The Case for Operating System Management of User Attention

Kyungmin Lee, Jason Flinn, and Brian Noble
University of Michigan

ABSTRACT

From wearable displays to smart watches to in-vehicle infotainment systems, mobile computers are increasingly integrated with our day-to-day activities. Interactions are commonly driven by applications that run in the background and notify users when their attention is needed. In this paper, we argue that existing mobile operating systems should manage user attention as a resource. In contrast to permission-based models that either allow applications to interrupt the user continuously or deny all access, the OS should instead predict the importance and complexity of new interactions and compare the demand for attention to the attention available after accounting for the user's current activities. This will allow the OS to initiate appropriate interactions at the right time using the right modality. We describe one design for such a system, and we outline key challenges that must be met to realize this vision.

1. INTRODUCTION

Mobile computing systems are increasingly integrated with our day-to-day activities. This trend will only grow with the rise of wearable computing platforms such as Google glass and smart watches, as well as with the deployment of pervasive platforms such as in-vehicle infotainment systems.

Many mobile applications help users while they perform primary tasks such as walking, driving, and interacting with other people and their environment. A user engaged in such tasks may have very limited attention to spare for the mobile application. Consequently, such applications often run in the background and try to interact with the user only when such interaction will be meaningful. This is a fundamentally different model of interaction than that used by traditional desktop systems. Instead of the user initiating the interaction at a convenient moment, e.g., by opening the application, the application now initiates the interaction, e.g., via a smartphone notification or by an audio tone from an in-vehicle system.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
HotMobile'15, February 12–13, 2015, Santa Fe, New Mexico, USA.
Copyright © 2015 ACM 978-1-4503-3391-7/15/02 ...$15.00.
http://dx.doi.org/10.1145/2699343.2699362.

Consequently, mobile systems can no longer defer to the user the decision of whether to initiate a new interaction; for instance, they must decide whether or not to disturb the user by delivering an audio, visual, or haptic notification. This decision requires that they balance the anticipated importance of the interaction with the distraction caused by interrupting the user's primary activity.

We argue that the operating system should ultimately be responsible for making this decision. User attention is a limited and precious resource, and the operating system should manage how applications are allowed to consume that resource. Thinking of attention as an OS-managed resource is a valuable framework; for instance, we can draw analogies to multiprocessor scheduling and consequently leverage techniques from that domain to help manage this new resource.

Currently, most mobile operating systems use a simple, permission-based approach for managing attention. For instance, when an application is installed, the user may allow or deny permission to deliver audio or haptic notifications on a smartphone. This approach is too coarse-grained. A user may consider some notifications to be more important than others. Availability of attention varies; e.g., the user may be able to briefly interact with an in-vehicle computer when stopped at a red light, but not when driving on a crowded highway. Permission-based systems do not adjust for these factors; an "allowed" setting causes the mobile device to beep, vibrate, or otherwise disturb its user with notifications when she is not available, and a "denied" setting causes her to miss important notifications even if she has ample attention to spare.

The OS is uniquely suited to manage user attention. Determining the right time to interrupt the user requires understanding the user's current activities, which in turn requires access to raw sensor data and sensitive information such as calendars. If the computer were to delegate attention management to applications, then every application would require access to a wealth of private information; in contrast, the OS is already trusted with the privacy of this data. Additionally, since the OS is traditionally responsible for allocating limited resources to competing applications, it is the logical point at which to consider the competing demands for attention from multiple applications so as to avoid overloading the user and so as to prioritize notifications that have the highest global importance to the user. Finally, a single implementation in the OS reduces development effort compared to implementing notification management in every application.

Applications should be responsible for notifying the OS when they wish to interact with the user; they should express the possible modes of interaction and assign an application-local importance to the request. The OS should convert the application-local importance to a global scale and quantify the expected attention that the user will need to devote to the interaction. The supply and demand for attention can be decomposed into distinct elements such as audio, video, haptic, and cognitive attention—this decomposition allows for the possibility, for example, of listening to streaming music while driving a vehicle. The OS will also infer the importance and attention demanded by the user's current activities; this includes both activities external to the computer system (e.g., walking or talking) as well as internal activities (e.g., the user's interactions with other applications). If the user has attention to spare, or if the importance of the notification is judged to be high enough to interrupt the user's current activity (and thus free up sufficient attention), the OS initiates the interaction. Otherwise, the interaction is deferred for a more opportune moment. If the application presents multiple modalities of possible interaction, then the OS chooses the best one based upon its assessment of available audio, visual, haptic, and cognitive attention.

In the next section, we describe in more detail our vision for system management of attention. In particular, we draw inspiration from the scheduling of jobs on a heterogeneous multicore system and model the supply and demand of attention within a similar framework. This allows us to leverage an existing body of work on multicore scheduling. Then, Section 3 presents some ides about how the OS can estimate the supply and demand of user attention, as well as the importance of current tasks and notifications. We then discuss related work and conclude.

2. OVERVIEW

Management of user attention as a resource should be the responsibility of the operating system, but the operating system needs input from both applications and the user in order to do a good job. An application possesses a great deal of domain-specific knowledge that can be used to determine the relative importance and attention required by the interactions that the application initiates. The user can provide valuable feedback about what interactions were and were not worthwhile so that the operating system can learn models that adapt the systems interruption behavior to that user's preferences. In this section, we outline interfaces that separate these concerns among the operating system, the user, and applications.

As with most current mobile systems, we envision that an application running in the background must go through the OS to initiate a user interaction (e.g., as today's mobile phone apps must go through system software to deliver a push notification). Rather than simply apply a blanket policy to allow or deny such interactions, we propose a more nuanced approach. The application provides a numeric measure of how important it believes the interaction will be to the user. This measure is application-specific; it ranks the importance only relative to other interactions initiated by that application. The application also provides a list of the possible modalities of interaction (e.g., an e-mail application might be able to read a new e-mail aloud or display the content on a touchscreen). For each modality, the application may optionally specify a quantitative prediction of the user attention that will be consumed during the resulting interaction.

This interface requires that the system have some model for quantifying attention. While any such model will necessarily be a gross simplification, there are some properties we wish to expose. First, attention may take many forms. For instance, a driver may have the attention to listen to directions from a GPS application but not to look at the screen. This suggests that attention could be expressed as a vector over those various forms (e.g., audio, visual, haptic, cognitive, and other forms of attention). Second, paying attention can be thought of imposing a load over a given time period on each of those forms.

This leads to a very useful analogy: attention management can be modeled as a scheduling problem in which each form of attention is a separate core on a heterogeneous multiprocessor. A user interaction can be modeled as imposing load on some or all of these cores. Interactions require gang scheduling [4] since performing the task requires simultaneous consumption of audio, cognitive, etc. attention. Further, attention consumed by external tasks can be modeled as additional jobs that impose a measurable but time-varying load. Interactions have an importance, equivalent to task priority in scheduling, and pausing and resuming interactions has a non-negligible cost for context-switching. Note that the advantage of such a model is not that it is the most accurate possible model of user interaction, but rather that it is an approximation that maps the problem into a domain (multiprocessor scheduling) in which there is a large body of existing research results. This allows the OS to leverage sophisticated algorithms and apply them to a new resource.

Returning to the API, an application specifies the attention it predicts an interaction will take as a load imposed on the attention vector over a time period. As described in Section 3.2, we envision that such estimates can be calculated as a function of low level I/O interactions such as text displayed, number of buttons pressed, and so on. We also describe methods by which the operating system can provide estimates for the application by observing its past interactions and learning models that predict future attention consumed.

As described in Section 3.3, the operating system monitors the user's current activities and interactions with other applications to continually estimate the load on attention. If the new interaction would not overload any of the forms of attention, then it is allowed. If not, the operating system converts the application-local importance into a global priority as described in Section 3.1. It then considers if the priority of the interaction is greater than that of any current activity by a margin sufficient to overcome the context switch cost of interrupting the current activity. It checks if the additional attention freed by interrupting that activity is sufficient to schedule the requested interaction. If so, it allows the application to initiate the interaction (e.g., by displaying a notification, generating an audio tone, etc.). If not, the operating system defers the interaction until a more opportune time (of course, the application may cancel the interaction request before that time arises). While we have described an algorithm for only a single possible mode of interaction and a single competing activity, sophisticated scheduling algorithms can be used to schedule multiple activities and/or choose between different modalities of interaction.

Next, the user chooses whether or not to respond to the application. Note that although some attention has already been consumed, a user may always choose to ignore or delay interacting with the application — this indicates the operating system made a poor decision. Learning from such experiences is a key part of our design. Some feedback can be gathered automatically. For example, if the user does not respond to the notification (e.g., by turning on the phone screen to see the notification), then the user was likely too busy to notice or deal with any interaction. Thus, the operating system's estimate of current activity importance or load was likely too low.

Often, explicit user feedback could be quite useful. The challenge is too allow for such feedback in a non-intrusive manner. One possibility is to provide a few simple options ("not now", "not important", "great, I wanted to know that", etc.) that are easily accessible via voice recognition, swipe gestures, etc. and improve system models from such examples. We discuss this possibility in Section 3.1.

Thus, we envision the operating system having ultimate responsibility for deciding when and how new interactions take place. However, the operating system takes input from both applications and users to make such decisions. In the next section, we describe some of the more challenging subproblems in building such a system.

3. CHALLENGES

In this section, we outline three challenges that must be overcome to realize our vision, and discuss possible solutions to each.

3.1 Predicting interaction importance

Ideally, a mobile system should only interrupt the user for activities that are more important than whatever tasks the user is currently performing. Further, the difference in importance should be sufficient to make up for the context-switch overhead of pausing and resuming the current activity. For instance, a mobile device should not bother a user who is in a meeting for an advertising e-mail from a retailer, but it should likely ping the user for an urgent e-mail from his boss.

The application is in the best position to assess the importance of interactions that it initiates relative to all other interactions it initiates. For instance, current e-mail applications assess with high accuracy spam, advertising, and priority e-mails, and social applications such as Facebook can identify high-interest updates. In contrast, the OS lacks the context and visibility to make such fined-grained distinctions. Thus, we ask the application to provide a numeric ranking of importance for each proposed interaction relative to the other interactions the application initiates.

The OS can not blindly rely on the application's ranking of importance. Some applications may purposely overestimate their importance to game the system. The ability to predict importance may vary substantially from application to application; e.g., some e-mail services may do a good job of identifying spam and high priority mail, while others may not. Finally, it is difficult for an application to assess how much worth it provides to any given user. Some people are social networking addicts and others care not a whit for the latest updates from their friends.

We therefore propose that the OS learn a function that maps the local importance provided by an application to a global scale. For each interaction, the OS can gather data about whether the user found the interaction important. Given sufficient data, the OS can normalize the application predictions. Thus, an application that games the system by marking all interactions as high-priority or one that is unable to differentiate between low-importance and high-importance interactions will see its global importance fall over time, while applications that consistently provide good estimates will be more likely to interrupt the user for their high-priority interactions.

Often, the OS can gather some data by observing user behavior after it initiates each interaction. If the user receives a notification and performs the task associated with the notification, then there exists positive evidence the user found the interruption useful. In contrast, if a user dismisses or ignores the notification, then there exists evidence that the interaction may have been less important than believed.

Although a user's response time to a notification is a good indicator, the response time alone is not always conclusive. We may also need to consider the user's attention level at the time the notification was received. For instance, if a user is driving a car and receives an important e-mail, that user will likely not read the e-mail until later. Even though the notification is important, the user's response time is high due to other high-priority tasks. We can account for this by also including the importance and complexity of the user's current activities in the model.

Due to the limits of passive observation, we also want to allow users to help train a better model through explicit feedback. One possibility is to give user's simple choices to describe the quality of the interaction. Some possibilities are: "Yes, I wanted to know that", "Why didn't you tell me sooner?", "I'm too busy", "That's not important", etc. Similar to spam filtering, we can present the user with a small number of optional feedback choices during each interaction. If specifying such choices are optional and easy to accomplish (e.g., via voice recognition or a special swipe), then a user can customize their models with only a moderate amount of additional work.

Explicit feedback also helps the system to adapt to different users' behavior and preferences. For instance, some users may rarely wish to be interrupted, while others may welcome constant notifications. Further, behavior may not map directly to importance: e.g., a text from a spouse to pick up milk from the store may be dismissed quickly but it also may be of high importance.

3.2 Predicting attention demand

In addition to predicting the importance of a future interaction, the OS must also predict its complexity, i.e., the amount user attention that will be consumed by the interaction. As described in Section 2, we view attention as a vector of different forms of interaction (audio, visual, haptic, and cognitive)—a prediction should therefore map the proposed interaction onto this vector.

The approach that we plan to take for this mapping is inspired by our prior work on AMC [15]. In that work, we measured load by observing interactions with an in-vehicle touchscreen. For example, we measured button presses required to complete a task, the amount of text on the screen, the size and placement of text, the presence of animation features, and so on. For each low-level measurement, we applied a threshold to determine whether or not an application

demanded too much attention to be used while driving a vehicle. A threshold was appropriate for this work since we only considered one possible foreground activity (driving).

We can broaden this approach by using quantitative functions to map low-level interactions to specific forms of attention such as audio or visual. For instance, a notification that displays text demands visual attention; a larger amount of text displayed naturally maps to a greater demand on attention. Using AMC-like tools, we can monitor how the user interacts with an application in response to a notification; e.g. through voice recognition, touchscreen events, and button presses, and we can measure the type and quantity of output produced. From low-level measures of, e.g., text displayed, buttons pressed, voice commands issued, etc., the OS can derive a quantitative measure of attention demanded during each interaction.

The simplest possible approach would be to predict that each new interaction will demand roughly the same attention required by prior interactions initiated by the same application. A more flexible approach could allow the application to also specify some local measure of complexity; e.g., an e-mail application could calculate this measure from the length of an e-mail, the presence of attachments or images, etc. Similar to the prior section, the OS could then learn a function that maps application-local complexity estimates to the global, measured complexity of the actual interaction.

An application may support multiple modes of interaction; e.g., it might read a text message aloud or display it on the screen. We envision that such applications will expose these modalities to the OS and the OS will then learn a separate model for each one. This would make it possible for the OS to realize that an incoming text could be read aloud to a walking or driving user even though the message could not be displayed on a screen.

3.3 Measuring available attention

The final major challenge is assessing the user's available attention. This is more challenging than measuring demand because it will usually involve detecting and evaluating user activities external to the computer system such as driving, walking, conversing, etc.

Further, a blanket classification of activity will be insufficient. For instance, a person driving a car on a empty, straight highway will typically have some attention to spare, e.g., to select music, whereas the same person driving at rush hour on a snowy day may have no available attention. Therefore, in order to accurately determine the available attention level, a mobile system needs to consider not only the user's current activity (or possibly multiple activities) but also user's engagement level with each activity.

Fortunately, there is a considerable body of work on activity recognition that we can use to meet this challenge. Mobile devices possess myriad of sensors (e.g., GPS, accelerometer, microphone, camera, gyroscope, etc). Usage of these sensors [14] for activity recognition has been well studied [13, 7, 18, 19, 20]. For instance, Kern et al. [13] use audio sensor data and a classification algorithm to determine whether a user is in a lecture, on the street, in a conversation, or at a restaurant. They also use body-worn accelerometers to determine whether a user is sitting, standing, walking, or running.

Thus, we could rely on these results to enable the OS to determine the activities that a user is currently engaged in performing. If the computation needed for classification is too burdensome for a mobile device, it can potentially be offloaded to a trusted remote server [6]. Note that the applications need not be entrusted with either the raw sensor data or the activity observations in this design.

Activity	Possible attention level
Sitting around	Very low - Very high
Playing with a phone	Mid - Very high
Walking	Mid - High
Having a conversation	Low - High
Writing an e-mail	Low - Mid
In a meeting	Very Low - Mid
Driving at high speed	Very Low - Mid

Table 1: Examples of activities and range of possible attention level

Next, the OS can infer the possible range of available attention based on the recognized activity. Table 1 illustrates one hypothetical mapping between some activities and their corresponding range of attention consumption. Additional sensors can be involved at this point to narrow the range. For instance, if a user is driving a vehicle, speed and position can be read from the CAM bus and GPS unit respectively, while road conditions can often be inferred from traction control and ABS data.

Additionally, an OS can rely on supplementary data such as a user's calendar to obtain information about the environment (e.g., whether the user in a important meeting). Since different environments require different level of user engagement, a mobile OS can use user's current environment as a hint about the user's engagement level.

In addition to activity and engagement level sensing, a mobile OS can use information about user's cognitive state to make decision on whether to interrupt or not. For instance, Lu et al. [17] created a system that determines user's stress level in real-time using a mobile device's microphone. Alternatively, a user's emotional state can be important. LiKamWa et al. [16] describe a system that can determine a user's emotion based on his mobile device usage patterns. When a user is annoyed or stressed, an unwanted notification could be more annoying than usual. In our model, the context switch cost increases in such instances, which would bias the OS against interruption.

Activity and engagement sensing is an important and nascent field. Our objective is to provide a framework in which research results from this field can be used by the operating system to make better decisions about initiating user interactions. Thus, as results in this area continue to improve, our OS can do a better job of determining how and when to request its user's attention. Even with the challenges outlined in this section, we believe that OS support will do substantially better than current allow/deny permission models.

4. RELATED WORK

The detrimental effect of poorly-timed notifications in a desktop environment has been well-studied [3, 2, 11]. However, as Iqbal et al. [12] suggest, users are willing to tolerate some disruption in return for receiving valuable no-

tifications. These results demonstrate the need for a user context-aware notification system.

Determining the best time to interrupt a user for a notification has been studied extensively [1, 8, 9, 10]. Horvitz et al. [9] developed PRIORITIES, an desktop e-mail notification system that uses a Bayesian model to infer the user's available attention level and compute the expected cost of interruption and deferring alerts. When the benefit of an alert outweighs the cost of interruption, the system delivers the notification to the user. We agree with the principles of this work, but argue that such solutions should be implemented by the OS rather than by individual applications. Further, current notification systems must deal with the complexity of mobile environments in which the user may be devoting attention to walking, driving, or other tasks.

Recently, there has been research on determining proper task break points for mobile devices. Fischer et al. [5] determined the end of mobile device interactions to deliver notifications at such instances. Okoshi et al. [18] determined accurate application-specific break points, during which the user can be interrupted while she is using an application. Ho et al. [7] determined when the user is transitioning from one physical activity to another (e.g., from sitting to walking) using body-worn accelerometers and used those moments to deliver notifications. These prior systems do not consider the importance of the notification, nor do they consider the possibility of interrupting an activity to initiate a new task. Our goal is to initiate appropriate interactions even when doing so requires the user to interrupt a current task.

Kern et al. [13] proposed a notification design that senses the user's environment and delivers socially acceptable notification modality to the user. It can sense when a user is in a lecture and knows not to disrupt the user with a loud noise. This design is the closest to our proposal, but it can only detect four environments and six user activities. Furthermore, it treats all notifications with the same importance.

Additionally, in contrast to all these prior approaches, we propose a specific framework for cooperation between applications, the user, and the OS to determine when to initiate new interactions. Thus, a major focus of our work is to determine how best to manage attention as a service provided by the mobile operating system.

5. CONCLUSION

In this paper, we argue that a mobile device's operating system should be responsible for managing user attention as a resource. With this new responsibility, a mobile OS can create a user attention-aware notification system that initiates new interactions at the right time with right modality without interrupting high-importance tasks. We have laid out a design and methodology for creating such a system, and we have identified key challenges in realizing our vision.

Acknowledgments

We thank the anonymous reviewers of this paper for their thoughtful comments. This research direction was inspired by a conversation with Venkatesh Prasad. The work was supported by a grant from Ford Motor Co. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Ford or the University of Michigan.

6. REFERENCES

[1] Piotr D. Adamczyk and Brian P. Bailey. If not now, when?: The effects of interruption at different moments within task execution. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 271–278, Vienna, Austria, April 2004.

[2] Brian P. Bailey and Joseph A. Konstan. On the need for attention-aware systems: Measuring effects of interruption on task performance, error rate, and affective state. *Computers in Human Behavior*, 22(4):685–708, 2006.

[3] Mary Czerwinski, Eric Horvitz, and Susan Wilhite. A diary study of task switching and interruptions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 175–182, Vienna, Austria, April 2004.

[4] Dror G. Feitelson and Larry Rudolph. Gang scheduling performance benefits for fine-grain synchronization. *Journal of Parallel and Distributed Computing*, 16(4), December 1992.

[5] Joel E. Fischer, Chris Greenhalgh, and Steve Benford. Investigating episodes of mobile phone activity as indicators of opportune moments to deliver notifications. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, pages 181–190, Stockholm, Sweden, August 2011.

[6] Jason Flinn. *Cyber Foraging: Bridging Mobile and Cloud Computing*. Morgan and Claypool Publishers, September 2012.

[7] Joyce Ho and Stephen S. Intille. Using context-aware computing to reduce the perceived burden of interruptions from mobile devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '05, pages 909–918, Portland, Oregon, April 2005.

[8] Eric Horvitz and Johnson Apacible. Learning and reasoning about interruption. In *Proceedings of the 5th International Conference on Multimodal Interfaces*, ICMI '03, pages 20–27, Vancouver, Canada, November 2003.

[9] Eric Horvitz, Andy Jacobs, and David Hovel. Attention-sensitive alerting. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI'99, pages 305–313, Stockholm, Sweden, July 1999.

[10] Shamsi T. Iqbal and Brian P. Bailey. Understanding and developing models for detecting and differentiating breakpoints during interactive tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 697–706, San Jose, California, April 2007.

[11] Shamsi T. Iqbal and Eric Horvitz. Disruption and recovery of computing tasks: Field study, analysis, and directions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 677–686, San Jose, California, April 2007.

[12] Shamsi T. Iqbal and Eric Horvitz. Notifications and awareness: A field study of alert usage and preferences. In *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work*, CSCW '10, pages 27–30, Savannah, Georgia, February 2010.

[13] Nickey Kern and Bernt Schiele. Context-aware notification for wearable computing. In *Proceedings of the 7th IEEE International Symposium on Wearable Computers*, pages 223–230, Washington, DC, October 2003.

[14] N.D. Lane, E. Miluzzo, Hong Lu, D. Peebles, T. Choudhury, and AT. Campbell. A survey of mobile phone sensing. *Communications Magazine, IEEE*, 48(9):140–150, Sept 2010.

[15] Kyungmin Lee, Jason Flinn, T.J. Giuli, Brian Noble, and Christopher Peplin. Amc: Verifying user interface properties for vehicular applications. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 1–12, Taipei, Taiwan, June 2013.

[16] Robert LiKamWa, Yunxin Liu, Nicholas D. Lane, and Lin Zhong. Moodscope: Building a mood sensor from smartphone usage patterns. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 389–402, Taipei, Taiwan, June 2013.

[17] Hong Lu, Denise Frauendorfer, Mashfiqui Rabbi, Marianne Schmid Mast, Gokul T. Chittaranjan, Andrew T. Campbell, Daniel Gatica-Perez, and Tanzeem Choudhury. Stresssense: Detecting stress in unconstrained acoustic environments using smartphones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 351–360, Pittsburgh, Pennsylvania, September 2012.

[18] Tadashi Okoshi, Hideyuki Tokuda, and Jin Nakazawa. Attelia: Sensing user's attention status on smart phones. In *16th International Conference on Ubiquitous Computing*, pages 139–142, Seattle, Washington, September 2014.

[19] Carlos Paniagua, Huber Flores, and Satish Narayana Srirama. Mobile sensor data classification for human activity recognition using MapReduce on cloud. In *Proceedings of the 9th International Conference on Mobile Web Information System*, pages 585–592, Ontario, Canada, August 2012.

[20] Yunus Emre Ustev, Ozlem Durmaz Incel, and Cem Ersoy. User, device and orientation independent human activity recognition on mobile phones: Challenges and a proposal. In *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*, UbiComp '13 Adjunct, pages 1427–1436, Zurich, Switzerland, September 2013.