Energy-Efficient and Robust Middleware Prototyping for Smart Mobile Computing

Saideep Tiku
Department of Electrical and Computer Engineering
Colorado State University, Fort Collins, CO, 80523
saideep@rams.colostate.edu

Sudeep Pasricha
Department of Electrical and Computer Engineering
Colorado State University, Fort Collins, CO, 80523
sudeep@colostate.edu

ABSTRACT

A large amount of data is produced by mobile devices today. The rising computational abilities and sophisticated operating systems (OS) on these devices have allowed us to create applications that are able to leverage this data to deliver better services. But today's mobile technology is heavily limited by low battery capacity and limited cooling capabilities, which has motivated a search for new ways to optimize for energy-efficiency. A challenge in conducting such optimizations for today's mobile devices is to be able to make changes in complex OS and application software architectures. Middleware has been becoming an increasingly popular solution for inserting energy-efficient solutions and optimizations in a robust manner, without altering the OS or application code. This is because of the flexibility and standardization that can be achieved through middleware. In this paper, we discuss some powerful and promising developments in prototyping middleware for energy-efficient and robust execution of a variety of applications on commodity mobile computing devices.

CCS CONCEPTS

• **Computing methodologies** → Machine learning • **Computer systems organization** → Embedded and cyber-physical systems • **Computer systems organization** → Dependable and fault-tolerant systems and networks • **Hardware** → Power and energy

KEYWORDS

Middleware, mobile computing, energy-efficiency, robustness

ACM Reference format:
Saideep Tiku, Sudeep Pasricha 2017. Energy-efficient and robust middleware prototyping for smart mobile computing. In Proceedings of the 28th International Symposium of Rapid System Prototyping (RSP'17). ACM, Seoul, Republic of Korea, October 2017, 7 pages.
DOI: 10.1145/3130265.3138855

1. INTRODUCTION

Over the past decade we have seen an increasing trend of people becoming dependent on mobile devices for communication and data access. A recent study [1] found that 99% of adults in the USA between 18-29 years of age own a smartphone. Another study from CISCO in 2016 [2] suggests that on an average 10.7 exabytes of mobile data traffic is offloaded each month from mobile phones to cloud datacenters (Figure 1). The study further forecasts that this value is set to increase 8-fold by 2021. This highlights an opportunity for software that is able to leverage the data being produced to deliver better services to the users of these devices.

Mobile devices typically make use of lithium-ion polymer batteries (LiPo) that have been used in portable electronics since the mid 1990's. The number of charge/discharge cycles (cycle life) and performance for a battery is heavily affected by the materials used for the electrodes and electrolyte. Where early batteries used graphite anodes, newer batteries will use silicon nanoflake anodes [19] that can deliver up to 3.3× larger capacity than conventional graphite anodes. Although lithium-ion battery technology and capacity has improved over the years, it still cannot keep pace with the energy demands of today's mobile devices (Figure 2). For example, over a period of 7 years, the processing capability of the Samsung Galaxy S series has grown by ~20×, whereas, the battery capacity has grown only by ~2× [36]. Battery life limitations inevitably constrain performance for applications on mobile devices.

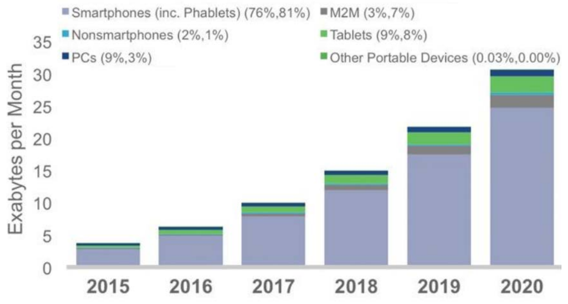


Figure 1: Data traffic offloaded per month from various types of computing devices to cloud datacenters [3].

Another critical determinant of mobile application performance is the device's thermal limitations. The heat produced by a mobile system-on-chip (SoC) must be removed efficiently by a cooling apparatus. As mobile devices cannot be equipped with bulky cooling solutions (e.g., fans) that will hamper their slim/small form factors, there are limits on the maximum power that mobile SoCs can dissipate (called thermal design power or TDP). Figure 2 shows the gap between the thermal power limits and desired power dissipation of mobile SoCs with respect to developments in mobile communication capabilities and chip technology. If application performance is to scale, such thermal limits must be overcome.

While battery technology improvements and better cooling solutions are slowly catching up to mobile requirements, OS based

RSP'17, October 15–20, 2017, Seoul, Republic of Korea
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5418-9/17/10…$15.00
https://doi.org/10.1145/3130265.3138855

software optimizations present an interesting opportunity for energy-efficient resource utilization. The most popular OS in smartphones today is Android that currently holds about 85% of the market share as of 2017 [3]. At the core of Android is a Linux kernel on top of which exist libraries written in C that serve as the Hardware Application Layer (HAL), and application software running on a framework that consists of Java-compatible libraries. An interesting aspect of this layered design is that often each layer can be developed independently and can be easily modified as per individual requirements of a device. This is especially true when the individual components in the software stack have vast amounts of code with high complexity. For example, the Linux kernel is continuously in development all year round, completely independent of other Android source code bases. Therefore, a modular and layered approach towards software allows for flexibility and rapid software deployment in mobile devices.

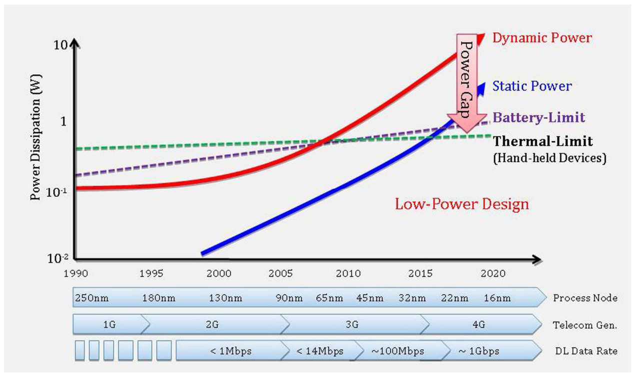


Figure 2: Trends for power dissipation relative to battery and thermal limits in mobile devices over the past few decades [4].

Middleware represents an important software component that takes advantage of this layered design found in complex software systems. It is intended to provide a service or functionality to the application developer that is not already a part of the OS. It is often dubbed as the "software glue" [6] or "plumbing" [18] as it allows independent software components to communicate and bring them together to produce new functionality for applications. A salient feature of middleware is the abstraction level that it can create for the application layer. An application developer does not need to be aware of how different modules come together for a new service to work. This allows for rapid development of applications by reducing the technical know-how required by app developers.

Middleware first became popular in the 1980's when it was used to interface newer applications to older legacy software. Today, it has found a widespread application in distributed systems and cloud based applications. One example is Microsoft Azure [18], that is a growing collection of integrated services (applications) and platforms (operating system) combined using middleware to deliver numerous services-on-demand in manner that is personalized per user. It must be noted that the function a middleware performs is heavily dependent on the context of the application and developer preference. They can also be used for security-authentication or creating completely new services by combining two or more existing services (e.g., merging database services and transaction processing services to enable transactional databases).

Lately, middleware has found a unique niche on mobile devices. The abstraction advantage and modularity of middleware allows it to be a very good choice for researchers to swiftly create services or modify the existing behavior of a deployed OS. For example, in [5] a smartphone middleware is developed that captures events from body sensor networks and relays the information to the appropriate mobile application. Throughout this paper, we look at various middleware solutions that lead to the rapid deployment of innovations in mobile devices with an intent to improve device energy-efficiency and performance robustness.

The rest of this paper is organized as follows. Section 2 discusses middleware for user-interaction aware execution of applications on mobile devices. Section 3 presents middleware that captures spatio-temporal context for various optimizations. Section 4 discusses middleware to enable mobile-to-cloud offloading. Section 5 explores middleware for mobile indoor localization.

2. USER-INTERACTION AWARE DESIGN

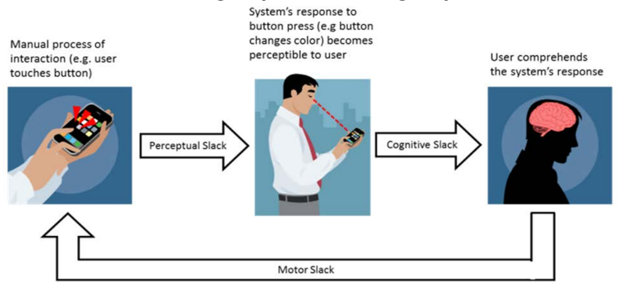
Mobile devices today are seen as a personal tool and their typical usage can vary across different users. OS- and hardware-driven energy-optimization techniques, such as CPU dynamic voltage and frequency scaling (DVFS), are not smart enough to make decisions based on the user's behavior. To enable more aggressive energy optimizations in mobile devices, we developed a novel application- and user-interaction aware energy management middleware framework (AURA) for mobile devices [7], [8]. AURA takes advantage of user idle time between interaction events of the foreground application to optimize CPU and backlight energy consumption. Most interestingly, AURA is able to adapt to changing behavior and learn from the individual user over time, to achieve longer battery lifetime without any user intervention.

Many research efforts have attempted to create energy-efficient mobile solutions. Most of this prior work mainly focuses on the management of wireless interfaces. For instance, [34] first measures the energy consumptions of various wireless interfaces with the goal of finding the most energy-efficient interface for use. A few methods [10], [11] involve allowing a mobile device to dynamically enter low-power sleep states by predicting idle periods, much like AURA. But these techniques are conservative in the savings they achieve as they do not exploit unique user characteristics as AURA does through its middleware-based framework.

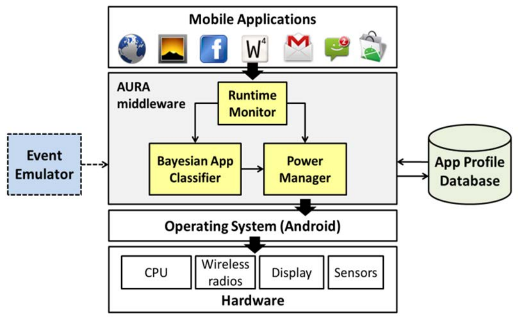
In order to create an energy-efficient middleware solution, it is important to first identify the components of the mobile device that are major contributors towards battery life. Our preliminary analysis revealed that the display, the processing (CPU/GPU) subsystem, and the various wireless radios (e.g., Wi-Fi, GPS, 4G/LTE) have a significant impact on battery life. Any framework that aims to optimize energy-efficiency must address the energy inefficiencies in these components. Our AURA framework [7], [8] was one of the first to reduce energy costs for both the display and the processing subsystems in an integrated manner. The framework consists of an app-aware and user-aware energy optimization middleware that uses powerful machine learning techniques on user-device interaction data. More specifically, AURA includes a runtime monitor that captures data related to user-specific and app-specific interaction slack to reduce energy costs.

Interaction slack (Figure 3 (a)) refers to the sum of the unused times between when a user first perceives a change on the display (perceptual slack) due to a previous interaction (e.g., a button on the screen changes color), then comprehends what the response "actually" represents (cognitive slack), and finally interacts with application again by touching the screen using his/her fingers (motor slack). By predicting this interaction slack interval on a per-app and per-user basis, AURA opportunistically reduce CPU voltage/frequency at the start of the interval and then increase the voltage/frequency just before the interval ends, to save energy without impacting user quality of service (QoS).

The Android OS allows for the creation of services that can run in the background. AURA's middleware was prototyped as a service that constantly runs in the background and creates an automated control system for CPU voltage/frequency scaling and display backlight modifications. The middleware consist of three main components: a runtime monitor, Bayesian app classifier, and a power manager. The runtime monitor (Figure 3 (b)) checks if the current foreground app has an entry in an 'interaction database', and if so then the interaction data (standard deviation and mean of a user's observed slack values from previous interactions) in it can be used for slack prediction. If a database entry does not exist, the middleware creates a new entry and starts collecting interaction statistics. A Bayesian classifier is then used to classify the interaction profile for the app using the collected data. Bayesian learning is a form of supervised machine learning that involves using evidence or observations along with prior outcome probabilities to calculate the probability of an outcome. The power manager runs a MDP (Markov Decision Process) to classify the apps. The apps were classified into seven categories ranging from very-low-interaction to very-high-interaction. This class of the app decides how to opportunistically decrease CPU voltage/frequency in between slack intervals. MDPs are discrete time stochastic control processes that are widely used as decision-making models for systems in which outcomes are partly random and partly controlled.


Manual process of interaction (e.g. user touches button)
System's response to button press (e.g button changes color) becomes perceptible to user
User comprehends the system's response
Perceptual Slack
Cognitive Slack
Motor Slack

(a)

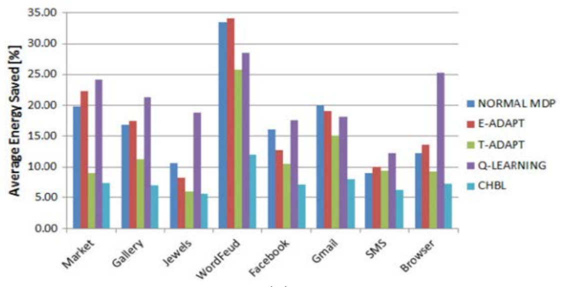


(b)

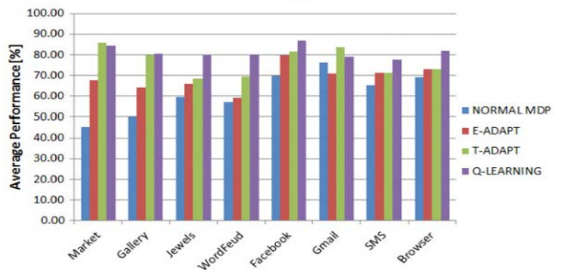
Figure 3: (a) Interaction slack estimation; (b) AURA energy optimization middleware framework for mobile devices [7].

In [7], we explored two derivatives of the (normal) MDP to dynamically adapt to real-time user-interaction during each invocation of an application. The E-ADAPT variant is event-driven and uses the most recent window of events to predict future interaction events whereas T-ADAPT makes use of a moving average window of a predetermined size, to dynamically track and predict user interaction events in a temporal context. In [8], we prototyped a new Q-learning based power manager. Q-Learning is a reinforcement learning technique that does not require a model of the environment and has the advantage that the next state probability distributions that are used in MDPs are not required. Using the Android services based modular middleware approach allowed us to the rapidly develop, update, deploy, and test new versions of AURA.

The AURA middleware also exploits the idea of change blindness [12], [35] as identified by research into human psychology and perception. Change blindness refers to the inability of humans to notice large changes in their environments, especially if changes occur in small increments. Multiple studies have shown this as a limitation of human perception– a majority of observers in one study failed to observe when a building in a photograph gradually disappeared over the course of 12 seconds; in another study, gradual color changes over time to an oil painting went undetected by a majority of subjects but disruptive changes such as the sudden addition of an object were easily detected. We used a similar approach by gradually reducing screen brightness over time using user-device and app-specific interaction data. In doing so, the power manager in AURA is able to achieve higher energy-efficiency without any noticeable loss in QoS.



(a)



(b)

Figure 4: Real user study results on Google Nexus One [8].

We deployed our AURA middleware framework on several smartphones such as the HTC Dream and Google Nexus One. Figure 4 shows the results for energy savings and interaction slack prediction on the Google Nexus One smartphone across various common mobile applications, averaged for several real users. In addition to the four variants of AURA (with power managers based on NORMAL-MDP, E-ADAPT, T-ADAPT, and Q-LEARNING) we compare against Shye et al.'s algorithm, CHBL [14], which was the best known algorithm for energy savings on mobile devices. It can be seen from Figure 4(a) that our user-aware and application-aware algorithms (particularly Q-LEARNING) offer higher energy savings than CHBL because unlike CHBL they can dynamically adapt to the user-interaction patterns and take full advantage of user idle time. Figure 4(b) shows the average successful prediction rates for the real user interaction patterns. CHBL was not included in the results because it does not contain defined states or prediction mechanisms, making determining mispredictions impossible. The figures show high successful prediction rates with AURA that result in high QoS for users. Our extensive experiments indicated 17% energy savings on average compared for

AURA to the baseline Android device manager and approx. 2.5× more energy savings over the best known prior work (CHBL [14]) on mobile CPU and display energy optimization.

3. SPATIOTEMPORAL CONTEXT-AWARE DESIGN

Today mobile devices come with a variety of wireless interfaces such as GPS, Wi-Fi and 3G/4G. Experimental analysis on the Google Nexus One Android smartphone [15] shown in Figure 5 indicate that even when 3G, Wi-Fi, and GPS interfaces are all enabled and idle, they account for more than 25% of total system power dissipation. Furthermore, when only one of the interfaces is active, the other two idle interfaces still consume a non-negligible amount of power. This is the motivation behind efforts to enable intelligent management of such wireless interfaces. It is important to note that activation of wireless interfaces, for location or data, is directly correlated with the context of the device itself, e.g., the type of application running, device motion, Wi-Fi availability, time of day, location, etc. This observation opens up an opportunity to realize a context-aware solution that is able to more efficiently manage wireless interfaces without human intervention.

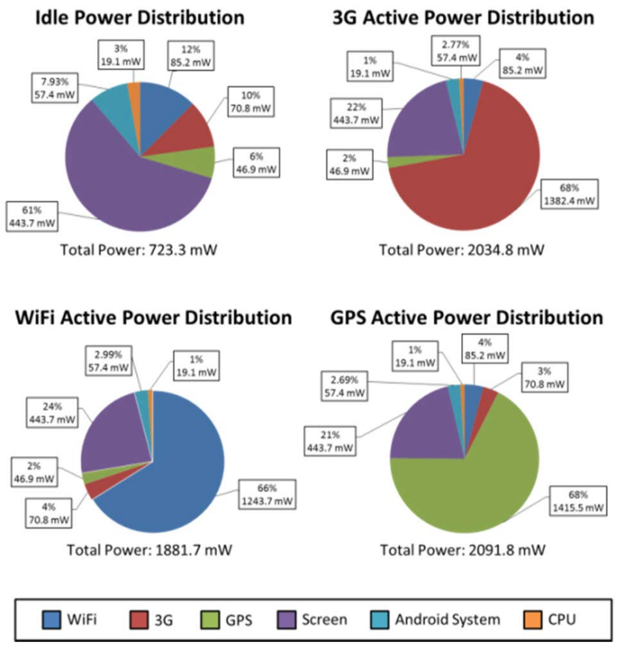


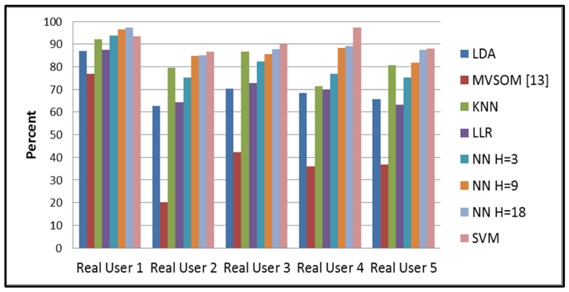
Figure 5: Google Nexus One mobile power distribution [15].

Our middleware framework in [16], [17] represents one of the first efforts towards seamless wireless interface energy management in mobile devices. The first step in our approach is to collect and learn from the contextual data of the device, the user, as well as the state of wireless interfaces. Our framework is able to transparently capture contextual data attributes such as temporal use data (e.g., day of week and time), spatial environment data (e.g., ambient light, Wi-Fi RSSI, 3G/4G signal strength), and device state (e.g., battery status, CPU utilization). To prune the massive amount of data captured, we employed Principal Component Analysis (PCA), a form of dimensionality reduction, by projecting the captured data from various sources onto a fewer number of optimally selected eigenvectors, effectively reducing the attribute space to the (limited) number of eigenvectors, to enable efficient prediction on resource-constrained mobile devices.

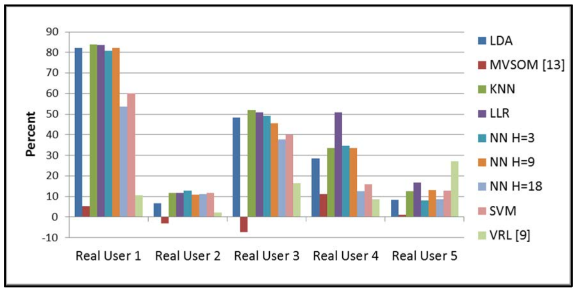
We then explored the use of five different classes of machine learning algorithms to learn from this contextual data. The algorithms we considered included LDA (Linear Discriminant Analysis), LLR (Linear Logistic Regression), NN (three variants of Neural Networks with number of hidden layers = 3, 9, and 18), KNN (K-Nearest Neighbor), and SVM (Support Vector Machines). These algorithms allowed us to predict user data/location usage requirements (e.g., is data transfer needed? is coarse-grained location needed? is fine-grained location needed?) based on the pruned spatiotemporal user and device context data collected.

Figure 6 (b) illustrates the energy savings achieved by the individual algorithms when deployed in the middleware layer of Galaxy Nexus mobile devices and evaluated with five different users. To get a view of the user's satisfaction in the presence of the energy enhancements, we also analyzed context prediction accuracy for the techniques (Figure 6 (a)). We compared our algorithms against the VRL technique (Variable Rate Logging [9]) and the configuration prediction strategy presented in [13] (MVSOM – Missing Values Self-Organizing Map).

From Figure 6(a), support vector machines and the application of neural networks with a number of hidden units of at least half the size of the attribute space (HH=9, 18) resulted in the highest prediction accuracy. K-nearest neighbor (KNN), linear logistic regression (LLR), and linear discriminant analysis (LDA) also performed fairly well, with prediction accuracies in the range of 60 – 90 %. However these approaches were much more sensitive to the usage pattern. MVSOM performed the worst and had a high degree of variance in both the usage pattern and random training data selection. Note that as VRL does not predict system state, results for its prediction accuracy are not shown.



(a)



(b)

Figure 6: (a) Algorithm prediction accuracy and (b) percent energy savings on Galaxy Nexus for real users [17].

It is important to note that despite high prediction accuracy, the amount of potential energy savings is still highly dependent on the user's device usage pattern and if the algorithms are positively or negatively predicting states where energy can be conserved. More complicated user patterns are more difficult for the algorithms to predict correctly. In addition, false predictions can cause either more or less energy to be consumed. From the results for energy savings in Figure 6(b), some of the highest energy savings are achieved with LDA and LLR. However, these higher savings come at the cost of degraded user satisfaction (Figure 6(a)). SVM and KNN overall perform fairly well in terms of both prediction accuracy and energy savings potential, as does the nonlinear logistic regression with NN approach. With the latter, an important point to note is that prediction accuracy is proportional to the complexity of the neural network and indirectly proportional to the net energy savings. This outcome is expected as less complex neural networks will result in more generalized models relaxing the constraint for inaccurate predictions that result in higher energy savings. It is also important to note that although energy savings are small for heavy users, this comes as an artifact of our optimization technique – we are exploiting windows of opportunity, which are fewer for heavy users. MVSOM, with its low prediction rates, also led to instances of negative energy savings, as it often predicted higher energy states when the true target state was one of less energy consumption. Thus we believe that the MVSOM approach is not very viable for use in mobile systems. VRL's energy saving capability is constrained because it does not disable device interfaces (only deactivates location logging or reduces logging rate), ignoring idle energy consumption.

A comparison of the runtime of our proposed middleware layer machine learning algorithms on different mobile and non-mobile chipsets is shown in Table 1. The use of middleware based approach allowed us to quickly switch between different machine learning techniques without having to modify the system as a whole. KNN's run time is several orders of magnitude larger than any of the other algorithms, because all computations are deferred until classification. Therefore, although KNN is as good as or better than the support vector machine (SVM) and neural network (NN) based approaches in terms of energy savings and prediction accuracy, it is not the most practical for deployment on mobile devices. Our SVM based middleware prototype provides good accuracy, good energy savings, and demonstrates the best adaptation to various unique user usage patterns, while maintaining a low implementation overhead on mobile devices.

Table 1: Average algorithm run times in seconds [17].

Algorithm	Intel Core i5	Qualcomm 8520 Snapdragon	Nvidia Tegra 2
LDA	0.00139	0.00361	0.07687
LLR	0.00118	0.00307	0.06525
NN (all 3 variants)	0.00962	0.02501	0.53199
KNN	97.7428	254.131	5405.18
SVM	0.00037	0.00095	0.02021
MVSOM [13]	0.82701	2.15023	45.7337
VRL [9]	0.01977	0.05140	1.09328

4. MOBILE-TO-CLOUD OFFLOADING

The collection and processing of data on smartphones can significantly hamper the battery lifetime of the device. A promising solution that is being considered to support high end mobile data processing applications is to offload mobile computations to the cloud [21]-[25]. Offloading is an opportunistic process that relies on cloud servers to execute the functionality of an application that typically runs on a mobile device. In many cases, such opportunistic offloading can not only improve energy-efficiency but also makes computation performance more robust.

Kumar et al. [26] presented a mathematical analysis of offloading. Broadly, the energy saved by computation offloading depends on the amount of computation to be performed (C), the amount of data to be transmitted (D), and the wireless network bandwidth (B). If (D/C) is low, then it was claimed that offloading can save energy. Our experiments have shown that this is a simplistic view of the problem, e.g., energy-efficiency is also highly effected by the type of wireless interface being used for the transmission. Cuervo et.al [21] proposed a framework called MAUI, based on code annotations to specify which methods from a software class can be offloaded to the cloud. Annotations are introduced in the source code by the developer during the development phase. At runtime, methods are identified by the MAUI profiler, which performs the offloading of the methods, if the bandwidth of the network and data transfer conditions are ideal. However, this annotation method puts an extra burden on the already complex mobile application development process. A better approach would be to have a middleware that is able to automatically make intelligent offloading decisions on the fly, without manual annotations.

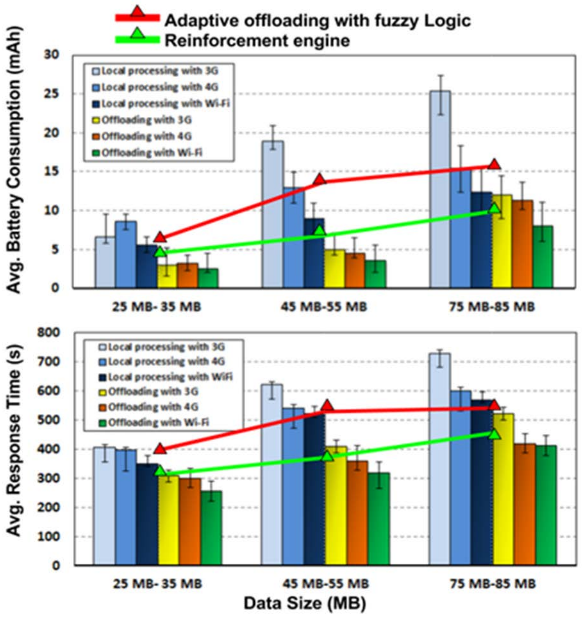


Figure 7: Average battery consumption and response time on a mobile device for a torrent file download application [27].

In spite of existing research highlighting the potential of offloading in mobile devices, current offloading techniques are far from being adopted widely in mobile systems. This is mainly because of some of the challenges associated with offloading. The first challenge is to be able to recognize the energy spent by the mobile device in the offloading process before the computation takes place. The energy saved with offloading should always be greater than the energy spent establishing and realizing computations in the cloud, which is not straightforward to estimate. The second challenge is that the response time of the application should be maintained such that the user does not experience a drop in QoS. In the real world, the response time and offloading energy is impacted by wireless network quality and the wireless interface in use. For example, 4G networks provide higher bandwidths than 3G but 4G also typically consumes more energy than 3G (and Wi-Fi). In other scenarios, poor Wi-Fi signal quality may make it more advantageous to use 4G for data transfers. The trade-off between 3G, 4G, and Wi-Fi is in general highly dependent on network conditions, which vary across locations and time of day.

In [27], we proposed such a middleware framework for mobile devices that utilized various sources of data such as the application communication/computation intensiveness, type and status of available wireless networks, and the capabilities of cloud servers, to make decisions on when and how to offload an application from the mobile device to the cloud, with the goal of improving energy-efficiency and performance robustness. The middleware framework was based on an unsupervised Q-learning machine learning technique that analyzed the data for the app type, network type, network conditions, and cloud capabilities to select the optimal network type and decide when and what to offload.

Figure 7 shows an example of how the Android based torrent app Flud [28] can benefit from our middleware-based offloading framework. In this experiment, a cloud based service (Amazon Web Services EC2 instance) first downloads and aggregates the file to be received through torrent, and then the smartphone downloads the file in a single process. The experiment was conducted on an LG G3 Android smartphone. From the bars in the figure, it can be observed that different network types, the state of the network, and data transfer sizes result in varying improvements in energy and response time. For instance, 4G performs slightly better than 3G in terms of energy consumption for higher data sizes (45-85 MBs), but for smaller data sizes 3G is more efficient. The colored lines in Figure 7 indicate the performance of our middleware framework (green line) and a framework based on fuzzy logic for making offloading decisions (red line) [24]. In all cases, it can be observed that our framework is able to provide better offloading performance and greater energy efficiency. This is because our framework employs a more sophisticated and powerful learning algorithm and considers many more variables related to device and network context when making decisions, than prior work.

Our experiments with real smartphones showed savings of up to 30% in battery life with up to 25% better response time when using our middleware framework compared to the state-of-the-art fuzzy logic based offloading approach from [24]. For certain applications, e.g., voice recognition, we found that offloading can also improve recognition robustness (accuracy) by approx. 10%.

5. MOBILE INDOOR LOCALIZATION

Location tracking has found various applications outdoors. One can not only use GPS based services for navigation purposes, but companies such as Google have been providing users location based services such as locating the best places to eat in their vicinity, local news, local weather, reminders to get an item when near a grocery store [29], etc. The outdoor location based services available today are extremely helpful, yet, in most cases they are limited once a user moves indoors. For instance, in the previously suggested example of reminders when near grocery stores, the application can only remind a user to buy the item from the store but is unable to provide any guidance on how to locate that item within the store. There are several other use cases that remain unrealized, such as being reminded to go to a certain store within a large indoor mall, notifications to the user when they are close to specific items/aisles in a store, or navigation help to reach specific rooms in a large building. These and many other examples make a strong case for the creation of indoor localization solutions on devices that most people carry with them everywhere: their smartphones.

Indoor localization is a challenge that cannot be resolved through a conventional outdoor solution such as GPS. This is because GPS signals are weak and ineffective in indoor environments, and the wireless signal-based infrastructure for indoor localization is diverse, prone to interference, and often entirely non-existent [30]. A possible approach to overcome this challenge is fingerprinting, where the goal is to use data captured through smartphone radio interfaces and sensors to estimate the location of the user indoors (inside of buildings, caves, etc.) in real time. However, continuous monitoring of radio and sensor data drains battery life, thus indoor localization solutions must be energy-aware.

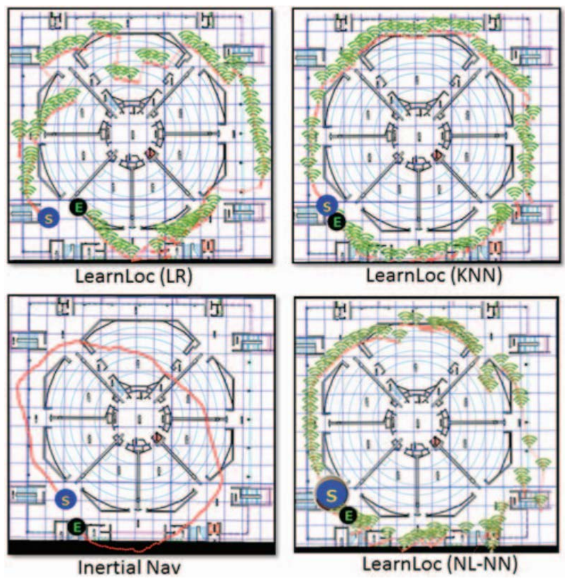


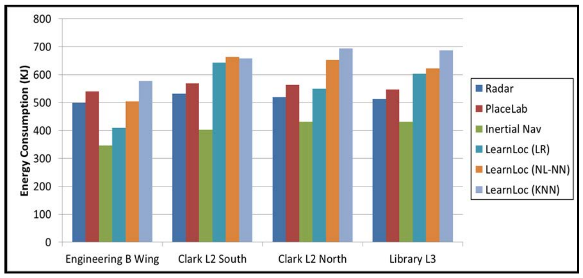
Figure 8: Paths traced by indoor localization techniques along the Clark L2 North building benchmark path [31].

We devised the LearnLoc middleware framework for mobile devices in [31] to improve indoor localization accuracy and energy efficiency in a variety of indoor environments. We capture Wi-Fi "fingerprint" data (signal strength, signal type, access point ID) in a continually updated database for indoor locales together with inertial sensing readings (accelerometer, gyroscope, and magnetometer sensor data) on mobile devices. This data is then analyzed and parsed with machine learning techniques to estimate location in real time. The framework is implemented as a middleware service to provide indoor location based updates and suggestions in a non-intrusive and energy-efficient manner. The most important feature of our approach is that it does not require any additional hardware setup indoors, as Wi-Fi access points have become increasingly common in indoor public spaces which brings down the cost of realizing indoor localization.

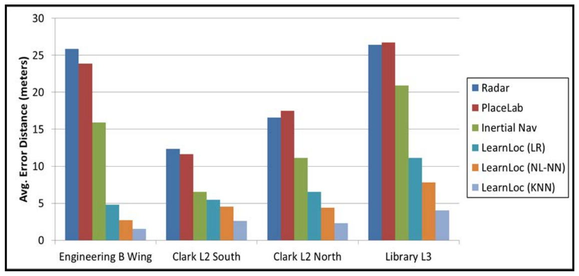
In [31], we prototyped and compared three variants of the LearnLoc framework that use Linear Regression (LR), non-linear neural networks (NL-NN) and K-nearest neighbor (KNN) techniques. Figure 8 shows the paths traced with the three variants of LearnLoc and a conventional inertial navigation (Inertial_Nav) approach for an indoor path in the Clark L2 North building at Colorado State University, with an HTC Sensation smartphone. It can be observed that the path traced by the Inertial_Nav technique greatly deviates from the actual path. This is due the accumulation of error overtime, whereas LearnLoc allows for recalibration by the use of Wi-Fi fingerprinting. The use of smart machine learning techniques helps LearnLoc significantly improve prediction accuracy and overcome noisy data readings compared to prior work.

Figure 9 shows a comparison of energy consumption and localization accuracy of the three variants of the LearnLoc framework with well-known techniques from prior work (Radar [32], PlaceLab [33], Inertial_Nav [20]) along four indoor paths of 110m-140m at Colorado State University. We observe that PlaceLab and

Radar consume much less energy but that comes at a price of accuracy. The experimental results suggest that the KNN delivers the most accurate location estimates, but also consumes the most energy. The LR variant performs considerably well overall. It is also important to note that Wi-Fi scan interval also plays a significant role in accuracy and energy-efficiency. The lowest Wi-Fi scan interval (1 second) delivers the best result, but a balance between energy consumption and accuracy can be achieved through the selection of a balanced scan interval. By prototyping a middleware framework that enables trade-offs between energy and accuracy, our work has brought viable indoor localization solutions that can be implemented on commodity mobile devices closer to reality.



(a)



(b)

Figure 9: Comparison of indoor localization techniques [31].

6. CONCLUSIONS

We are witnessing the rise of the data-driven science paradigm, in which massive amounts of data is produced and processed by mobile devices on a very strict power budget. These conditions call for energy-efficient software solutions that are able to help in the robust development of applications and services that can make the most out of available hardware resources. In this paper, we argue that the flexibility, scalability and abstraction level provided by a middleware based solution can lead to rapid prototyping of energy-efficient and robust performing solutions required to enhance the capabilities of emerging smart mobile devices.

ACKNOWLEDGMENTS

This work was supported by grants from the National Science Foundation (ECCS- 1646562, CCF-1252500).

REFERENCES

[1] 'Pew Research Center', 2017 [Online] Available: http://www.pewglobal.org /2016/02/22/smartphone-ownership-and-internet-usage-continues-to-climb-in-emerging-economies/

[2] Cisco Global Cloud Index: Forecast and Methodology, 2015–2020, White paper, [Accessed: 30 July 2017]

[3] 'IDC: Smartphone OS Market Share', 2017 [online] Available: http://www.idc.com/promo/smartphone-market-share/os

[4] 'System level Power Budgeting', 2017 [Online] Available: http://chipdesignmag.com/sld/blog/2014/03/12/system-level-power-budgeting/ [Accessed: 30 July 2017]

[5] C. Seeger, K. Van Laerhoven and A. Buchmann, "MyHealthAssistant: An Event-driven Middleware for Multiple Medical Applications on a Smartphone-Mediated Body Sensor Network," Journal of Biomedical and Health Informatics, vol. 19, no. 2, pp. 752-760, March 2015.

[6] 'Middleware: The Glue of Applications', 2017 [online] Available: https://www.gartner.com/doc/300144/middleware-glue-modern-applications/ [Accessed: 6 September 2017]

[7] B. Donohoo, C. Ohlsen, S. Pasricha, "AURA: An Application and User Interaction Aware Middleware Framework for Energy Optimization in Mobile Devices", IEEE ICCD 2011, Oct. 2011.

[8] B. Donohoo, C. Ohlsen, S. Pasricha, "A Middleware Framework for Application-aware and User-specific Energy Optimization in Smart Mobile Devices", Journ. Pervasive and Mobil Comp., vol. 20, pp. 47-63, Jul 2015.

[9] C. Lee, M. Lee, and D. Han, "Energy efficient location logging for mobile device," in Proc. SAINT, Seoul, Korea, Oct. 2010, p. 84.

[10] A.W. Min, R. Wang, J. Tsai, M.A. Ergin, T.C. Tai, "Improving energy efficiency for mobile platforms by exploiting low-power sleep states", in: CF'12, 2012, pp. 133–142.

[11] L. Huang, "Optimal sleep-wake scheduling for energy harvesting smart mobile devices", in: WiOpt'13, May 2013, pp. 484–491

[12] D.J. Simons, S.L. Franconeri, R.L. Reimer, "Change blindness in the absence of a visual disruption", Perception 29:1143–1154, 2000.

[13] L. Batyuk, C. Scheel, S. A. Camtepe, and S. Albayrak, "Contextaware device self-configuration using self-organizing maps," Proc. OC, 2011, pp. 13–22.

[14] A. Shye, B. Scholbrock and G. Memik, "Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures," IEEE/ACM Int. Sym. on Microarchitecture (MICRO), pp. 168-178, 2009.

[15] "Google Nexus One Tech Specs" [Online]. Available: http://www.htc.com/ us/support/nexus-one-google/tech-specs

[16] B. Donohoo, C. Ohlsen, S. Pasricha, C. Anderson, "Exploiting Spatiotemporal and Device Contexts for Energy-Efficient Mobile Embedded Systems", IEEE/ACM DAC 2012, Jul. 2012.

[17] B. Donohoo, C. Ohlsen, S. Pasricha, C. Anderson, Y. Xiang, "Context-Aware Energy Enhancements for Smart Mobile Devices", IEEE Trans. on Mobile Computing (TMC), 13(8):1720-1732, 2014.

[18] 'Microsoft Azure, [online], Available: https://azure.microsoft.com/en-us/overview/what-is-middleware/

[19] T. Kasukabe, et al. "Beads-Milling of Waste Si Sawdust into High-Performance Nanoflakes for Lithium-Ion Batteries." Science Reports 7, article num. 42734, 2017.

[20] J. Á. B. Link, P. Smith, N. Viol and K. Wehrle, "FootPath: Accurate map-based indoor navigation using smartphones," IPIN 2011.

[21] E. Cuervo, A. Balasubramanian, D. K. Cho, A.Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in Proc. ACM Mobisys, 2010.

[22] H. Flores and S. Srirama, "Mobile code offloading: should it be a local decision or global inference?" Proc. ACM Mobisys, 2013.

[23] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," Proc. IEEE INFOCOM, 2012.

[24] H. R. Flores and S. Srirama, "Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning," Proc. ACM Mobisys, 2013.

[25] A. Khairy, et al., "Smartphone energizer: Extending smartphone's battery life with smart offloading," in IEEE IWCMC, 2013.

[26] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" Computer, vol. 43, 2010.

[27] A. Khune, S. Pasricha, "Mobile Network-Aware Middleware Framework for Energy Efficient Cloud Offloading of Smartphone Applications", to appear, IEEE Consumer Electronics, 2017.

[28] 'Fuld – Torrent Downloader app', 2016, [Online]. Available: Android App Store, [Accessed: 3 Nov 2016].

[29] 'Guide to Google Cards, 2017 [Online] Available: https://www.android central.com/ultimate-guide-google-now-cards

[30] C. Langlois, S. Tiku, S. Pasricha, "Indoor localization with smartphones", to appear, IEEE Consumer Electronics, 2017.

[31] S. Pasricha, V. Ugave, Q. Han and C. Anderson, "LearnLoc: A Framework for Smart Indoor Localization with Embedded Mobile Devices," ACM/IEEE CODES+ISSS, Oct 2015.

[32] P. Bahl, and V. Padmanabhan, "RADAR: An in-building RF-based user location and tracking system," IEEE INFOCOM. 2000.

[33] A. LaMarca, et al., "Place Lab: Device Positioning Using Radio Beacons in the Wild," Proc. PERCOM, 2005, pp. 116-133.

[34] M. Segata, B. Bloessl, C. Sommer, F. Dressler, "Towards energy efficient smart phone applications: Energy models for offloading tasks into the cloud", in: IEEE International Conference on Communications, ICC, 2014.

[35] D.J. Simons, R.A. Rensink, "Change blindness: Past, present, and future", Trends Cogn. Sci. 9 (1) (2005).

[36] 'Samsung Galaxy S specifications', [Online], Available: https://www.phone arena.com/phones/Samsung-Galaxy-S_id4522