A Middleware for Power Management in Multicore Smartphones

Shaosong Li
University of Colorado at Boulder
shaosong.li@colorado.edu

Shivakant Mishra
University of Colorado at Boulder
mishras@cs.colorado.edu

Abstract

Increased power consumption is a critical concern for smartphone users. While multi-core processors in smartphones have already emerged in market, current applications are yet to take full advantage of this new architecture, particularly in the area of managing power consumption. This paper addresses the issue of managing power consumption in multicore smartphones via a middleware layer that schedules optimal number of cores for currently running applications taking into account the tradeoff between power consumption, performance and user experience. The paper first describes a simple and accurate method to measure the overall power consumption and then studies the impact of scheduling seven different popular applications over one to four cores on the overall power consumption. Based on this study, the paper proposes three new power-aware scheduling algorithms that dynamically schedule optimal number of cores as well as dynamically adjust the voltage frequency of each online core to achieve the best tradeoff between power consumption, application performance and user experience under the current context. Evaluation from a prototype implementation of the middleware on a quad-core HTC One shows that these algorithms result in significant reduction in power consumption while ensuring good performance and user experience.

Keywords

Multi-core smartphones, Power management, Scheduling

1. INTRODUCTION

Mobile use cases such as HD video playback, streaming video and audio, multitasking, browsing the web, 3D gaming and 3D interfaces have become feasible with the availability of multi-core architectures in mobile devices. High performance computing components in mobile devices, however, consume significant amount of power reducing battery life, which is one of the most critical concern mobile users have. Although multi-core processors are meant to be energy efficient units, increasing the number of cores can drain the battery much more rapidly than low power, single-core embedded processors, particularly if the cores are run at high voltage frequencies. Since more and more applications are being developed as multi-threaded and users are increasingly executing multiple tasks concurrently, the issue of managing the overall power consumption in multicore smartphones is of paramount importance.

Techniques to reduce the overall power consumption in smartphones can be divided into three high level categories. The first category deals with designing power-aware applications, wherein utilization of power hungry hardware resources is minimized and power efficient alternatives to power hungry scenarios are incorporated while recognizing trade-offs between extended battery life and user experience or performance. Examples include [5] and [7]. The second category deals with dynamic voltage and frequency scaling (DVFS), wherein voltage or frequency of processor is dynamically adjusted "on the fly" to conserve power. Examples include [17]. The third category deals with efficient utilization of the multiple CPUs of a device. When a smartphone is idle or lightly used, some cores may be turned off. Techniques such as hot plugging turn off one or more cores when not needed and thus save power. These three categories of power management techniques are complimentary with one another.

This paper takes a holistic view of *managing the overall power consumption* in multi-core smartphones. While reducing the overall power consumption is certainly an important aspect of managing power, there are several other issues that must be considered as well. First, there are technical issues that include how to accurately measure the overall power consumption in modern smartphones, and understanding how the overall power consumption is affected by the number of cores that are online at any point in time and the frequencies at which these online cores are running. Second, there are three quality indicators that must be considered together: power consumption, application performance and user experience. Under some circumstances, it may be reasonable to sacrifice application performance or user experience to save power, while at other instances, it may be reasonable to improve performance at the expense of increased power consumption. It is important to understand the tradeoffs between power consumption, application performance and user experience. Finally, contextual issues such as the current battery level or expected time interval before next phone charge have an impact on the relative importance of reducing power consumption and performance.

MOBIQUITOUS 2015, July 22-24, Coimbra, Portugal

DOI 10.4108/eai.22-7-2015.2260049

This paper proposes a middleware layer that addresses all these issues to manage power in modern, multi-core smartphones. It dynamically schedules an optimal number of cores online and dynamically adjusts the frequencies of each of the online cores based on the current CPU load and a *balance factor*. Balance factor is used to determine the right tradeoff between power, performance and user experience, and the CPU load is then used to determine the right number of cores and their frequencies to realize that tradeoff.

The paper first describes a simple and accurate method to measure the overall power consumption in smartphones, and then studies the impact of scheduling seven different popular smartphone applications over one to four cores on the overall power consumption. Next, the paper discusses the important issue of tradeoff between power, performance and user experience as well as the impact of remaining battery life and expected time interval for charging the phone next. This tradeoff is then abstracted via *balance factor*. The paper then proposes three new power-aware scheduling mechanisms that dynamically schedule optimal number of cores and then determine the optimal frequency for each core based on the balance factor and current CPU load. These mechanisms have been implemented on Android HTC One smartphone, which is quad-core. Prototype evaluation from seven popular applications shows that these mechanisms can reduce the overall power consumption of the smartphone by as much as 40% over the current default Android scheduler.

2. RELATED WORK

To prolong battery life, most off-the-shelf smartphones and tablets adopt power management schemes that make use of dynamic voltage and frequency scaling (DVFS) [16,18] and processor hot plugging [17]. In [13], the authors have analyzed the effectiveness of various power management schemes for multi-core smartphone systems in terms of energy efficiency and user-perceived response latency. In [14], the authors have created an application framework that allows execution of different types of threads, comparing their efficiency and measuring power consumption on mobile devices. In [15], the authors have created an online power estimation technique for multi-core smartphones with advanced display components. However, none of these prior work propose new scheduling mechanisms for multi-core smartphones.

Measurement of power consumption in smartphones has also been extensively addressed. PowerBooter is an automated power model construction technique that uses built-in battery voltage sensors and knowledge of battery discharge behavior to monitor power consumption while explicitly controlling the power management and activity states of individual components [4]. AppScope, an Android-based energy metering system that monitors application's hardware usage at the kernel level and accurately estimate energy consumption is proposed in [5]. A tool to profile the energy usage of applications is proposed in [6]. Finally, a new, system-call-based power modeling approach that gracefully encompasses both utilization-based and non-utilization-based power behavior is proposed in [8]. Besides software measurements, MonSoon [9] provides accurate hardware-based measurement. However, all these prior research in power modeling apply to single-core smartphones.

The default scheduler for Linux kernel since the 2.6.23 release is CFS (Completely Fair Scheduler). Android kernel uses CFS with slight changes to the CPU process scheduler and time-keeping algorithms. Linux kernel also has a number of CPU frequency governors that vary CPU frequencies based on certain criteria. The default CPU governor for Android kernel 3.4.10 is OnDemand. This governor has a hair trigger for boosting clock speed to the maximum speed set by the user. If the CPU load placed by the user abates, the OnDemand governor slowly steps back down through the kernel's frequency stepping until it settles at the lowest possible frequency. The Performance governor locks the phone's CPU at maximum frequency aimed to maximize performance. The Powersave governor on the other hand locks the CPU frequency at the lowest frequency set by the user aimed to minimize performance consumption.

3. POWER MEASUREMENT

Power measurement using external metering such as MonSoon is expensive and require opening the battery compartment, which is not allowed in newer smartphones. We propose a software approach that is simple to use and highly accurate. Newer Android phones provide current and voltage information in system files that are regularly updated. Current value is provided in the system file: */sys/classes/power_supply/battery/bat_current_now*, and the voltage value is provided in another system file: */sys/classes/power_supply/battery/bat_voltage_now*. So, it is straightforward to measure power consumption by reading the current and voltage values from these files and calculating the power ($Power = Voltage \times Current$). However, it is not clear how accurate these current and voltage values in the system files are.

So, to assess the accuracy of power measurement using this software approach, we developed an Android service to log these values at regular intervals in a file. Android OS updates both of these files regularly at the same time interval. However, the time interval rate differs between different hardware. The test phone we used was Google Nexus One. The reason for using this phone is that it is easy to open the battery chamber of this phone, and so it allows us to compare the power measured via our software approach with the power measured via the hardware approach such as MonSoon, which has been shown to be highly accurate. In Google Nexus One, the battery information is updated nearly every 50 seconds. So, we logged the current and voltage values very minute for 30 minutes. In addition, we used Agilent 34411 A Multimeter to measure the current draw. To measure the voltage, we used a Fluke 45 Dual Display Multimeter, in which the sense resistors are connected via twisted-pair wiring. We wrote a program to control the two instruments to sample data per minute and saved the current and voltage values with time stamp in a second log file.

3.1 Accuracy of Software Approach

Power measurement using both the software approach and the hardware approach was done at the same time. Figure 1 shows the values of voltages measured using the two approaches. We can see that the range for the voltage is $3840mV$ to $3930mV$. Average voltage value measured by instrument is $3869.148mV$ while the average voltage value measured by the software application is $3881.444mV$. The difference in the average voltage values is only 0.318%. Fig-

Application	Usage details
Facebook	Refresh new posts; Comment on new posts; view images and articles
Pandora	Moderate volume; Shuffle play album
Facebook_Pandora	Run Facebook and Pandora at the same time
Candy Crush	Normal play operations; Background music always on
Google Maps	Search new places; Navigation simulation; Street view; Zoom in and zoom out operations
Youtube	Play HD movie
MPEG Convertor	Convert video from .avi to .mp4 format

Table 1: Usage details of applications. Wi-Fi connection is used in all applications

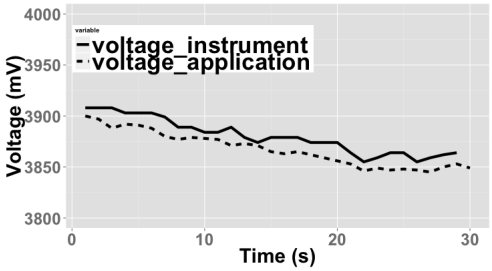

Figure 1: Accuracy of voltage value

ure 2 shows the values of current measured using the two approaches. We can find that the range for the current is $110mA$ to $220mA$. Average current value measured by the instrument is $131.741mA$, while the average current value measured by the software application is $135.444mA$. The difference in the average current values is only about 5.322%. Since these differences are quite low, we conclude that the current and voltage information obtained from the system files is quite accurate and we can use it for measuring the overall power consumption in the smartphone.

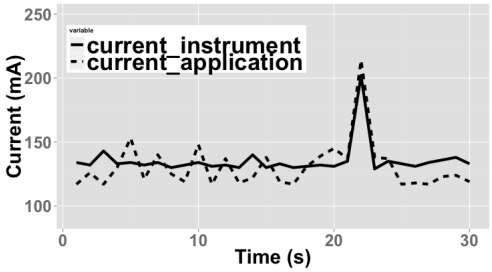

Figure 2: Accuracy of current value

4. IMPACT OF NUMBER OF CORES

To understand the impact of running different number of cores on the overall power consumption, we have experimented with running a variety of applications and measuring power consumption along with CPU load and performance. Earlier research has shown that display, GPS and network consume the most amount of power in smartphones. So, we have chosen applications that use one or more of these components quite extensively and are very popular at present among Android phone users. In particular, we have chosen seven applications: Facebook, Pandora, Facebook_Pandora, Candy Crush, Google Maps, YouTube and MPEG Convertor. In all of our experiments, we used Wi-Fi connection for all applications. Table 1 shows the details of the sequence of actions for each of the seven applications that we used in all experiments.

All experiments were done on HTC One developer edition, which is quad-core and has 2 GB DDR2 RAM and 64 GB storage. The Android version is 4.2.2 and the Kernel version is 3.4.10. Current and voltage values are updated on this phone every 50 ms, which is much higher than the update rate on Google Nexus One. We mainly focus on application performance and thread distribution using CPU load as the main parameter. CPU load information is read from the system file $/proc/stat$. We have reported the overall CPU load resulting from all cores, ranging from 0.00 to 1.00.

4.1 Application Performance

We ran each application on one core, two cores, three cores and four cores respectively. On-Demand governor was used in each of these runs. We measured current, voltage, overall CPU load, CPU load for each core, and power consumption. Each application was run for five minutes and measurements were taken every second. Table 2 shows the results of running Pandora. We can see that the voltage values decrease as the number of online cores increases. However, current and power values do not have this linear property. All other applications also provided similar behavior. For Pandora, the least power consumption occurs under two-core scenario.

# of online cores	Current mA	Voltage mV	Power mW	Overall CPU load
1	504	4029	2034	0.10
2	491	3917	1922	0.127
3	551	3889	2143	0.16
4	625	3814	2385	0.18

Table 2: Power and CPU load for Pandora

Table 3 shows the CPU load and power consumption for each application. Here, the top line for each application shows the CPU load and the second line shows power consumption. We can see that for Facebook, Pandora, YouTube, Facebook_Pandora and MPEG Convertor, power consumption is lowest when using one or two cores. Power consumptions difference between one core and two core scenarios is relatively small in these applications. However, for Google Maps, and Candy Crush, power consumption is lower when running three or four cores. We also notice that there is high correlation between power consumption and overall CPU load. Lower CPU load generally implies lower power consumption.

Based on this observation of high correlation between CPU load and power consumption, our guiding scheduling principle is: as the CPU load increases beyond some threshold

Application	1 core	2 cores	3 cores	4 cores
Facebook	0.10	0.12	0.13	0.127
	2031	2014	2722	2403
Pandora	0.10	0.127	0.16	0.18
	2034	1922	2143	2385
Facebook	0.129	0.133	0.184	0.201
Pandora	2705	2755	3187	3224
Candy	0.317	0.309	0.305	0.297
Crush	4598	4464	4387	3849
Google	0.353	0.334	0.314	0.29
Maps	5457	4917	4714	4090
Youtube	0.125	0.178	0.206	0.21
	2760	2907	3103	3263
MPEG	0.134	0.145	0.166	0.171
Convertor	2322	2450	2441	2843

Table 3: CPU load and power for 7 apps. For each app, top line: CPU load, bottom line: power (mW)

value, start a new core, and similarly, as the CPU load decreases below some threshold value, shut down one core. To determine these threshold values, we need to take a closer look at the relation between the number of cores, power consumption and CPU load. One problem is that CPU load caused by each application changes over the application duration, i.e. an application may incur high CPU load at one point in time and a low load at another point in time. To determine the threshold values, we need an application that exhibits steady CPU load variation with a user-controlled knob to steadily increase or decrease the CPU load incurred. We address this issue in Section 6.

5. POWER, PERFORMANCE AND USER EXPERIENCE TRADEOFF

The proposed power management middleware needs to achieve a balance among three important goals: minimizing power consumed by the application, maximizing the performance it exhibits and providing a good user experience. These goals of course may conflict with one another. For example, power consumed by an application is generally lowest when the cores are run at the lowest frequency. However, running cores at the lowest frequency worsens application performance and may negatively impact user experience. So, it is important to incorporate a tradeoff mechanism to achieve an optimal balance among these three goals.

There are two important factors that have an impact on this tradeoff: application type and current context. Application type determines which of the three goals is more important than the others. For some applications, such as gaming and video streaming, user experience is the most important goal. For such applications, the middleware tries to maintaimaintain a good user experience at minimal power consumption. On the other hand, for high performance applications such as *mpeg* encoding and FFT calculation, performance is the most important goal. For such applications, the middleware tries to maintain an acceptable performance at minimal power consumption. In addition, relative importance of power, performance and user experience depends on at least two contextual features: remaining battery level and the expected time interval before the next charge. For example, if the remaining battery level is low and the next time to charge the phone is not too close, reducing power consumption at the expense of application performance or user experience is a reasonable tradeoff.

To arrive at an appropriate balance between power, performance and user experience, our middleware incorporates a *balance factor*. This balance factor can take one of seven values: +3, +2, +1, 0, -1, -2, -3. Positive values mean performance is more important and negative values mean reducing power consumption is more important. A balance factor of +3 means the performance is the most important goal and the system must be configured to achieve the best performance, no matter what the power cost is. On the other hand, a balance factor of -3 means power consumption is the most important goal and the system must be configured to minimize power consumption, no matter what the performance or user experience cost is. A value of +2 (or +1) means that the performance is important, but up to 5% (or 10%) of the performance may be sacrificed if that helps in reducing some power consumption. Similarly, a value of -2 (or -1) means that reducing power consumption is important, but up to 5% (or 10%) of power may be sacrificed if that helps in improving some performance. Finally, a balance factor of 0 means up to 10 % of power and performance may be sacrificed. The choice of 5% or 10% is arbitrary, and can be adjusted as needed. Compute a balance factor for an application under a particular contexual scenario is discussed in Section 7.4.

6. DESIGN AND IMPLEMENTATION

6.1 Optimal Number of Online Cores

CPU Load	# of online cores	Power (mW)	Performance (second)
0.02	1	967	5.13
	2	1083	5.16
	3	1064	5.09
	4	1123	5.13
...	...	...	...

Table 4: Relation between CPU load, number of cores, power and performance

To understand the relationship between total CPU load, number of online cores, power consumed and performance, we implemented a simple application that exhibits steady CPU load throughout its run, and the value of this load is user-controlled. This application consists of two nested loops and has three parameters, loop counts for the two loops and the number of threads created to execute these loops. These three parameters can be adjusted to vary the CPU load from 2% to 100% in steps of 2%. In other words, there are fifty versions of this application exhibiting CPU loads of 2%, 4%, 6%, ..., 100% respectively. For each CPU load ranging from 2% to 100%, we obtained power consumed and performance exhibited if the application is run on one core, two cores, three cores and four cores. The entire table is too large to include here. Table 4 illustrates one entry of this table.

From this table, we calculated the optimal number of cores for each CPU load and balance factor. For example, for a balance factor of -3, the optimal number of cores for a given

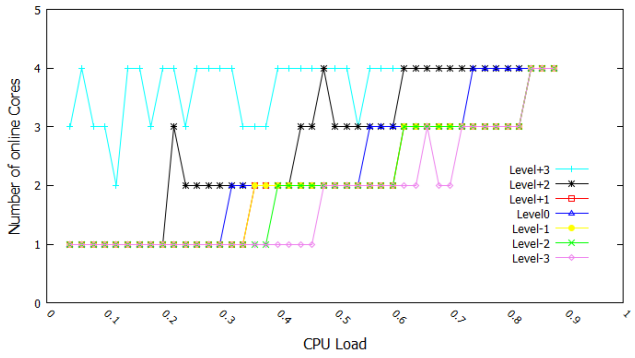

Figure 3: Optimal number of online cores for different balance factors and CPU loads

CPU load corresponds to the one that consumes least power for that CPU load. For a balance factor of +3, the optimal number of cores for a given CPU load corresponds to the one that provides the best performance for that CPU load. For a balance factor of −2 (or −1), the optimal number of cores for a given CPU load corresponds to the one whose performance is the best among those entries whose power consumption is with in 5% (or 10%) of the least power consumed for that CPU load. For a balance factor of +2 (or +1), the optimal number of cores for a given CPU load corresponds to the one whose power consumption is least among those entries whose performance is with in 5% (or 10%) of the best performance for that CPU load. Finally, for a balance factor of 0, the optimal number of cores for a given CPU load corresponds to the one where both the power onsumption and the performance are with in 10% of the best power consumption and best performance respectively. In a few cases, where there is no such entry, we gradually increased the percentage by one until we got an entry that satisfied this requirement.

Figure 3 illustrates the optimal number of online cores for each CPU load and balance factor. We can see from this figure that the optimal number of online cores for each balance factor starts at a lower number and generally increases as the CPU load increases. For example, for balance factor of −3, the optimal number of cores is one for CPU loads up to 46%, two for CPU loads between 48% and 70%, three for CPU loads between 72% and 82%, and four for CPU loads higher than 82%. Similarly, for a balance factor or +3, the optimal number of cores is generally three for CPU loads up to 38%, and four for CPU loads higher than 38%. We also notice that the optimal number of online cores for performance-oriented balance factors (+3, +2 or +1) is higher than or equal to the optimal number of online cores for power-oriented balance factors (−3, −2 or −1) for the same CPU load. This is expected, since smaller number of online cores in general result in lower power consumption but poorer performance.

Another observation we make is that the transition of the optimal number of online cores to the next higher number takes place at smaller CPU loads as we move from balance factors −3 to +3. For example, this transition from one to two cores for balance factor −3 occurs at $46-48\%$, while it occurs at $36-38\%$ for balance factor −2, $32-34\%$ for balance factor −1, and so on. This is expected, since smaller number of online cores in general result in lower power consumption but poorer performance.

We expect that for any given balance factor, the optimal number of online cores for a given CPU load will be higher than or equal to the optimal number of online cores for a lower CPU load. This holds true for almost all cases in Figure 3 with a few exceptions. There are a few instances in this figure where the optimal number of cores is actually lower than the one for a lower CPU load, e.g. CPU load 52% and balance factor +3. Also, there are a few instances where the optimal number of cores is actually higher than the one for a higher CPU load, e.g. CPU load 64% and balance factor −3. These are unexpected results. A closer examination of these unexpected values shows that there is very little difference (less than 1%) in power consumed or performance from the optimal number of online cores for adjacent CPU loads. For example, the performance value for 52% CPU load for three online cores is 6.71 sec while it is 6.69 sec for CPU load of 50% and four online cores or 6.73 sec for CPU load of 54% and four online cores. So, we attribute these unexpected results to the slight variations in operating conditions of different experiments.

Since there is some cost associated with bringing a new core online or shutting off a current online core, we ignore these unexpected transitions in order to minimize the number of transitions as CPU load changes. Table 5 shows the optimal number of online cores for different balance factors and CPU loads, calculated from the results shown in Figure 3 and ignoring the unexpected transitions.

Balance factor	One core	Two cores	Three cores	Four cores
+3			<=38%	>=40%
+2	<=18%	20% - 42%	44% - 58%	>=60%
+1	<=28%	30% - 54%	56% - 72%	>=74%
0	<=28%	30% - 54%	56% - 72%	>=74%
-1	<=34%	36% - 60%	62% - 82%	>=84%
-2	<=38%	40% - 60%	62% - 82%	>=84%
-3	<=46%	48% - 70%	72% - 82%	>=84%

Table 5: Optimal number of online cores for various CPU loads and balance factors

6.2 Optimal Voltage Frequency

CPU Load	Frequency Hz	Power mW	Performance (second)
0.02	384,000	844	5.23
	918,000	987	5.18
	1,135,000	1033	5.08
	1,728,000	1141	5.10
...	...	...	...

Table 6: Relation between CPU load, frequency, power and performance

The next question is at what frequencies should each of these online cores run? Clearly, this again depends on the value of the balance factor and the current CPU load on each core. A core in HTC One developer edition provides 14 different frequencies, ranging from $384,000\ Hz$ to $1,728,000\ Hz$. To understand the impact of frequency on power consumption and performance, we followed a similar methodology as the one we used in determining the optimal number of online cores for different CPU loads and balance factor values. We

ran our application for CPU loads ranging from 2% to 100% on a single core under four different frequencies: 384,000 Hz, 918,000 Hz, 1,350,000 Hz and 1,728,000 Hz. We chose these four frequencies as they provide a good coverage of the overall range of frequencies available and are commonly used by current CPU governors, including OnDemand, performance and powersave governors. For each run, we measured power consumption and performance. The entire table listing fifty different values of CPU loads, four different frequencies for each CPU load along with measured power consumption and performance is too large to include here. Table 6 illustrates some parts of this table. From this table, we calculated the optimal frequency for each CPU load and balance factor.

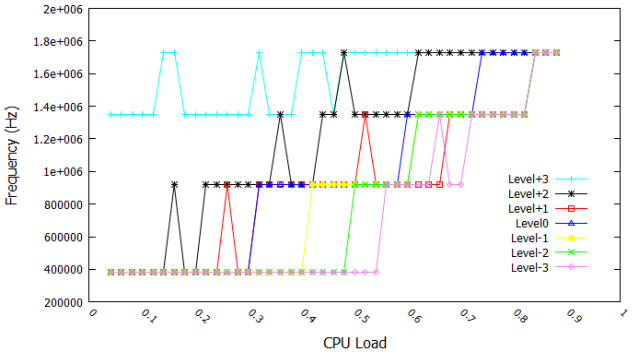

Figure 4: Optimal frequency for different balance factors and CPU loads

Figure 4 illustrates the optimal frequency for each CPU load and balance factor. Comparing this figure with Figure 3, we see that the variation of optimal frequency with increasing CPU loads and different balance factors follows a similar pattern as the variation of optimal number of online cores. Table 7 shows the optimal frequencies for different balance factors and CPU loads. This table is based on the results from Figure 4 and ignoring the unexpected transitions.

Balance factor	384000 Hz	918000 Hz	1350000 Hz	1728000 Hz
+3			<=38%	>=40%
+2	<=18%	20% - 38%	40% - 58%	>=60%
+1	<=28%	30% - 64%	66% - 82%	>=84%
0	<=28%	30% - 56%	58% - 72%	>=74%
-1	<=38%	40% - 60%	62% - 82%	>=84%
-2	<=46%	48% - 58%	60% - 82%	>=84%
-3	<=52%	54% - 70%	72% - 82%	>=84%

Table 7: Optimal frequency for various CPU loads and balance factors

6.3 Scheduling Mechanisms

We now propose three new scheduling mechanisms for multicore Android devices. Thees mechanisms are based on current CPU load and balance factor, and dynamically change the number of online cores as well as the frequency of each online core to maintain a balance between power consumption and performance. They operate in two stages. First, based on the overall CPU load and balance factor, they schedule the optimal number of cores to run using Table 5. Next, for each online core, they schedule the optimal frequency using Table 7 based on the CPU load on that core and the balance factor.

Threshold based scheduling: The first mechanism is *threshold based scheduling mechanism*. This mechanism strictly follows the optimal number of cores and optimal frequencies of Tables 5 and 7. For example, suppose two cores are online at present and the current CPU load goes to 44% and the balance factor is +2. The threshold based scheduling mechanism will bring in a third core online in this case. On the other hand, if the current CPU load comes down to 18%, the scheduling mechanism will shut off one of the current online core. A similar strategy is used for determining the frequency for each online core based on the CPU load on that core and the balance factor.

While this mechanism ensures that the number of online cores is always optimal, there is a potential problem for applications that exhibit frequent changes in CPU loads. For such applications, the scheduler will result in frequent core switching, which in turn will lead to increased power consumption resulting from increased thread migrations between different cores. The next two scheduling mechanisms attempt to reduce the number core switching while keeping an optimal number of cores online.

Time interval based scheduling: The *time interval based scheduling mechanism* relies on a preset time interval, *time_int*. For a given balance factor, whenever the CPU load crosses one of the thresholds for a change in the optimal number of cores, the middleware turns on or off a core only if the CPU load remains in the new threshold interval for at least *time_int* time units. A similar strategy is used for determining the frequency for each online core based on the CPU load on that core and the balance factor and a time interval.

Threshold interval based scheduling: The threshold interval based scheduling mechanism relies on a preset threshold interval, *thresh_int* to prevent frequent core switching. For a given balance factor, when the CPU load goes up, a new core is turned on only if the CPU load crosses threshold for a change in the optimal number cores plus *thresh_int*. Similarly, when the CPU load goes down, a core is turned off only if the CPU load crosses a threshold value for a change in the optimal number cores minus *thresh_int*. A similar strategy is used for determining the frequency for each online core based on the CPU load on that core and the balance factor and a threshold interval.

7. EVALUATION

Our evaluation of the three scheduling mechanisms consists of three parts. First, time interval based scheduling and threshold interval based scheduling mechanisms depend on the values of *time_int* and *thresh_int*. We have conducted several experiments to determine optimal values of these thresholds. Second, we evaluate the three scheduling mechanisms for individual applications with respect to the overall power consumption and performance for various values of balance factor. Finally, we explore the question of how to compute an appropriate value of the balance factor for a given application under various contextual scenarios.

7.1 Time Interval

Time interval based scheduling mechanism relies on time interval *time_int*. To determine an optimal time interval, we experimented with four different time intervals. Since the

highest update rate for HTC One is 50 ms, we experimented with time intervals of 50 ms, 200 ms, 500 ms and 1 second. For each time interval, we ran the seven applications using the time interval based scheduling mechanism for five minutes and recorded their power consumption with a balance factor of zero. Figure 5 shows the power consumption for seven applications under the four different scenarios.

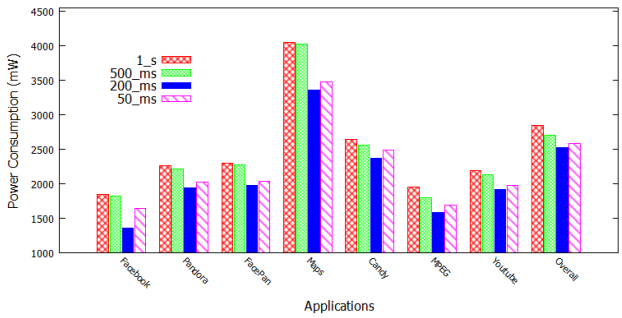

Figure 5: Power consumption for different time intervals

From this figure, we can see that the 200 ms time interval results in the lowest power consumption among the four time intervals. Also, we can already see that the time interval based scheduling mechanism does result in saving power when compared to the default Android scheduling mechanism (See Table 3). Based of these results, we decided to use $time\_int = 200$ ms for all our experiments.

7.2 Threshold Interval

Threshold interval based scheduling mechanism relies on the threshold interval value *threshold_int*. Based on Table 5, we observe that a switch in the optimal number cores can occur over a CPU load difference of as low as 15%, which means there would be overlap if we set the threshold interval to be more than 7.5%. So, we experimented with threshold intervals of 1%, 3%, 5%, and 7% to avoid this overlap. For each threshold interval, we ran the seven applications using the threshold interval based scheduling mechanism for five minutes and recorded their power consumption for a balance factor of zero. Figure 6 shows the power consumption for seven applications under the four different scenarios.

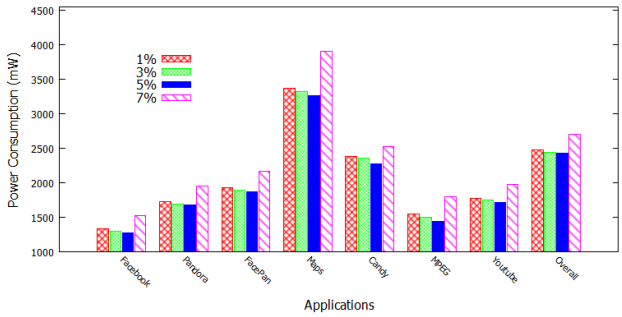

Figure 6: Power consumption for different threshold intervals

From the figure, we can see that the 5% threshold interval results in the lowest power consumption among the four threshold intervals. Also, we can already see that the threshold interval based scheduling mechanism does result in saving power when compared to the default Android scheduling mechanism (See Table 3). Based of these results, we decided to use $thresh\_int = 5\%$ for all our experiments.

7.3 Scheduling Mechanisms Evaluation

We have measured power consumption from running the seven popular applications under the three scheduling mechanisms for all values of balance factors. To get a better fine-grained understanding of our three schedulers, we have experimented with two variations. In *variation one*, we measured power consumption for each application under the three scheduling mechanisms when each of these scheduling mechanism schedule the optimal number cores but the frequency of each core is determined by the OnDemand governor. In *variation two*, we measured power consumption for each application under the three scheduling mechanisms when each of these scheduling mechanisms schedule the optimal number cores as well as the optimal frequency for each core. For comparison purposes, we also ran each of these applications using the default OnDemand governor, Performance governor and the Powersave governor.

Results are shown in Figures 7 and 8 for Facebook, Candy crush, Google Maps, YouTube, Pandora and MPEG converter. In both of these figures, the three horizontal lines show the power consumption under OnDemand governor (middle horizontal line), Performance governor (upper horizontal line) and the Powersave governor (lower horizontal line). The three vertical bars for each balance factor show the power consumption under variation two (optimal number of core and optimal frequency) of the three scheduling mechanisms. Finally, the symbols on top of each bars that are connected using lines show the power consumption under variation one (Optimal number of cores and OnDemand governor) of the three scheduling mechanisms.

The first important observation we make is that the three scheduling mechanisms provide significant reduction in power consumption compared to the current state of the art schedulers on Android. Even compared to the Powersave governor that provides the lowest power consumption in Android today, all three proposed scheduling mechanisms consume less power when balance factor is -3. Similarly, if the performance is important (balance factor $+3$), the three proposed scheduling mechanisms result in significantly lower power consumption than Performance governor. Table 8 provides the range of power savings for the seven applications as the balance factor is varied from $+3$ to -3 compared to the power consumption under OnDemand governror. We can see that maximum power saving is as high as 40.5%. However, it is important to note that this power saving comes at a cost to the application performance or user experience. This issue is further evaluated in the next subsection.

The second important observation we make is that both stages of the three scheduling algorithms, scheduling of optimal number of online cores in the first stage and scheduling optimal frequency in each online core in the second stage contribute to power saving. This is true for all values of balance factors in all applications. The third important observation we make is that both the time interval based and the threshold interval based scheduling mechanisms provide higher power savings than threshold based scheduling mechanism. This indicates that there is power cost associated

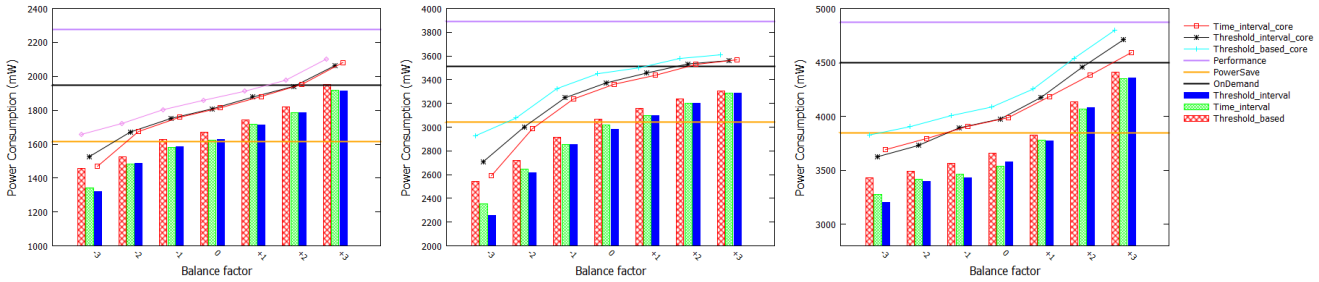

Figure 7: Power consumption in Facebook (left), Candy Crush (middle) and Google Maps (right). Horizontal lines: Performance governor (upper), OnDemand governor (middle), Powersave governor (lower); Lines joined by dots: Schedulers (Optimal # cores only); Vertical bars: Schedulers (Optimal # cores and optimal frequencies).

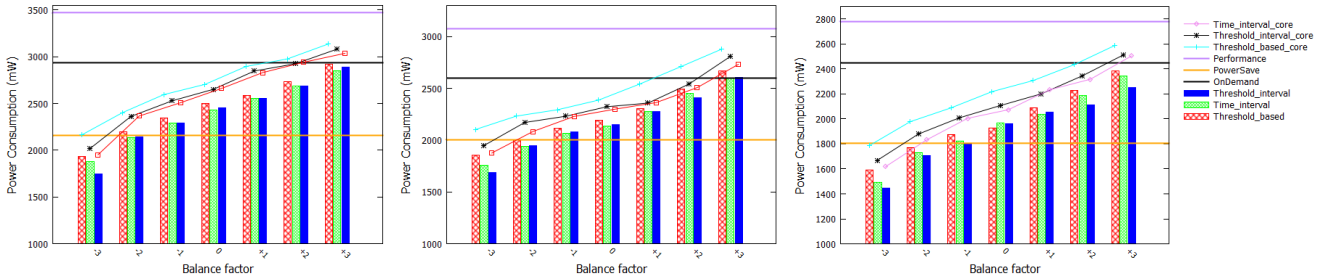

Figure 8: Power consumption in YouTube (left), Pandora (middle) and MPEG Converter (right). Horizontal lines: Performance governor (upper), OnDemand governor (middle), Powe rsave governor (lower); Lines joined by dots: Schedulers (Optimal # cores only); Vertical bars: Schedulers (Optimal # cores and optimal frequencies).

Application	OnDemand governor mW	Threshold based %	Time Interval %	Threshold Interval %
Facebook	1950	0.1 - 25.2	1.7 - 31.1	1.9 - 32.3
Pandora	2600	-2.7 - 28.7	0 - 32.4	-0.2 - 35
Facebook Pandora	2750	2.5 - 25.3	4.5 - 28.8	3.0 - 31.1
Google Maps	4500	2 - 33.7	3.3 - 37.1	3.1 - 38.8
Candy Crush	3512	5.8 - 27.6	0.4 - 33	6.4- 35.7
Youtube	2936	0.4 - 34	2.9 - 35.9	1.5 - 40.5
MPEG Convertor	2450	2.7 - 35	4.2 - 39	8 - 39

Table 8: Percentage range of power consumption savings from the three scheduling algorithms over default Android scheduler for seven applications

with bringing in new cores online or taking cores offline. For most cases, the threshold interval based scheduling mechanism consumes slightly lower power than the time interval based scheduling mechanism. However, this difference is quite small, and in fact the time interval based scheduling mechanism does consume less power in a few cases. So, based on our current experiments, there isn't suffcient evidence to definitely choose one mechanism over the other.

Finally, power savings is different for different applications. For example, there is greater power saving in case of Google Maps than Candy Crush. This is related to the variance in the CPU load as an application runs. When the CPU load changes frequently as an application is running, the new scheduling mechanisms ensure that the number of online cores is always optimal and the frequency of each online CPU remains optimal, resulting in increased power savings. When the CPU load is steady, switching between different number of cores or frequencies is not needed as frequently. Thus the power saving is relatively low for such applications. It is interesting to note that even for applications such as Candy Crush where the CPU load is mostly steady, the new scheduling mechanisms save significant power 27.6%. This indicates that the default Android scheduler does a poor job in allocating the optimal number of cores even for applications that have relatively steady CPU load.

7.4 Balance Factor

We now consider the issue of balance factor. The key question is how do we determine a reasonable value of the balance factor for a given application being used under a specific context. Among the seven applications that we have experimented with, performance is likely the most important goal for Mpeg converter, while user experience is the most important goal for Facebook, Candy crush, Google Maps, YouTube, Pandora and Facebook+Pandora. So, to determine the right balance factor for these applications, we need to look at the performance exhibited by Mpeg under the seven different balance factor values, and user experience for the other applications under the seven different balance factor values.

Figure 9 shows the performance of Mpeg converter for different balance factors under the three scheduling algorithms. The test converts one 4-min, 4.64 MB avi format file to mp4 format. For comparison purposes, we also measured the performance of this application using the Performance governor (horizontal line in Figure 9). The three vertical bars for each

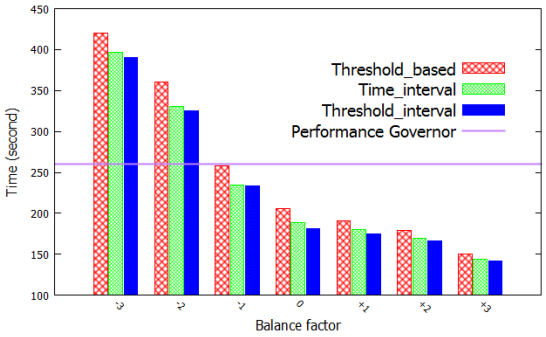

Figure 9: Performance of Mpeg converter for different balance factors under the three scheduling alogorithms

balance value in this figure show the power consumption under the three scheduling mechanisms. We can see from this figure that all three scheduling mechanisms with balance factor +3 provide better performance than the performance governor. The threshold interval based scheduling mechanism provides the best performance in this case. Furthermore, from Figure 8, we can see that the power consumption under all three scheduling mechanisms is much lower than the power consumption under performance governor for balance factor of +3.

To understand the impact of our scheduling mechanisms on user experience, we conducted a user study among 35 smartphone users. Each user installed our middleware and ran seven applications under four different scheduling mechanisms (default OnDemand governor, Threshold based scheduler, Time Interval based scheduler, and Threshold Interval based scheduler). For our three schedulers, users ran each application for seven different values of balance factor. Thus, each user ran a total of 156 apps. After running each application, each user filled out a survey form (See Table 9 for a sample survey form). For Facebook, we mainly focus on the response time, for YouTube and Pandora, we mainly focus on the loading time, and for Google Maps and Candy Crush, we focus on both loading time and response time.

Scheduler		Balance factor	
Apps	Loading Time	Response	Start
Facebook	□in one second □1 to 3 seconds □> 3 seconds	□immediately □noticeable delay □stuck	□cold □warm □hot
Pandora	□in one second □1 to 3 seconds □> 3 seconds	□immediately □noticeable delay □stuck	□cold □warm □hot
...			
Maps	□in one second □1 to 3 seconds □> 3 seconds	□immediately □noticeable delay □stuck	□cold □warm □hot

Table 9: Survey form for each scheduler and each balance factor for different applications

The entire survey results are too large to fit in the paper. Here we report a summary of our findings. The survey results showed that even though we can save significant power with balance factor −3, the performance in most cases was unacceptable. The loading time for YouTube and Pandora was much longer than 3 seconds, and there was a noticeable delay for Facebook. The Google Maps was totally stuck. However, for the balance factor −2, the loading times were less than one second for YouTube and Pandora, and there was no delay for Facebook. But there wass still a noticeable dalay for Google Maps and Candy Crush. For the balance factor −1, there was no delay in any application.

Balance factor	Ondemand governor (mW)	Threshold based (%)	Time Interval (%)	Threshold Interval (%)
Facebook	1950	0.1 - 20.7	1.7 -23	1.9 - 22.7
Pandora	2600	-2.7 - 22.2	0 - 23.3	-0.2 - 23.1
Facebook Pandora	2750	2.5 - 21.1	4.5 - 23.8	3.0 - 23.1
Google Maps	4500	2 - 23.3	3.3 - 25.1	3.1 - 24.5
Candy Crush	3512	5.8 - 18.6	0.4 - 20.1	6.4- 19.7
Youtube	2936	0.4 - 22.5	2.9 - 23.9	1.5 - 23.5
MPEG Convertor	2450	2.7 - 21.4	4.2 - 24.0	8 - 23.4

Table 10: Percentage range of power consumption savings from the three scheduling algorithms over default Android scheduler for seven applications with acceptable performance or good use experience

Based on the survey results, we have identified values of balance factor for each application under which the performance or user experience is acceptable. Table 10 provides the overall power savings over the OnDemand governor for these balance factor values. We can see from the upper end of the percentage ranges of savings in this table that the three scheduling algorithms can save as much as 18 - 25% power, while ensuring that the performance or the user experience remains acceptable.

We now consider the impact of current context on determining a right value of balance factor for an application. We consider two contextual features, remaining battery power and the expected time to charge the phone next. We consider three possible values of remaining batter power: less than 30%, between 30% and 60%, and greater than 60%. Similarly, we consider two possible values of the expected time to charge the phone next, “soon” and “not so soon”. Here “soon” implies that the charging facilities are immediately available, e.g. the user is in or close to his/her home or office. “Not so soon” implies that the user may not be able to charge his/her phone for at least another hour.

Table 11 provides suggestions for the appropriate values of balance factor for different types of applications under different contexts. These suggestions are based on the results reported in Table 10 and how critical the available power situation is in the smartphones. If the user can charge his/her phone soon, the remaining battery power has no impact on balance factor value. Otherwise, the choice of balance factor progressively moves towards increased power savings (towards −3) as the remaining battery power is reduced.

Application Type	Next Charge / Available Battery			
	Soon / NA	Not soon / < 30%	Not soon / 30-60%	Not soon / >60%
Video streaming	+1	-1	0	+1
Audio streaming	0	-1	-1	0
Games	+1	-1	0	+1
Navigation	+2	-2	0	+2
Web browsing	-1	-2	-1	-1
High Performance	0	-2	-1	0

Table 11: Suggested balance factors for different types of applications under different contexts

8. CONCLUSION AND FUTURE WORK

Power management in smartphones is a key issue that requires balancing three important goals, reducing power consumption, maximizing application performance and providing good user experience. With mobile device transition to multicore architecture, this paper addresses the issue of power management in multicore smartphones via a middleware layer that dynamically schedules optimal number of cores running at optimal frequencies to realize a good tradeoff between power consumption, performance and user experience. A prototype evaluation on Android HTC One smartphone shows that the proposed scheduling mechanisms result in significant power savings over the default Android scheduling mechanisms while ensuring good performance and user experience. These power savings hold across a range of popular smartphone applications.

Our first future direction is related to balance factor. In this paper, our choice of balance factor ranging from -3 to $+3$ and the meaning of each balance factor value are arbitrary. There are several issues that need to be considered here. How does a shorter or longer range of balance factor impact finding an appropriate balance factor value for a given application running under a particular context? What impact does it have in managing power consumption? Another issue related to balance factor is that its value is related the application. This poses a problem when multiple applications with different requirements are running at the same time, e.g. Mpeg converter where performance is an important goal and Pandora where user experience is an important goal. What should be the right balance factor in this case? We plan to conduct additional experiments with different ranges of balance factor and also conduct additional user studies to address these issues. Finally, our current schedulers operate in two stages. An interesting question is can these two stages be combined so that a decision about the optimal number of cores and optimal frequency for each core is taken in one step based on the CPU load and balanced factor, and will this result in additional power savings? We plan to explore this approach in our future work.

9. REFERENCES

[1] The Benefits of Multiple CPU Cores in Mobile Devices. NVIDIA Whitepaper.

[2] C.H. (Kees) van Berkel. Multi-Core for Mobile Phones. DATE '09 Proceedings of the Conference on Design, Automation and Test in Europe, Pages 1260-1265

[3] A. Carrol and G. Heiser. An analysis of power consumption in a smartphone. USENIX ATC, 2010.

[4] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. Dick, Z. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphone. CODES+ISSS, 2010.

[5] C. Yoon, D. Kim, W. Jung, C. Kang, H. Cha. AppScope: Application energy metering framework for Android smartphones using kernel activity monitoring. USENIX, 2012.

[6] J. Flinn and M. Satyanarayanan. Powerscope: A toolfor profiling the energy usage of mobile application. WMCSA, 1999.

[7] M. Dong and L. Zhong. Self-constructive high-rate system energy modeling for battery-powered mobile system. MobiSys, 2011.

[8] A. Pathak, Y. Hu, M. Zhang, P. Bahl, and Y. Wang. Fine-grained power modeling for smartphones using system call tracing. EuroSys, 2011.

[9] Monsoon power monitor. http://www.msoon.com/LabEquipment /PowerMonitor/.

[10] CPU Load Calculation. http://www.linuxhowtos.org/System/procstat.html

[11] W. Jung, C. Kang, C. Yoon, D. Kim, and H. Cha. Non-instrusive and online power analysis for smartphone hardware components. Technial Report. MOBED-TR-2012-1, Yonser University, 2012.

[12] Robert Basmadjian, Hermann de Meer. Evaluating and Modeling Power Consumption of Multi-Core Processors. ACM, 2012.

[13] Thomas Hubbard, Rainmondas Lencevicius, Edu Metz, Gopal Raghavan. Performance Validation on multicore Mobile Devices. Verified Software: Theories, Tools, Experiments, Pages 413 - 421

[14] Sangwook Kim, Hwanju Kim, Jongwon Kim, Joonwon Lee, Euiseong Seo. Empirical Analysis of Power Management Schemes for Multi-core Smartphones. ICUIMC '13 Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication, January 2013

[15] Marius Marcu, Dacian Tudor, Sebastian Fuicu, Silvia Copil-Crisan, Florin Maticu, Mihai Micea Power Efficiency Study of Multi-threading Applications for Multi-core Mobile Systems. WSEAS TRANSCATIONS on COMPUTERS, Issue 12, Volume 7, December 2008

[16] Minyong Kim ; Joonho Kong ; Sung Woo Chung An online power estimation technique for multi-core smartphones with advanced display components Consumer Electronics (ICCE), 2012 IEEE International Conference, January 2012

[17] T. Mudge Power: A first class architecture design constraint. IEEE Computers, 34(4):5258, April, 2001

[18] Weiser, Mark and Welch, Brent and Demers, Alan and Shenker, Scott Scheduling for Reduced CPU Energy Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation, 1994

[19] Z. Mwaikambo, A. Raj, R. Russel, and J. Schopp Linux kernel hotplug CPU support. In In Proceedings of the Ottawa Linux Symposium, 2004.

[20] Liang, Y., P. Lai, and C. Chiou An energy conservation DVFS algorithm for the android operating system. IJournal of Convergence 1.1, 2010.