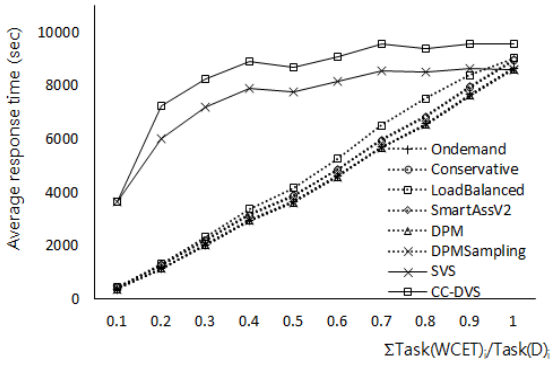

Fig. 7 Average response time of tasks by EDF over varying actual execution times

그림 7은 EDF 스케줄링에서 태스크의 평균 총 응답 시간을 보여준다. CC-DVS는 태스크의 이른 실행 완료가 발생하면, 태스크의 실제 실행 시간을 CPU 주파수 설정에 사용한다. 따라서 CC-DVS는 SVS보다 낮은 동작 주파수를 사용할 수 있다. 이에 SVS보다 낮은 응답 시간을 보여준다.

Ⅳ. 결 론

본 논문에서는 안드로이드 CPU 거버너와 DVFS기반 실시간 스케줄링 기법을 비교 분석하였다. 첫째, 태스크 스케줄링의 유형과 관계없이 개별 거버너가 소비하는 전류량의 차이는 미미하였다. 그러나 태스크 스케줄링과 거버너간의 상관관계에 대한 추가 연구가 필요하다. 둘째, TFS 정책을 개별 거버너에 적용한 결과, 전력 소비량의 증가에 거의 영향을 끼치지 않으면서, 태스크의 응답 시간 및 마감 시한 측면에서 성능 향상을 보여 주었다. 셋째, CPU 이용률이 높은 경우, 거버너는 전류 소비량 감소에 큰 영향을 주지 못하였다. 넷째, EDF 스케줄링 환경에서는 기존의 거버너들도 태스크의 실시간성을 보장하였다. 다섯째, SVS와 CC-DVS 기법이 항상 거버너보다 우수한 성능을 보여주는 것은 아니었다. 마지막으로, LoadBalanced 거버너는 다른 거버너에 비해 낮은 전류 소비량을 보여 주었다. 그러나 CPU 부하가 증가하는 경우, 특정 주파수로 바로 전환할 수 없어 태스크의 응답 시간 및 마감 시한 초과율도 증가하였다. 그리고 SmartAssV2는 다른 거버너보다 복잡한 알고리즘을 사용하지만 이로 인해 성능 개선은 미약함을 보여 주었다. 향후 분석한 결과를 기반으로 하여, 거버너와 DVFS 기법의 장점이 수용된 CPU 거버너를 설계하고자 한다.

REFERENCES

[1] M. Kim, Y. G. Kim, and S. W. Chung, “Measuring variance between smartphone energy consumption and battery life,” *IEEE Computer*, vol. 47, no. 7, pp. 59-65, July 2014.

[2] Y. Zhu, M. Halpern, and V. J. Reddi, “The role of the CPU

in energy-efficient mobile web browsing," *IEEE Micro*, vol. 35, no. 1, pp. 26-33, Jan. 2015.

[3] V. Seeker, P. Petoumenous, H. Leather, and B. Franke, "Measuring QoE of Load balanced workloads and characterizing frequency governors on mobile devices," in *Proceedings of the IEEE International Symposium on Workload Characterization*, Raleigh: NC, pp. 61-70, 2014.

[4] V. Pallipadi and A. Starikovskiy, "The ondemand governor," in *Proceedings of Linux Symposium*, Ottawa: Canada, pp. 223-238, 2006.

[5] W-Y. Liang and P-T Lai, "Design and implementation of a critical speed-based DVFS mechanism for the android operating system," in *Proceedings of the 5th International Conference on Embedded and Multimedia Computing*, Cebu: Philippine, pp. 1-6, 2010.

[6] A. Shye, B. Scholbrock, and G. Memik, "Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, New York: NY, pp. 168-178, 2009.

[7] J. Yoo, S. Huh, and S. Hong, "Limitations on DVFS-based power saving mechanisms of android smartphones," *Communications of the Korea Information Science Society*, vol. 30, no. 7, pp. 9-16, July 2012.

[8] T. Burd, and R. Brodsersen, "Energy efficient CMOS microprocessor design," in *Proceedings of the 28th Annual Hawaii International Conference on System Sciences. Volume 1: Architecture*, Hawaii: USA, pp. 288–297, 1995.

[9] V. Spiliopoulos, A. Bagdia, A. Hansson, P. Aldworth, and S. Kaxiras, "Introducing DVFS-management in a full-system simulator," in *Proceedings of the 21st International Symposium on Analysis and Simulation of Computer and Telecommunication Systems*, San Francisco: CA, pp. 535-545, 2013

[10] 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*, New York: NY, pp. 89-102, 2001.

[11] 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.

[12] S. Tak, T. Kim, and E. K. Park, "Integrating real-time inter-task communication channels into hardware-software codesign," *Microprocessors and Microsystems*, vol. 34, no. 6, pp. 182-199, June 2010.

탁성우(Sungwoo Tak)

2003년 2월 미국미주리주립대학교 Computer Science 박사
2004년 ~ 현재 부산대학교 정보컴퓨터공학부 교수
※관심분야 : 네트워크, 위치인식, 스마트폰 CPU 거버너