SmartSync - Profile Driven, Power Aware Synchronization Manager

Frank Maker
Department of Electrical and Computer Engineering
University of California, Davis
flmaker@ece.ucdavis.edu

Yichuan Wang
Department of Computer Science
University of California, Davis
yicwang@cs.ucdavis.edu

Eric Jung
Department of Computer Science
University of California, Davis
eajung@cs.ucdavis.edu

Xin Liu
Department of Computer Science
University of California, Davis
liu@cs.ucdavis.edu

Venkatesh Akella
Department of Electrical and Computer Engineering
University of California, Davis
akella@ece.ucdavis.edu

ABSTRACT

Aggressive power optimization is critical in Android devices as their new functionalities continually increase. The existing synchronization manager in Android does not distinguish between users or applications. We propose SmartSync, which explicitly takes user profiles and current battery status into account. It customizes sync scheduling based on the future predicted battery recharge time, T. SmartSync formulates synchronization as an optimization problem which can be solved efficiently using a Markov decision process (MDP) model. In this proposal, we describe the system architecture and the APIs for SmartSync.

1. INTRODUCTION

Android-based smartphones are compelling because they provide sophisticated functionalities and with each new generation they are expected to do even more. For example, next generation devices have multicore processing, near field communication (NFC), video conferencing and increased cellular data rates (4G). However, battery technology is not improving at the same pace; New and improved functionality continually outpaces power reduction gains. So it is important to manage battery resources more aggressively than ever. Hardware oriented power management schemes such as frequency scaling, voltage domains and power down modes are already widely in use. But, as hardware becomes more and more efficient, what is next? What else can be done to support new functionality with available battery energy?

We argue that smartphones are different and somewhat unique in the way they are used compared to general purpose computers. A smartphone is a personal device and is used by one user almost exclusively. But, people have dramatically different usage patterns. For example, a business user might want email to be always be synchronized, whereas a teenage user could be more concerned with social media. Different users have different priorities and different latency tolerances for particular applications. Currently, Android does not take the *user profile* into account for energy optimization. For every user, all applications are treated equally as if one-size-fits-all.

Instead, we believe that the next frontier in software optimization is to exploit usage profiles. The system should maximize the user experience based on synchronization delay tolerances and user-defined application priorities. This distinction should be observed inside the Android framework. It should also be exposed in the APIs, so that third-party applications can take advantage of this feature.

We focus on data synchronization because it is quickly becoming a central operation for many applications, which communicate with cloud-based services. As of version 2.2, Froyo, there is now a unified synchronization framework in Android. Each application can register synchronization tasks with the Sync Manager. Then the manager can coordinate syncs so that each task is synchronized at the same time. This amortizes the energy overhead of waking up the communication radio over each task. Without coordination this cost could be paid multiples times with no extra benefit to the user.

We propose a new synchronization framework called SmartSync, that maximizes the frequency of data synchronization, while preserving the ability to make voice calls until a predefined time (i.e. when the phone can be recharged). This is accomplished by statisticallly analyzing past user history and creating a decision table based on the remaining energy and last synchronization time. The table specifies the dynamic sync interval based on the user profile, current time and battery status.

We start with the theoretical underpinning of SmartSync - a Markov Decision Process based technique and then formulate the optimization problem. Next, we describe details of the implementation in the Android framework. Then, we provide the SmartSync API and how third-party developers can use it. Lastly, we conclude with a project schedule and team member biographies.

2. PROBLEM FORMULATION

SmartSync is based on our previous power optimization work using a Markov decision process[1, 2]. A Markov decision process (MDP) is a widely used mathematical technique for

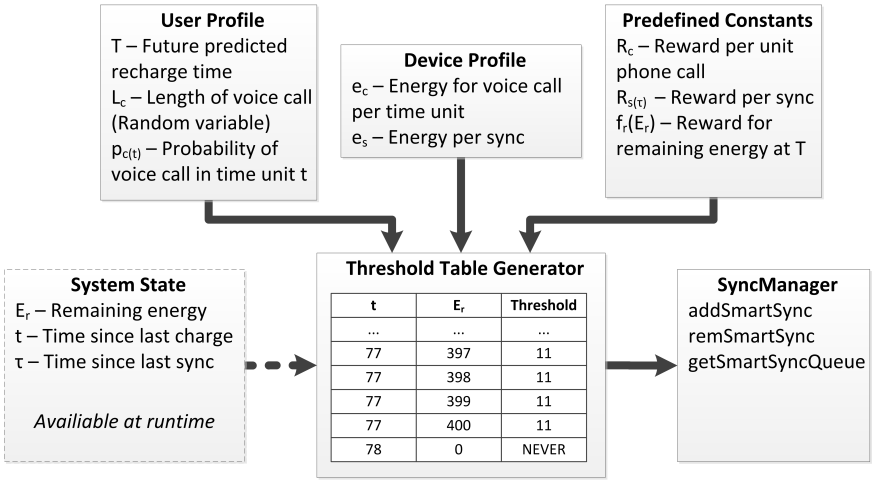

Figure 1: System Architecture

modeling decision making[3]. It is applicable when results are partly random and partly under control. A MDP is a discrete time stochastic control process, formally presented by a 4-tuple:

- S - state space, where $s \in S$ is the current state
- A - action space, where $a \in A$ is an action taken based on state s
- $P_a(s, s')$ - probability of action a in state s at time t will lead to state s' at time $t+1$
- $R_a(s, s')$ - immediate reward of action a after transition from state s to state s' with transition probability $P_a(s, s')$

We use the following values for our MDP model:

- s is a 3-tuple consisting of:
  - time since last sync, t
  - time since last charge, τ
  - remaining energy, E_r
- a is the decision whether to sync or not
- $P_a(s, s')$ is the probability of a sync decision in the current state s, at the current time slot, t, resulting in state, s', at the next timeslot, $t+1$.
- $R_a(s, s')$ is the immediate reward for an action, syncing or unit of phone call, after transitioning from s to s'.
- $f_r(E_r)$ is the reward function for remaining energy, E_r

This formulation describes a true MDP because user energy consumption and phone call arrivals are random[1] and synchronization is under control. The ratio between the reward of a sync and a unit of phone call time reflects the relative priority of each task. As the reward of one rises compared to other, MDP favors former and vice verse. Our overall goal is to maximize utility for the user. Therefore, our objective is to maximize total reward until the predicted battery charge time. To accomplish this we must determine the optimal action for each state, s, at each time, t. This decision depends on the current state, immediate reward and future reward. For example, synchronization now yields an immediate reward, at the cost of energy consumption, which may reduce future rewards. But, continuing to delay synchronization reduces the current reward as well. Consequently, each of these factors must be considered jointly in the decision.

However, state transitions are partially random. The action taken, a, and time since last sync, τ are both predetermined. Whereas phone calls received and energy consumption are random. With the help of the user's call logs, we take the expectation over the possible phone call arrivals in the future. Thus, we can decide which action is the most likely to yield the maximum cumulative reward.

We use this MDP construction to generate an *optimal policy*, which dictates which action should taken based on the current state. This decision table typically has the same dimensions as the state space. In practice, the main challenge of MDP modeling is to manage its computational complexity; It grows grows exponentially with the number of states in the general case. A high dimension decision table also requires a significant amount of storage.

In previous work [4], we proved that the optimal policy for this application is threshold based. Instead of having a high dimension table (time, remaining energy, time since last sync, etc.), it can be reduced to a single dimension: energy threshold. For each state and each time slot we have exactly *one* threshold value. If the current remaining energy is larger than this value, the optimal action is to sync, otherwise to remain idle. By simplifying the variables required at run-time, we greatly reduce the complexity and storage required.

3. SYSTEM DESIGN

Our SmartSync architecture (Figure 1) is centered around the MDP which generates the sync threshold table. All the

[1]Actually energy usage is pseudorandom because it can be predicted with a sophisticated model. However, real-time prediction on an Android device, is unfeasible and therefore consumption values are random

components combined provide the variables required for the MDP formulation. Next, we discuss each component's role in the system. In Section 4, we describe the implementation of this design inside the Android synchronization manager framework.

3.1 System State

System state contains all the factors which determine element, s, of the state space, S, at run-time. During each time slot these values are read. Once known, the state is used to index the threshold table. Then, the threshold is compared to the time since synchronization, τ. If the threshold is less than or equal to τ then a synchronization is performed.

3.2 Predefined Constants

Before we can optimize sync behavior, each reward value must first be defined. These values determine the cost function which is then optimized by the threshold table generator. For each operation in our user model, phone calls and syncs, we have a corresponding reward, R_c and $R_{s(\tau)}$ respectively. By adjusting these values we can modify the relative importance of each operation. The goal of our system is to maximize reward. Therefore, the MDP will deplete all battery energy with these rewards alone. This behavior is impractical since the future charge time, T, is in fact only an estimate. We introduce a reward for remaining energy at time T to overcome this limitation. This factor allows for a reservoir of battery energy to be reserved.

3.3 User Profile

Our primary goal is to dynamically adjust synchronization based on energy usage. To meet this goal, the system must track user actions impacting remaining battery energy. We require the charge history of the device. It is used to record the time since the last charge, t. We can also predict the future recharge time, T, using statistical analysis. In addition to charging information, runtime user actions also impact stored energy. In our user model these actions take two forms: synchronization and voice calls. For synchronization tasks only the time since the last task, τ, is necessary. Whereas phone calls are stochastic in nature; They require both the probability of a voice call arrival, $p_{c(t)}$, and a random variable characterizing the expected call length, L_c. These statistics can be derived from previous call logs stored on the phone itself.

3.4 Device Profile

In addition to user profiling, device specific behavior also impacts our energy budget. The remaining battery energy itself, E_r, is of primary importance. At runtime this value is used to select the correct decision in our threshold table. Each device will also have its own power consumption characteristics. This results in identical user decisions on heterogeneous platforms having different energy costs. We need these values for both user driven actions: voice calls (e_c) and synchronization tasks (e_s).

3.5 Threshold Table Generator

With the all the aforementioned components the threshold table can be generated prior to run-time. With the system state at runtime, the threshold table indicates, for each time slot, what is the optimal action to take. The threshold table is generated every time the phone is charged. During charging time, energy consumption is effectively zero. This allows our threshold table to "reset" for the next battery discharge cycle. Lastly, the user profile is updated using recent and previous phone call logs.

3.6 SyncManager

Each aspect of the exposed SmartSync interface is described in the API component. *addSmartSync* adds a new SmartSync job to the sync queue. The minimum sync frequency is given as a parameter. Although, this frequency is not guaranteed. SmartSync uses a best effort approach to meet all the requested sync frequencies in the queue. *remSmartSync* removes a job from the queue, effectively stopping it. *getSmartSyncQueue* returns the entire queue for inspection, bulk operations, etc.

4. IMPLEMENTATION

Next, we discuss the details of our SmartSync implementation on Android. The goal of this section is demonstrate the feasibility of our proposal. In order to clarify some of these details we identify specific paths in the Android Open Source Project tree (AOSP)[5]. Each path is relative to the root project folder, called $AOSP.

4.1 Predefined Constants

Each of these constants are defined by the developer beforehand. In our previous work we performed simulations using the following rewards: $R_c = 3$, $R_{s(\tau)} = \sqrt{\tau + 1}$ and $f_r(E_r) = 0$. However, implementation on an actual phone will definitely require a different value for $f_r(E_r)$. $f_r(E_r)$ must be greater than zero because it is too aggressive and will leave no margin for error. The other values will also require tuning as we experiment with different values empirically. Therefore, our project will expose them as a XML file to allow for easy modification.

4.2 Device Profile

In Android version 2.2, the battery usage screen was introduced (Figure 2) . Each device since, contains an XML file with device-specific power constants (Listing 1). They are used to calculate coarse grain energy consumption. Although these calculations are not presented to the user, their relative ranking and value are. The file path for each device is based on the manufacturer and model name. It can be found at: `$AOSP/device/$MANUFACTURER/$MODEL/overlay/frameworks/base/core/res/res/xml/power_profile.xml`. We will use these constants to calculate the energy remaining for any device. This overcomes the challenge of profiling each device individually.

4.3 User Profile

An Android Service[2] will be used, running in the background, to maintain the user profile information. To characterize an individual user's profile it will keep a histogram of call activity. This data will allow us to predict the future recharge time using a weighted average. We can also easily calculate the probability that a phone call will be received in each time slot. We will achieve this by comparing

[2]Analogous to a background daemon server process on other platforms.

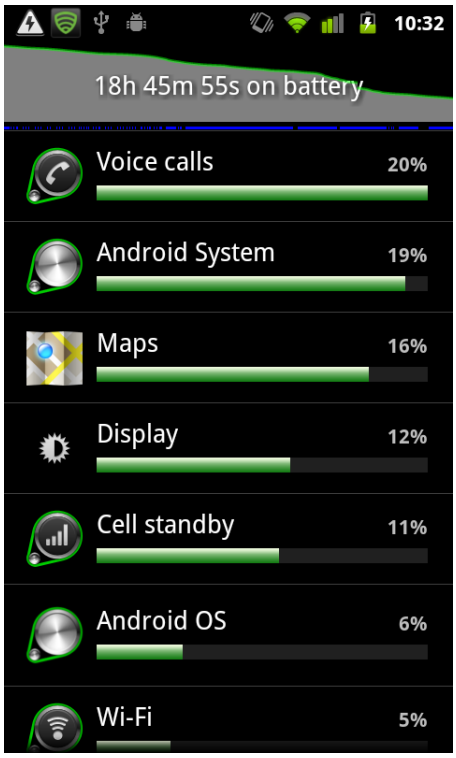

Figure 2: Android Battery Usage Screen

```
<device name="Android">
  <item name="none">0</item>
  <item name="screen.on">49</item>
  <item name="bluetooth.active">142</item>
  <item name="bluetooth.on">0.3</item>
  <item name="bluetooth.at">35690</item>
  <item name="screen.full">260</item>
  <item name="wifi.on">4</item>
  <item name="wifi.active">120</item>
  <item name="wifi.scan">220</item>
  <item name="dsp.audio">88</item>
```

Listing 1: Excerpt from power_profile.xml

the total calls in the histogram to those received during an individual time slot. The length of each call will then be saved in another histogram. By combining this data we can then calculate the expected voice call length using each slot's values.

4.4 Threshold Table Generator

Our threshold table will be exposed to the SyncManager as a Content Provider. Content Providers expose persistant data sources with a create, read, update, delete interface (CRUD). This abstraction allows us to decouple the table from the manager itself. Decoupling is important to provide future flexibility and reduce code complexity. The table will be presented as a read-only interface to the SmartSync methods. It serves as a threshold lookup table for a given time since sync, τ, and energy remaining, E_r.

The table data itself will be regenerated each time the user recharges the device. This is a convient time for a couple reasons. First, battery energy is no longer being consumed so our user profiler is unnecessary. Secondly, we prefer not to waste the user's battery life if possible. Lastly, it ensures that our table is rebuilt periodically since recharging is often done daily. This event is easily captured on Android by a system-wide broadcast Intent. Specifically this intent is identified with the string `ACTION_POWER_CONNECTED`. All that is required to receive this intent is to register our application for it and setup a broadcast listener.

4.5 System State

System state is the runtime value of since the last sync, τ, and remaining energy E_r. τ can be recorded by monitoring the SyncManager to record each sync time. E_r require two pieces of information. We need the last battery percent value sent from the `ACTION_BATTERY_CHANGED` Broadcast Intent. We also need the total battery capacity which can be read from the battery driver at `/sys/class/power_supply/battery/capacity`. Finally, our system multiplies the percentage by the total capacity to record the current energy remaining.

4.6 SmartSync

We will modify the existing Sync Manager located at `$AOSP/frameworks/base/core/java/android/content/SyncManager.java`. We will supplement the existing methods with smart variants: `addSmartSync`, `remSmartSync` and, `getSmartSyncQueue`.

5. API

We will implement our manager, SmartSync, on top of the built-in sync manager. Our solution will replicate the existing sync methods *addPeriodicSync*, *removePeriodicSync*, and *getPeriodicSync* with *addSmartSync*, *removeSmartSync*, and *getSmartSyncs* respectively. However, the fundamental difference, from the developer's perspective, is the specification of a minimum as opposed to a fixed sync frequency.

6. PROJECT SCHEDULE

An overview of our project schedule is shown in Figure 3. The most complex and important part of the design, the threshold generator, will be developed first. We already have a simulation version from our previous work[2]. This

ID	Milestones	Owner(s)	Start	Finish	Duration
1	Threshold Generator	Eric Jung, Yichuan Wang	7/15/2011	8/16/2011	4.5w
2	Device Profiler	Frank Maker	7/15/2011	8/4/2011	3w
3	User Profiler	Frank Maker, Eric Jung	8/5/2011	9/1/2011	4w
4	SmartSync Adapter	Yichuan Wang	8/17/2011	9/6/2011	3w
5	Integration	All	9/2/2011	9/20/2011	2.5w
6	Testing	All	9/21/2011	10/4/2011	2w
7	Prepare Demo	All	10/5/2011	10/10/2011	.8w

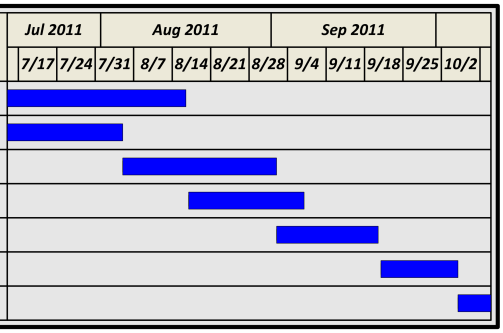

Figure 3: Project Schedule

version will serve as a complete reference. However, we need to rewrite the generator for Android and its resource constraints. Previously, we used 1 Joule increments for energy remaining. However, we do not have that resolution on the actual phone and instead are using battery percentage levels. These new factors add more development time because we have not implemented them before. Lastly, the threshold generator is at the heart of our design. Consequently, we budgeted a bit more time to tune the reward functions.

Concurrently Frank will work on both the device and user profilers. The device profiler has minimal design elements. This is because each device's profile settings are easy to acquire. The user profiler, however, will take more time because of the statistical analysis involved. Furthermore, we may also need to develop heuristics that make the use of these values acceptable. Then they must also be tested across varying user profiles. This process will take a fair amount of time to acquire and analyze.

Eric designed the MDP model and proved the optimality of the threshold simplification. Therefore, his time is best spent initially supporting the table generator itself. Afterwords, our user profile findings will impact the effectiveness of the final system. If user profiles in practice deviate from those previously simulated, Eric will make modifications to our design.

Yichuan initially will support Eric in rewriting the threshold generator. Afterwords, he will be in charge of exposing our system to the Android framework inside the SyncManager. We don't anticipate this task to be overly complicated. As we begin integration of each part there will surely be modifications required for the adapter to better tie in with our design.

All team members will help with integration and testing. Testing will continue for the last few weeks of the project. Just before the conference we will prepare our demonstration.

7. BIOGRAPHIES

Frank Maker

He is a Ph.D. candidate in Electrical and Computer Engineering at the University of California Davis. His research focuses on automatic power adaptation of heterogeneous smartphone platforms. In past two years he has interned with the Android systems group at Google, Inc. and the Android mobile multimedia systems group at Broadcom Corp. Prior to graduate school, he worked as a software engineer in the automotive industry for Motorola and BMW.

Yichuan Wang

He is a Ph.D. candidate in Computer Science at the University of California Davis. His research focuses mobile Internet, mobile system, with a focus on user behavior modeling and optimization. Last summer, he interned with Deustche Telekom Lab in Palo Alto, CA. Prior to beginning his doctorate studies, he worked in the mobile industry for Nokia Corporation.

Xin Liu

She received her Ph.D. degree in Electrical Engineering from Purdue University, and is currently an associate professor in the Computer Science Department at the University of California, Davis. Her research is on wireless communication networks, with a focus on resource management. She received the Best Paper of year award of the Computer Networks Journal, NSF CAREER award, and the Outstanding Engineering Junior Faculty Award from the College of Engineering, UC Davis.

Venkatesh Akella

He received his PhD in Computer Science from University of Utah and is a professor of Electrical & Computer Engineering at University of California in Davis. His current research encompasses various aspects of embedded systems and computer architecture with special emphasis on embedded software, hardware/software codesign and low power system design. He is member of ACM and received the NSF CAREER award.

8. CONCLUSION

In conclusion, we have introduced a new profile and power aware Android synchronization framework called SmartSync. Our framework dynamically allocates limited battery energy for synchronization based on previous user call history. This optimization problem is then formulated using a Markov Decision Process model. From our previous work, we described an optimal threshold policy which reduces MDP table dimensions. We then outlined our specific design and implementation of this solution on the Android platform including the programmer API. We discussed our project schedule

with each team member's role and project milestone dates. Lastly, we presented the biographies and relevant experience of each individual team member. In summary, we have provided a complete proposal for an improved Android synchronization manager which intelligently optimizes sync scheduling based on prior call arrival history.

9. REFERENCES

[1] E. Jung, F. Maker, T. L. Cheung, X. Liu, and V. Akella, "Markov decision process (MDP) framework for software power optimization using call profiles on mobile phones," *Design Automation for Embedded Systems*, vol. 14, pp. 131–159, June 2010.

[2] T. L. Cheung, K. Okamoto, F. M. III, X. Liu, and V. Akella, "Markov decision process (MDP) framework for optimizing software on mobile phones," in *International Conference On Embedded Software*, pp. 11–20, 2009.

[3] Wikipedia, "Markov decision process - Wikipedia, the free encyclopedia," 2011.

[4] E. Jung, Y. Wang, I. Prilepov, F. Maker, X. Liu, and V. Akella, "User-profile-driven collaborative bandwidth sharing on mobile phones," *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services Social Networks and Beyond - MCS '10*, pp. 1–9, 2010.

[5] Google, "Android Open Source Project."