Batch Scheduling of Recurrent Applications for Energy Savings on Mobile Phones

Matt Calder and Mahesh K. Marina
School of Informatics
The University of Edinburgh, UK

***Abstract*—Recurrent applications that mostly run in the background are a significant source of power consumption on battery-limited mobile phones. We highlight the pitfalls of scheduling such applications independently without awareness of each other's schedules. We illustrate the significant energy savings that can be achieved via *batch scheduling* of recurrent mobile phone applications. We then present our on-going work on developing a general batch scheduling framework for such applications and also outline our early experiences studying the benefit of batch scheduling on two different mobile phone platforms — Nokia N95 and HTC (Android) — that are commonly used in the research community.**

I. Introduction

With more than 3 billion mobile phones in use today, they have become indispensable companions for personal and business communication. The emergence of smartphones is rapidly increasing the sophistication of mobile phones (in terms of computing, communication and sensing capabilities), but battery capacity, growing at a much slower pace, is fast becoming the bottleneck [1].

A class of mobile phone applications that we refer to as *recurrent applications* can be a significant source of power consumption [2]. This class of applications are primarily characterized by repeated execution, mostly as a background process, to perform a task periodically. A secondary characteristic is that these applications are also delay tolerant. This means that while ideally they would like to be run exactly at every specified interval, they are content to run every *interval-ish*. For phones with deep sleep modes, there is also a cost in terms of time for the phone to wake up and go to sleep. When these recurrent applications are left to schedule themselves independently of other recurrent applications, they can have a very negative impact on battery life by bringing the phone out of sleep mode an unnecessary number of times. For general consumer use, there are several examples such as email, RSS, news readers, podcast, Twitter, and Facebook. Recurrent applications are also used frequently in mobile sensing domains where it is necessary to take context-sensitive sensor measurements such as location, ambient light, sound and accelerometer readings [3], [4].

Our research goals are as follows:

1) Quantify the energy costs associated with independent recurrent application scheduling and demonstrate that there can be significant energy savings by maximizing sleep time via *batch scheduling* of recurrent applications.
2) Develop a general batch scheduling framework for recurrent applications and experimentally study the benefits on real mobile platforms (e.g., Android phones).
   a) Minimize the number of times the phone is woken-up by recurrent applications.
   b) Minimizing up-time power usage by scheduling applications with complementary resource usage together.

In this paper, we will first discuss related work (§II) done in the area of application level optimizations for mobile phones and then motivate the need for batch scheduling of recurrent mobile applications (§III). The key idea is to maximize sleep time from a holistic view of repeating intervals and resource usage of all active recurrent applications and thereby achieve energy savings. In §IV, we present our initial work towards developing a general batch scheduling framework. We will conclude with an overview of our future plans for extending the framework.

II. Related Work

There has been much recent work done in the area of application-level optimizations for mobile phones by exploiting network radio behavior, and using context and location aware sensing. TailEnder [5] and Cool-Tether [6] describe strategies for exploiting residual energy "tail" from GPRS/EDGE and 3G radio communication. TailEnder describes an algorithm for energy saving by batch scheduling network requests *internal* to a single delay tolerant application and also by doing bursty prefetching for web browsing. Xiao et al. [7] also explore the power characteristics of 3G and WiFi but focus on mobile YouTube viewing. Wu et al. [8] describe an optimization to reduce unnecessary WiFi scanning by detecting motion using the changes in signal strength from nearby cellular towers.

The Google Android mobile phone platform from version 1.5 has introduced the notion of *inexact repeating* which includes APIs for developers. Those APIs can be used for batch scheduling applications as long as they have a interval (or some multiple) of 15 minutes. In our work, we seek to quantify the energy characteristics of batch scheduling *in general* and also to establish a framework for optimized scheduling that is not limited to a specific set of recurrence intervals. Imposing a restriction on specific intervals is unsuitable for many important types of recurrent applications

978-1-4244-7151-5/10/$26.00 ©2010 Crown

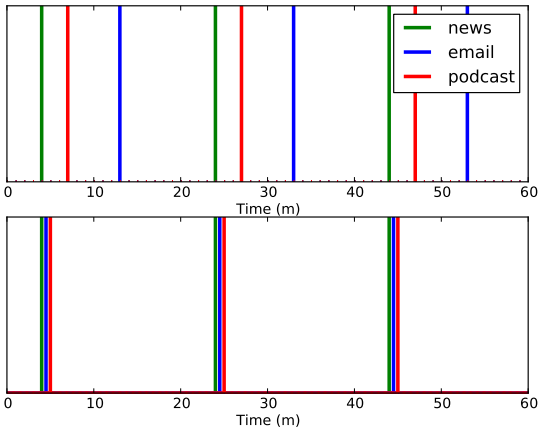

Fig. 1: 60 minute view for the execution schedules of three long-running networking applications. The top figure is the result of an independent schedule while the bottom shows the applications scheduled in a batched fashion.

such as mobile sensing which may have sampling requirements specific to a particular domain.

In [9], the authors describe an energy-optimization framework using Markov Decision Processes to maximize the talk time on a mobile phone with respect to a known next charge time. The technique used is to dynamically change the frequency of email synchronization based on available energy to extend battery life until the next charge time. In contrast, our work focuses on the case of *multiple* recurrent applications.

III. Motivation

We now consider a realistic case to highlight inefficient energy usage resulting from independent scheduling of recurrent mobile applications and to motivate the need for batch scheduling. In particular, we consider three common and recurrent network applications on mobile phones: news feed, podcast and e-mail. The energy usage of network radios has been well investigated recently, so considering network applications allows us to use data from existing studies. For simplicity, we assume that all three applications have the same recurrence interval of 20 minutes and they all download 50KB data whenever they execute. In our example, we arbitrarily set the start times of the three applications — news feed application starts on the 4th minute of the hour, e-mail on the 13th minute and the podcast application on the 7th minute of the hour. Finally, we assume the use of Nokia N95 platform for which network energy consumption results are available in the literature [5].

With independent scheduling, each of these applications are scheduled to repeat based on their time of launch. With this schedule, *Fig. 1 top* shows that the applications wake up the phone 9 separate times in a one hour period. The result of a batch scheduling system, seen in *Fig. 1 bottom*, shows the applications still executing at their 20 minute intervals but their schedules have been grouped together for execution so that phone is only woken up 3 times.

An interesting property of 3G radios is that there are some large constant energy costs to do any sized transmission [5]. The initial start up cost of the radio is called the *ramp* and there is a longer residual energy plateau between 10 to 12 seconds called the *tail*. Intuitively, it is much better to perform several bursty transfers at once with a single ramp and tail cost than to have several ramp-transfer-tail cycles. This is an example of complementary resource usage. For measuring the benefits of this, we have the following set of variables and equations:

C_R is the energy cost of the ramp.
C_T is the energy cost of the tail.
C_{50k} is the energy cost of the 50k transfer.

The total energy cost of n independent 3G transfers is

$$n(C_R + C_{50k} + C_T) \tag{1}$$

In the case of batched applications, the cost of a applications in b batches is

$$b(C_R + aC_{50k} + C_T) \tag{2}$$

In our example we have 9 independent network transfers so we have $9C_R + 9C_{50k} + 9C_T$ and the batched schedule has 3 batches with 3 applications, so this gives $3C_R + 9C_{50k} + 3C_T$. The former is clearly greater than the latter. We can get a better idea of potential energy savings by plugging some actual values to various cost variables. From [5], we have $C_R = 2.5J$, $C_{50k} = 3J$, and $C_T = 7.5J$. Substituting these values in equations (1) and (2) gives us $117J$ and $57J$, respectively. Thus, in this example, batch scheduling gives a 51.3% reduction in energy consumption for the same number of application executions in a one hour period.

IV. Batch Scheduling of Recurrent Mobile Applications

In this section we present our initial algorithm for working with the batch scheduling problem. Our primary goal is to maximize sleep time of the phone by overlapping the execution of recurrent applications as much as possible so that the number of times the phone is woken is minimized. We place two restrictions: (1) execution duration of an application is ignored and (2) the schedule of an application once created is not altered thereafter. We make another assumption that batch scheduling applications together is always beneficial.

The batch scheduling algorithm works in the following way. Assume there is a new application a with an interval ρ. The first step of the algorithm is to find the interval of an already scheduled application to synchronize with. It first checks if ρ is an already scheduled interval. If not, it checks if the `gcd` of ρ and any of the existing intervals are greater than 1 and chooses the lowest matching value. If an interval has still not been found then the algorithm picks the smallest existing interval. Let the application corresponding to the interval obtained by the above algorithm be b.

The second step of the algorithm is to calculate the initial execution time for a so that it coincides with the next execution time of b. If ρ is greater than the interval of b or if $now + \rho$ time units is greater than the next execution time of b then the initial execution of a should be the next execution of b.

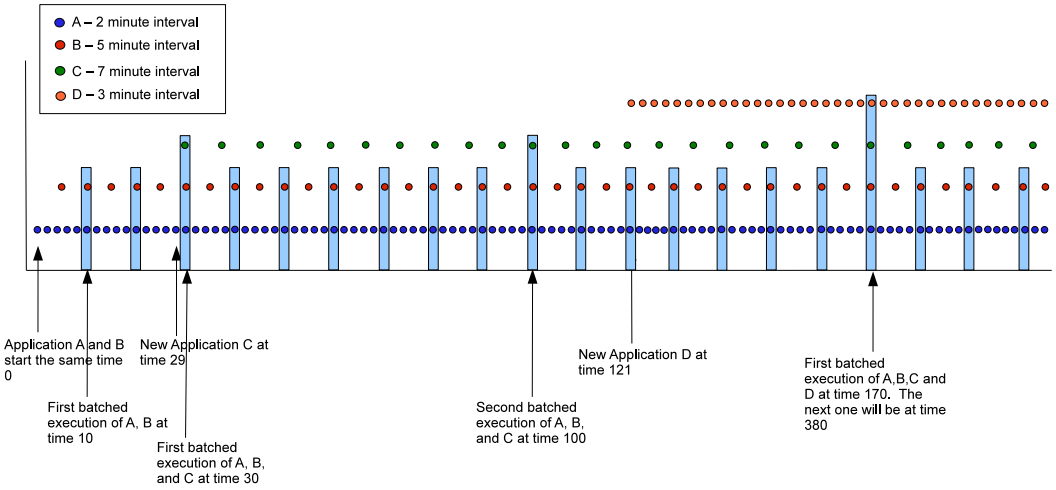

Fig. 2: Time line progress of the batch scheduling algorithm. The highlighted blue bars indicate the times where certain combinations of applications have been batched together although many are not shown. For instance, the total number of overlaps for the combinations of two applications is 62 and there are 9 overlaps for the combinations of three applications.

Otherwise, set the initial execution time of a to be an offset from now such that an eventual execution of a will coincide with b's next execution.

It is important to note that our batch scheduling work attempts to exploit application intervals with similar properties; namely by finding a greatest common divisor between new and existing applications. For applications with dissimilar (coprime) intervals, the energy savings will be relatively smaller.

In *Fig. 2*, we show a scenario to illustrate what happens when applications are added using the batch scheduling algorithm. In contrast to *Fig. 1*, this scenario is a worst case for batch scheduling because all the application intervals are coprime. The schedule starts off with two applications A and B starting at the same time with intervals of 2 and 5 minutes respectively. With these two applications, the entire schedule period is 10 minutes. We can see that application C with interval 7 tries to start at time 29. Checking the existing intervals we see that interval 7 does not exist and neither do any multiples of 7. Since we see that the next execution of the smallest interval, A, is at time 30, C is first scheduled to run with A and B. Application D requests to be scheduled at time 121. There are no matching intervals so the scheduling algorithm calculates the offset to start application D at 122 and it becomes aligned to be batch scheduled with A, B, and C at time 170.

While in this work we have discussed an initial algorithm for performing batch scheduling, in a longer version of this work [10] we propose a more general scheduling framework that uses threshold values to allow even greater opportunity for batch scheduling. The algorithm described here is a special case when threshold values are set to 0.

V. Discussion and Conclusion

We have discussed in this work why batch scheduling is an important energy saving technique to use with recurrent applications. We have shown that batch scheduling can offer significant energy savings and demonstrated how this can be achieved with our scheduling framework. While our initial scheduling work has a few restrictions, the focus of our current work is on relaxing these restrictions and making the model more general. We are planning to investigate the potential benefits of dynamically adjusting an existing schedule to allow greater batch scheduling opportunity for a new application. We will integrate application runtime duration into our scheduling model and establish how knowledge of complimentary resource usage among applications can further improve batch scheduling. Details of our experimental investigations with different mobile phone platforms is available in the longer tech report [10].

References

[1] P. Ranganathan. Recipe for Efficiency: Principles of Power-Aware Computing. *Communications of the ACM*, 53(4), Apr 2010.

[2] J. Sharkey. Coding For Battery Life. http://code.google.com/events/io/2009/sessions/CodingLifeBatteryLife.html, 2009.

[3] M. Musolesi, M. Piraccini, K. Fodor, A. Corradi, and A. T. Campbell. Supporting Energy-Efficient Uploading Strategies for Contiguous Sensing Applications on Mobile Phones. In *Proc. IEEE PerCom*, 2010.

[4] J. Kukkonen, E. Lagerspetz, P. Nurmi, and M. Andersson. BeTelGeuse: A Platform for Gathering and Processing Situational Data. *IEEE Pervasive Computing*, 8(2), 2009.

[5] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. In *Proc. Internet Measurement Conference (IMC)*, 2009.

[6] A. Sharma, V. Navda, R. Ramjee, V. N. Padmanabhan, and E. M. Belding. Cool-Tether: Energy Efficient On-the-fly WiFi Hot-spots using Mobile Phones. In *Proc. ACM CoNext*, 2009.

[7] Y. Xiao, R. Sri Kalyanaraman, and A. Yla-Jaaski. Energy Consumption of Mobile YouTube: Quantitative Measurement and Analysis. In *Proc. NGMAST*, 2008.

[8] H. Wu, K. Tan, J. Liu, and Y. Zhang. Footprint: Cellular Assisted Wi-Fi AP Discovery on Mobile Phones for Energy Savings. In *Proc. ACM WiNTECH*, 2009.

[9] T. L. Cheung, K. Okamoto, F. Miller, X. Liu, and V. Akella. Markov Decision Process (MDP) Framework for Optimizing Software on Mobile Phones. In *Proc. EMSOFT*, 2009.

[10] M. Calder and M. K. Marina. Batch Scheduling of Recurrent Applications for Energy Savings on Mobile Phones Technical Report. The University of Edinburgh, Informatics Research Report, June 2010. http://homepages.inf.ed.ac.uk/s0933443/papers/BatchScheduling2010TR.pdf.