Application Sequence Prediction for Energy Consumption Reduction in Mobile Systems

Conference Paper · October 2015

DOI: 10.1109/CIT/IUCC/DASC/PICOM.2015.7

4 authors:

Ismat Chaib Draa
LAMIH-University of Valenciennes and Hainaut-…
5 PUBLICATIONS **2** CITATIONS
SEE PROFILE

Jamel Tayeb
Intel
10 PUBLICATIONS **7** CITATIONS
SEE PROFILE

Smail Niar
University of Valenciennes and Hainaut-Cambr…
146 PUBLICATIONS **441** CITATIONS
SEE PROFILE

Emmanuelle Grislin-Le Strugeon
University of Valenciennes and Hainaut-Cambr…
74 PUBLICATIONS **244** CITATIONS
SEE PROFILE

Application Sequence Prediction for Energy Consumption Reduction in Mobile Systems

Ismat Chaib Draa*, Jamel Tayeb†, Smail Niar*, Emmanuelle Grislin*
*LAMIH, University of Valenciennes, Valenciennes, France
{firstname.lastname}@univ-valenciennes.com
†Intel Corporation, Portland, USA
{firstname.lastname}@intel.com

***Abstract*—The success of mobile devices as measured by market penetration is undeniable. Tablets and Ultrabooks are enjoying similar growing enthusiasm among the worldwide population. By offering an ever increasing number of functionalities, these devices turned to be even more subject to energy constraints than the previous generation. Limited battery autonomy becomes a major user concern and cause of dissatisfactions. Therefore, managing efficiently the energy consumption can improve mobile systems reliability, battery life as well as user experience. In this paper we propose a new approach to optimize mobile devices energy efficiency based on use patterns detection. By identifying and classifying users behaviors, we can significantly improve over the platforms stock power managers. To do so, a run-time service is proposed to collect usage data, which in turn can be mined using machine learning techniques. Our apprach allows us to predict future applications usages, so the CPU frequency, Wi-Fi connectivity and the playback sound-levels can be optimized while meeting the applications and the users requirements. Our experimental results show that the proposed solution can lower the energy consumption by up to 20% vs. the out-of-the-box power governor, while maintaining a negligible system overhead.**

***Index Terms*—Energy consumption, Run-time users and application analysis, Applications sequences prediction.**

I. Introduction

Mobile phones and smart devices such as Tablets, E-readers, Ultrabooks are being adopted at phenomenal pace and are fundamental tools used in business, communication and social interactions. In 2015, the mobile cellular subscriptions per 100 citizens has exceeded 240 in Hong Kong, 150 in Singapore and 140 in Brazil. These mobile systems provide to the user high computing and processing performances, continuous network access, audio and video recording and more. They also contain a set of powerful embedded sensors as an accelerometer, gyroscope and GPS. A closer look to the hardware components shows that the devices contain powerful processor, RAM and colorful screen display. All these characteristics and components impact significantly the total energy consumption. One of the biggest issues of the user is the fear of full battery discharge. All these features and details highlighted several academic and industrial works around the optimization of energy consumption in mobile systems which must be achieved at four different levels: microelectronic, electronic, operating system and applications. In mobile systems, the operating system (OS) itself acts to improve the battery life by adapting resources usage via automatic control of display back-light, dynamic scaling of CPU frequency and controlling the communication peripherals. However the increasing number of applications, diversification of their nature in terms of computing and communication requirements makes energy the primary bottleneck for handled smart devices [9]. The key for power saving is therefore to leverage users' behaviors and habits to predict applications' run patterns and improve upon the default energy management policies of the OS. The main contributions of this work can be summarized as follows.

1) With rich sensor hubs, a large set of data is used to derive user context information. These metrics are also used to gauge user needs and characterize his/her habits.
2) Classify and characterize the user behaviors and the running applications to decrease the energy consumption of the whole system.
3) The information obtained by classification and characterization about prior launched applications is collected to find frequent set of executed application sequences and to perform prediction. Applications sequences are repetitive over time and identify user and application needs over time.
4) By using the prediction mechanism, we are able to have information about the close future running applications. We adjust the resources automatically depending on the prediction phase and context information by performing optimizations such as: Dynamic Voltage Frequency Scaling (DVFS) [11], data prefetching and device management without impacting the user satisfaction. Some of theses optimizations will be detailed in the following.

The global architecture of our approach is shown in figure 2. This figure presents an abstraction of the several phases of our approach. The first one is collecting data from the user behavior such as running applications, background process, date and time. These data are exploited thru two mechanisms: static classification of applications in terms of resources and prediction of applications' patterns. Then, the dynamic **Optimizer Actuator** uses the outputs of the classification and the prediction to perform actions like device management and applications scheduling in order to reduce the energy consumption. The rest of this paper is organized as follows. In Sec. 2 the architecture of our framework is presented. In

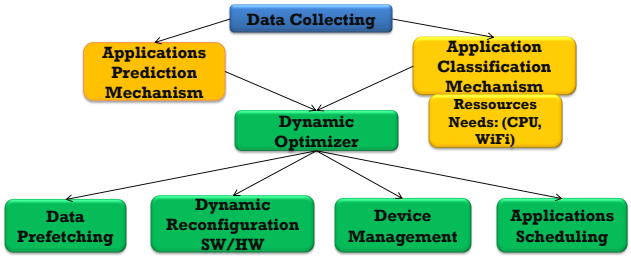

Figure 1: Overview of the proposed approach

Sec. 3, we present in details the classification mechanism in terms of connectivity and computing needs. In Sec. 4, we present the used probes to collect information about the user. We illustrate the key idea of this paper which consists in the prediction of application sequences and execution time for each application. In Sec. 5, experimental results with 6 user applications traces is given. Comparison in terms of consumed energy between our approach and the OS energy management is used as measurement for energy savings. Sec. 6 presents the related word and finally we conclude in Sec. 7.

II. Framework Architecture for Energy Consumption Optimization

Our approach consists in the development of a new methodology to optimize the energy consumption of the mobile system and improve the energy management of the OS. The functional architecture of the proposed framework includes 6 steps, given in figure 2 which is a specification of figure 1. The objective of the first step is to represent the mobile device user with a specific usage and behavior. The purpose is to improve the energy management by capturing user behavior and system usage. In the second step, we process the collected data thru the **analyzer** which is used to guaranties user privacy anonymizing the data. Then in third step, the data collected for the optimization are stored in a data base. In the fourth step, we have 2 phases. In the first one, the collected data are uploaded to the back end component in order to be analyzed via operations like application classification, application sequences and user behavior profiling. The second phase consists in building optimization rules. In step 5, these rules are pushed to a component (actuator) in order to implement a specific optimization for each hardware component. In this paper, we present principally the data collecting mechanism, the application classification, the prediction and the optimization mechanism. First, we collect a large set of data which are: running applications, date, time, elapsed time of each application and background process. In this paper, we focused on two components which are Wi-Fi and CPU. In the current implementation, we focused on these 2 components because they are among the components that consume the most in a mobile system. The proposed solution is generic and can be extended for screen, microphone, GPS, ... etc.
The first phase is the application classification in terms of Wi-Fi and CPU need, this phase is achieved **statically**. Secondly, the average time for each application is calculated, this phase is also static but the data are updatable into database every week. Finally, we have the prediction of applications which is dynamic and is achieved online. All these phases are combined and exploited by the **Optimizer Actuator** which manage Wi-Fi connection and CPU frequency in order to optimize the energy consumption. The next section presents the classification phase.

Table I: Wi-Fi CLASSIFICATION

Apps	Internet Rate	Download	Upload	class
Messenger	32.55	12.93	19.62	ON
Youtube	366.09	325.81	40.28	ON
2048	1.56	1.02	0.54	OFF

III. Classification Mechanism

In this paper, the classification is achieved offline and is taken into account in the optimization phase like mentioned above. We classify applications in terms of CPU and Wi-Fi. In terms of CPU, an upper threshold in terms of CPU frequency is fixed for the applications. For the Wi-Fi, we have a binary classification (ON/OFF) for each application. Before adjusting resources the **Optimizer Actuator** consult the background process to avoid any conflict.

A. Classification in terms of Wi-Fi

This classification contributes to manage the wireless connection in the appropriate interval of time during running applications which don't require internet connexion. The aim of this classification is to avoid the user dissatisfaction when the wireless connection is disabled by the **Optimizer Actuator**. To achieve this classification, we realized some preliminary experiments. We calculate internet rate which is defined by the sum of the upload and download rate, when no application and background process are running. Then, a threshold was fixed at 10 KB because of the connectivity management in Windows, even if no applications needs connection, there is some connexion rate test provided by the OS. Secondly, we run application which we want to classify individually and a correlate between it and the internet rate. When the sum of the upload and download rate during the execution of the application is under the fixed threshold, we determinate that this application doesn't need a wireless connection and vice versa. To achieve so, some experiments have been achieved for a set of applications. The table 2 shows the classification result.

B. Classification in terms of CPU need

Windows 8.1 manages the CPU frequency automatically. However in some cases, the computing resources provided by the OS exceed what is required by the running applications. We propose to classify the applications in terms of CPU frequency relatively to several thresholds. In the current implementation we define 4 classes according to the application need in term of CPU. We realized some experiments to classify some applications in the 4 classes.

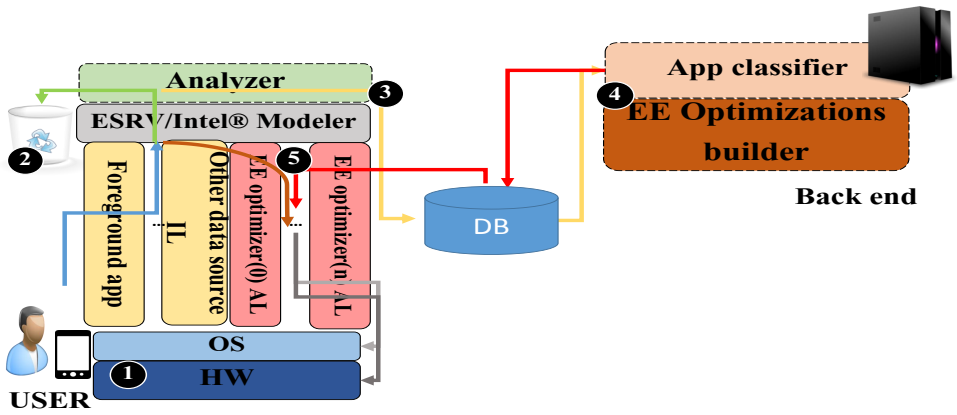

Figure 2: Architecture of the proposed framework

1) Class c_1: Applications requiring a **low** CPU frequency. Text processing applications such as *Word*, *Excel* or simple games *Imperial Sudoku* belongs to this class.
2) Class c_2: Applications requiring a **medium** CPU frequency. Web browser such as *Firefox*, *Google_Chrome* are in this class.
3) Class c_3: Applications requiring **high** computing resources. Elaborated games such *2048* belong to this class.
4) Class c_4: Applications requiring **very high** computing resources. Image processing and synthesis such as Image ray-tracing, simulation applications and mathematical applications are in this class. In the experiments no application is in this class.

We measure CPU utilization and CPU frequency during running applications in order to achieve our classification with the help of a known Bayesian technique. We first realize a profiling step where data collection is implemented. This step corresponds to *Data Retreving* in figure 1. In this phase, the average (m), the standard deviation (e), and the ratio of the used CPU frequency relative to the maximal frequency (f) are used to characterize each application. These values are then uses to assign each application to one of the 4 classes.

$$P(c_i|m,e,f) = max_{1\leq j\leq 4} P(c_j|m,e,f)$$

With

$$P(c_i|m,e,f)$$

is the probability for an application to classified in class i given m, e, and f

$$P(c_i|m,e,f) = \frac{P(c_i) * P(m,e,f|c_i)}{P(m,e,f)}$$

Where

$$P(m,e,f|c_i) = P(m|c_i) * P(e|c_i) * P(f|c_i)$$

And

$$P(m,e,f) = \sum_{i=1}^{4} P(m|c_i) * P(e|c_i) * P(f|c_i)$$

Table II: CPU CLASSIFICATION

Apps	avg-usage	stand dev	ratio freq	class
Chrome	25.88	7.65	0.71	Medium
Foxit	7.45	4.07	0.55	Low
2048	46	10.58	0.87	High

To calculate the different simple probabilities, we consider the distributions (m,e and f) as Gaussian, e.g. with μ the average and σ^2 the ratio variance,

$$P(f|c_i) = \frac{1}{\sqrt{2 * \pi * \sigma^2}} * \exp\left(-\frac{(f-\mu)^2}{2 * \sigma^2}\right)$$

The table 2 gives an example of application classification. For the CPU, the **Optimizer Actuator** acts on 3 frequencies. According to the application class, the Optimizer Actuator selects the most suitable upper threshold among the 3 following frequencies: 800 MHz (low), 1.25 GHz (medium) and 1.75 GHz (high). Obviously, the **Optimizer Actuator** does not disable the resources that are required by running background applications, like Wi-Fi for the application *Skype* for instance. After the static classification, we present in the next section the data collecting and prediction mechanism.

IV. Data Collecting and Prediction

The parameters related to the different users, like job, life style, age and gender, vary from one person to the next. This difference must be taken into account in order to propose an appropriate and customized mobile energy management for each user. In fact, application sequences, also called scenarios, are recurrent and correspond to distinct user situations. The idea is to analyze the various applications launched by the user according to the day of the week, the time, and the background processes.

A. User probe for data Collecting and time processing

User probe is linked to different applications launched by the user during a long period of time. As mentioned previously, the main parameter in our approach is date and time. In fact we assume that the user has different behaviors between weekdays and weekend, and also between distinct periods of a given

day. For example, the applications launched during Monday morning at work are different from the applications running during a Saturday night. The behaviors are also supposed to differ from one user to another, e.g. according to their jobs as one can work in an accounting office while the other one works outside. The **User Probe** is executed during the normal use of the mobile device. The probe collects each foreground window launched by the user and recovers the name of the related application, the application running in background for the parallel usage, the day of week, the date and time. The figure 3 resumes the functionality of the probe.

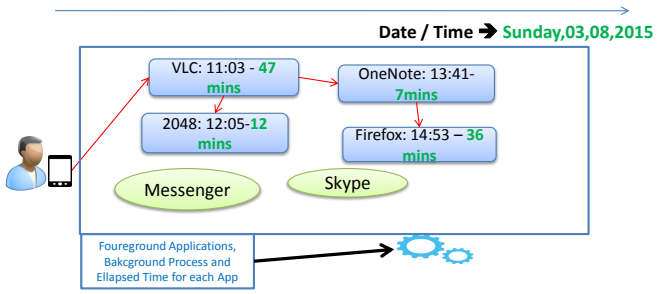

Figure 3: User Probe

Whenever the user launches a new application, the time spent on the previous one is calculated and registered in a data base in order to calculate the average running time length for each application, for a given day of the week and a given period of launching time.

B. Prediction of Future Running Applications

This phase represents the main purpose of this paper. As mentioned above, the principal idea is to propose a customized energy management of mobile systems which improves the standard energy management proposed by the operating system. Our approach depends on the user habits and the running application over time in order to predict the future running applications for a system resources consumption adjustment, and thus to reduce the energy consumption in specific scenarios. For the prediction needs, we used Generalized Sequenced Patterns (GSP), which is a well-known sequence mining algorithm [7]. The GSP algorithm is used to find frequent sequences of items, eventually revealing time-related correlations or causal structures among sets of data. Our motivation for using the GSP algorithm is to find regularities in the applications launched according to the day of the week, the time and the background process. More precisely, the items processed by the algorithm are the running applications in the same period of a given day during several weeks. For example, we would like to know if the same applications are frequently launched in the same order every Monday between 8 am and 10 am. The GSP algorithm can detect such frequent application sequences. A sequence is frequent when its occurrence in the database is over a specified threshold. At the end of the GSP processing, we have the number of applications occurrence from 1 application sequence to k

Table III: GSP input data example

Mon [9- 11]	Application Sequences
1st week	Excel,Mozilla,Spotify,2048,VLC
2nd week	Excel,Mozilla,Word,CALC,VLC
3rd week	Notepad,Mozilla,Word,2048,VLC
4th week	Mozilla,Word,Spotify,2048,VLC

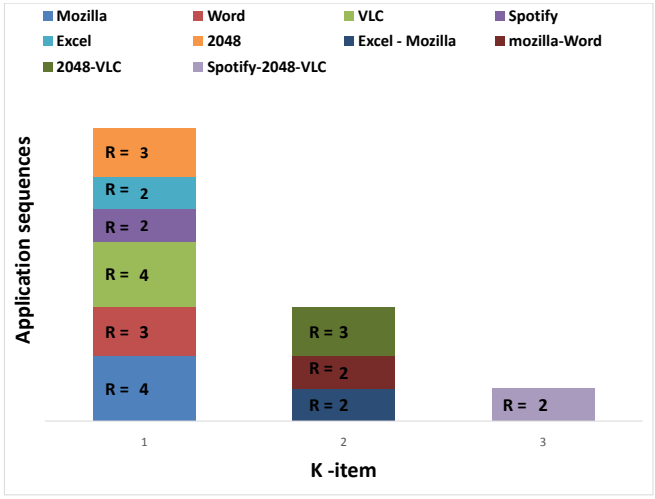

Figure 4: GSP Output Example

applications sequence. The table 1 shows a small example of data provided as input to the GSP process. These data show the running application sequences during 4 Mondays from 9 am to 11 am. The table 1 represents the input of the GSP algorithm. The output is the number of applications occurrence as shown in the figure 4. K represents the sequence length, i.e. the number of items in the sequence, and R is the repetition frequency, i.e. the number of times this sequence occurs. With K=1, the occurrence number R gives the number of times each single application appears; we note that *MOZILLA* and *VLC* are the most frequent applications. With K=2, the occurrence number R concerns sequences composed of 2 applications, e.g. the sequence *(2048, VLC)* that appears 3 times. With K=3, the occurrence number R concerns sequences composed of 3 applications; we note that the sequence *(SPOTIFY, 2048, VLC)* appears 2 times and several other sequences which appears one time. On the base of the GSP output and with the data about the foreground application, date and time dynamically collected by the **User probe**, the applications that will be launched the next Monday can be predicted. Distinct predictions can be made according to the sequence length, i.e. according to the K factor. With the length equal to 1 (K=1), the most probable application is predicted, whatever the current running application, whereas K=2 can predict the application that will follow the current one, like the game *2048* that appears after *VLC*. K=3 can be used either to predict which application will be launched after the sequence formed by both the current and previous application launches or to predict which will be the two applications that are likely to be launched after the current one. In this first works,

we only studied the latter case (prediction based only on the current application), in which choosing the lowest K is more accurate to predict the next application that will be launched. However, this increases the frequency at which the GSP rules are consulted, causing energy and time consuming. A balance must be found. The figure 5 illustrates the prediction mechanism thru **GSP** processing.

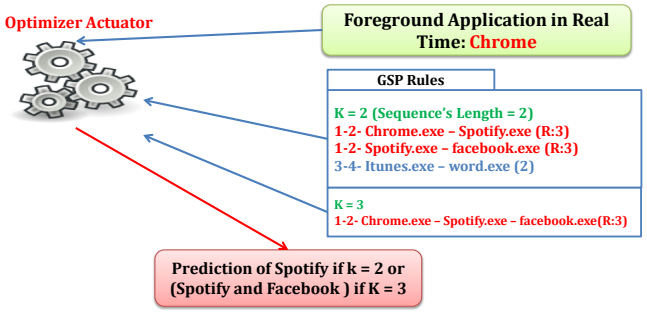

Figure 5: Prediction Example

In order to determine the most appropriate K factor, we conducted measures to study the energy overhead generated by different values for K. Let's take the example of the sequence [*Mozilla*:20mn - *Word*:17mn - *VLC*:47mn - *2048*:7mn]. With *Mozilla* as the current running application, the GSP rules predict the lauchning of *Word*, *Word-VLC* or *Word-VLC-2048* depending on the chosen K. The cost of the GSP rules consultation in terms of energy is by 2 watt/s. By choosing K=2, the energy overhead is about 6 watt/s, whereas if K=4, the energy overhead is about 2 watt/s. The difference is negligible and it is about 4 watt/s of overhead for 71 minutes. The difference prompted us to choose K=2 for more precision and a negligible overhead.
The next step is to adapt the energy management depending on the prediction phase, the elapsed time of the application runs and a classification in terms of resources. The **Optimizer Actuator** begins adjusting resources gradually for the application B before the end of the application A in order to not impact the user satisfaction. Indeed, a sudden change in terms of resource may impact heavily the satisfaction of the user, who would react in increasing resources manually such as CPU frequency, luminosity, ...etc after the automatic adjustment done by the **Optimizer Actuator**. In this paper, for The Wi-Fi management, we use only the classification phase.

V. Experimental Results

This section is divided into two parts, the first one presents the tools and the experimental environments, the second one presents the results.

A. Tools and Experimental Environments

This section presents the global framework, the tool components and their usage. As our work is developed in partnership with the Intel society, The Intel Energy Checker SDKit (iESDK) [1] has been used to implement our solution. The SDK has been designed to measure and optimize applications energy efficiency. Two components of the SDK are leveraged in this work: the main driver (ESRV - Energy Server) and the Modeler. The Modeler provides the services required to implement the use data collection and energy saving heuristics. Several data collection expansion modules, a.k.a. Inputs Libraries (ILs), as well as Actuators Libraries (ALs) have been developed. The Modeler is composed of three components: the Front-End, the Input Bus (IB), and the Back-End.

- The **front-end** collects the data: CPU Utilization, display brightness, battery level, front end applications, etc. Each metrics is called an input. Collecting new metrics requires the development of an **Inputs Libraries (ILs)**.
- Once collected by the ILs, metrics are made visible on the **Input Bus**. Any agent connected to the bus has direct access to the metrics. The IB is the main interface between the FE and the BE.
- The **Back-end** provides core services e.g. a logger or a power-to-inputs automatic correlation, a watchdog as well as an interrupts and communications manager. The BE can be expanded via **Actuators Libraries (ALs)**. ALs are designed to perform specific actions such as dynamic OS or platform configurations. Usually ALs are used to implement various optimization heuristics that are driven in real-time by the inputs provided by the FE.

Figure 6 depicts this architecture.

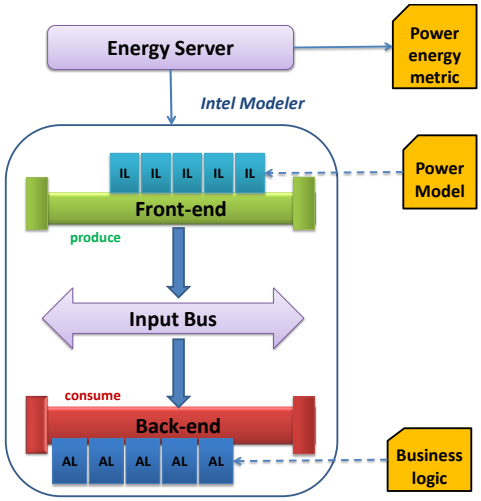

Figure 6: Interaction between the components

In this paper, the user probe was developed as IL to produce inputs to the back-end. and the optimizer actuator as AL to apply business logic on HW/SW components to manage the energy consumption of the mobile system.

B. Evaluations

In this section, we report the results of our experiments conducted to evaluate the performance of our proposal. The experimental results are based on the generated applications sequences from 3 synthetic users (simulated) and 3 real

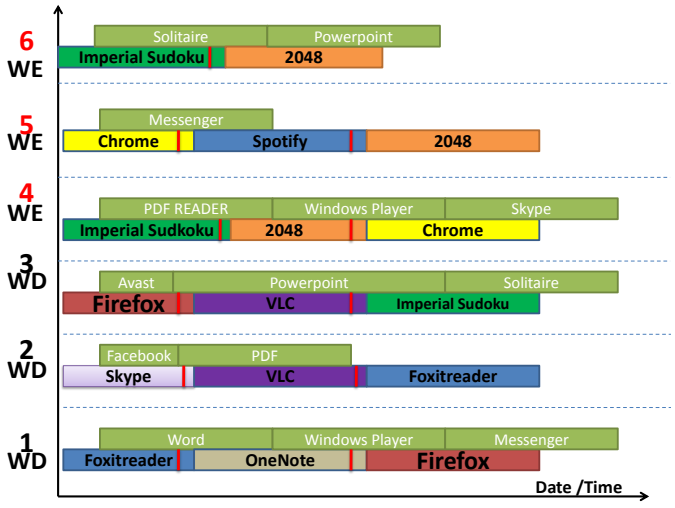

Figure 7: Users Application Sequences

users. These constitute first tests to validate the feasibility of the solution and is followed by validation with only real users in further works. In this section, for each user, we present the prediction application scenario and the applications classification. Finally, we present results about the obtained gain by our approach in comparison with the standard energy management provided by the OS. The experimentations have been realized on an Ultrabook running Windows 8.1 with a 2.50Ghz Intel dual-core i7-3667U processor and 4GB of RAM. Energy measurement has been experimented using a Yokogawa WT210 [3] power meter and the Intel Energy Server (ESRV).

1) Scenario Prediction and Applications Classification: In order to validate our approach, we simulated 3 users and collect data from 3 real users. For each user, we present the predicted sequence with the background process, the time spent in each application and its classification in terms of Wi-Fi and CPU need. In order to demonstrate the difference in the users behaviors and the launched applications depending on the day of week, we selected 3 users on a weekday and 3 users on weekend as shown in figure 7. The red lines on the applications (Figure 7) represent the beginning of the resources adjustments for the next application. Prior to evaluating the efficiency of the proposed approach. We first classified the different applications for each user. We use the information logged for two weeks to classify these information like mentioned in Sec 6. The table 4 shows the classification of applications of each user in terms of CPU need. The class value L represents the lowest class and the value H is the highest. A first look on the tables 4 shows the behavioral difference between the users whose run the same application. For example, with user 1 *Firefox* is classified as medium while it is classified as low with user 3 (Tests have been achieved in the same device).
This difference is due to the difference in the manner which user 1 and user 3 interact with *Firefox*.

Table IV: Applications Classification for CPU need

Apps	CPU Application Class					
	Usr1	Usr2	Usr3	Usr4	Usr5	Usr6
Firefox	M		L			
Skype		L				
Chrome				L	M	
Spotify		L			L	
VLC		L	L			
2048				H	M	M
Sudoku			L	L		L
Foxit	L	L				
OneNote	L					

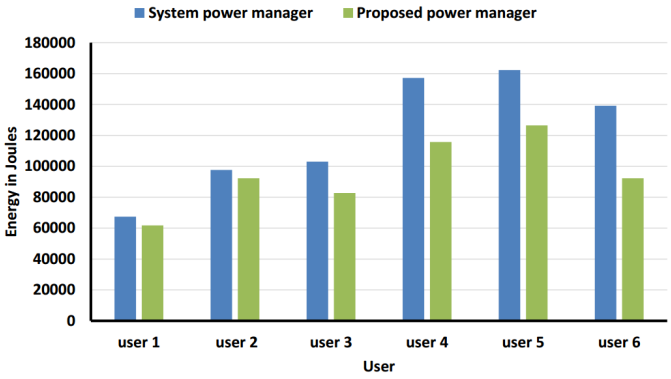

Figure 8: Energy Consumption For Each User

2) Power Saving: Obtained experimental results show that our approach can reduce the whole energy consumption of the system by 33% in comparison with the OS. Figure 8 shows the impact of the proposed solution on energy savings over varying durations of usage for 6 users. The results show that the saved energy varies from an user to another. For user 1 and user 2 the saved energy is by an average the 8 % and 5 %. For user 3 and user 5 the saved energy is more than 19% and 22%. For user 6 the saved energy is more than 33%. This variance is due to the difference of the running applications and the interaction manner of each user. For example, User 4 and user 6 run the applications *2048* and *chrome*, but power saving with user 4 is higher by 4% for power saving with user 5. The reason is the difference in terms of classes for *chrome* and *2048* between the user 4 and the user 5. In other hand the results confirm that there is a considerable impact of the type of the running application for the power saving, the applications characterized by a low work load, low connectivity need and low interaction with user are the lowest in terms of power saving. For example, user 2 runs *VLC, FoxitReader and Spotify*.
These applications have a low connectivity and computing need which impact the energy reduction. In order to understand the impact of each application on the energy consumption, the figure 9 shows the power reduction for each application with the 6 users.
A first look on the figure 9 indicates that each application can impact the energy consumption. For example, *Spotify* is an application characterized by low CPU need and low connectivity need, so the power reduced by this application is very

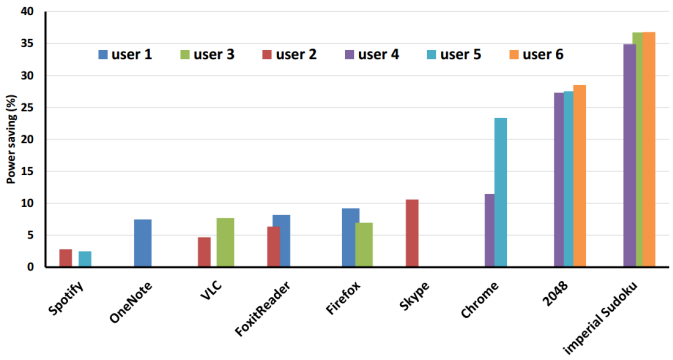

Figure 9: Power Saving For Each Application

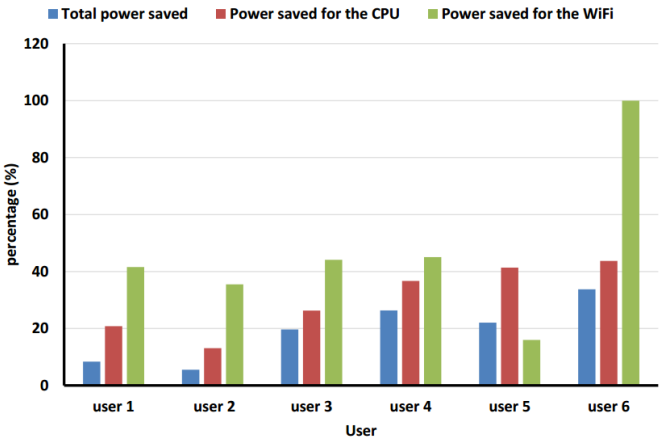

Figure 10: Power Saving With Our Approach

low around 2.5%. However *OneNote* is an application which has the same characteristics of *Spotify* unless the connectivity need, but the energy reduction with this application is more than 7%.

We can conclude an other observation, with an application characterized by a high need for CPU performance we can reduce an important amount of power. *Imperial Sudoku*, *2048* and *Google Chrome* are examples for these application. We can also conclude that there is relation between user interaction and consumed power by the application, for example *Google Chrome* reduces around 11% when it needs low CPU performance, but it reduces more than 23% when it needs medium CPU performance. Figure 10 shows the percentage of the power reduction by the proposed approach. By analysing the figure 10 we note that power saving for the Wi-Fi is relative to the time spent in running a specific application. The gain in terms of energy reduction obtained by switching OFF the Wi-Fi interface is constant. By switching OFF the Wi-Fi interface, the whole power consumption of the Ultrabook is decreased by 1.6 WATT. The total of the power reduction obtained by managing the CPU is relative to the running applications and the user behaviors and interaction manner as shown previously.

3) Overhead Evaluation: The overhead cost of the online mechanism is given in terms of CPU usage, physical memory,

Table V: Solution overhead

Mechanisms	Power Over-head	CPU (%)	RAM (MB)	TIME (ms)
Prediction (IL)	6	0.3	3.74	2000
Optimize Actuator (AL)	3.8	0.3	1.45	1500

power consumption of the prediction and optimization mechanisms. These results are presented in order to demonstrate that our proposed solution based on ILs and ALs does not affect system and does not cause performance degradation for the user, rather it allows the user to save his mobile system energy consumption in comparison with what it is proposed by the OS.

VI. Related Works

There is a large body of work focusing on energy consumption optimization in mobile devices. However, very few of them use the changing dynamic user and application needs in the control of the different resources. In this section, we some the existing works in energy consumption optimization for mobile systems at the application level. We mainly focus on the approaches that take into account user behavior and experience in energy consumption reduction. Finally, we highlight the main differences with our approach. One of the first works in this area is [12]. The authors demonstrated the benefit to study real user activities to characterize power consumption and to control the development of power optimization. Their experiments on an HTC ARM based mobile phone show important differences between users behaviors. The author demonstrated also that CPU and screen are the most demanding components in terms of energy. For the screen the total utilization time is dominated by a small number of long intervals, with a duration of about of 100 seconds. Other works like [4] show the importance to study the user's activities and behaviors to optimize power consumption in mobile systems. Theses studies demonstrated the correlation between energy consumption of a mobile system and user actions. In [13], the authors proposed an approach which takes into account the user experience to apply different power optimization techniques. They developed a new *cpufreq governor*. Their dynamic clock scaling approach provides a mechanism to change the clock speed of the CPUs at run-time. Their proposed *cpufreq governor* analyze the user perceived response time of the applications at runtime. Then this information is used to control the processor CPU frequency. The CPU energy consumption is reduced by up to 65.5% over the Android's default on-demand [8] *cpufreq governor*. The authors also exploited the characteristics of User/System interactions to minimize energy consumption. They used the elapsed time between two consecutive interactions to decrease the screen brightness during this interval in order to have a gain in terms of the whole system energy consumption. Authors in [5] and [10] proposed user activities and context information-based technics In [5]. Then, they proposed several management policies for each component. In

their approach the CPU frequency, for example, was adjusted dynamically depending on the workload. They also proposed to reduce background process life time depending on the obtained patterns. An energy consumption reduction of up 20% in comparison with commercial solutions like Juice Defender [2] has been obtained. However their solution was not completely automatic and required modifications in the running application source code. In [10], the authors proposed a classification of user activities in terms of screen brightness needs and correlate these data with the context information. Then, they used machine learning techniques to predict the required luminosity. With their approach, the authors reduced the energy consumption of the whole system to 23.5% without performance degradation. In [6], the authors presented the power monitor which is a client-server architecture developed to collect usage logs from Android powered devices. Based on the utilization patterns, power saving profiles are generated and are personalized to match the needs of each device in the system. The experimental results show that the power monitor can increase the battery life by almost 90%. However this solution has some privacy issues which are due to the exploitation of usage pattern generation. At the opposite of all the previous works and paper, our project differs in the following point:

- In our approach applications are first analyzed and classified off line. Then the launched applications over time and their needs are predicted using the new concept of application sequences. The aim here is to minimize the provided resources for each application.
- For application classification, in terms of computing and communication resources, data mining techniques. One of the contribution of this paper is that we combine the prediction mechanism with the classification. Predicting the application needs is very helpful and allowes to control the CPU frequency and Wifi interface.
- Our framework for energy optimization is modular architecture with the introduction of the ILs and ALs. This modularity along with the utilization of the Intel Energy Checker SDK, make our solution flexible. Adding a new input, for enhancing the optimizer by new data from a new sensor, or a new actuator for managing a new hardware component will be very easy.

VII. Conclusion and Perspective

In this paper, a new technique for energy consumption reduction in mobile systems has been proposed. Our approach is based on data mining, machine learning techniques. We also used the Intel Energy Server tool which allows the definition of new Input Libraries (IL) for collecting data and Actuators Libraries (AL) for controlling hardware components such CPU frequency and Wi-Fi. At the opposite of existing methods, where the users needs and behaviors are rarely taken into account, in our approach we, not only take these elements but, we also classify possible running applications, in terms of ressources needs, and we predict the future applications. In comparison to the advanced energy management provided by windows 8.1, for some situations the gain offered by our approach reaches 30%. As perspectives of the project, we will consider more possible user patterns and more applications. We will also use user satisfaction level as parameter to control our energy saving technique. The correction of our prediction mechanism will be also developed to increase the prediction accuracy. The offline classification phase will be extended to include luminosity, sound level and GPS needs. For this reason, we will expand the use of IL and AL in order to exploit other sensors, such as compass and GPS, and for other mobile platforms such as smartphones and tablets running Android/Linux and iOS. Finally, as applications are likely to have multiple requirements at different application phases at runtime, it will be interesting to consider application phases within the application, rather than an entire application as a whole.

VIII. Acknowledgement

The Authors would like to thank Intel Corporation and especially the Intel Research Council for the support given to the project and the tools.

References

[1] Intel energy checker software development kit userguide https://software.intel.com/en-us/articles/intel-energy-checker-sdk.
[2] Juice defender.http://www.juicedefender.com/.
[3] Power meter yokogawa wt210. http://www.electro-meters.com/yokogawa/yokogawa-power-meters/wt210/.
[4] C. Bunse. On the impact of user feedback on energy consumption. In *28th International Conference on Informatics for Environmental Protection: ICT for Energy Effieciency, EnviroInfo 2014, Oldenburg, Germany, September 10-12, 2014.*, pages 759–764, 2014.
[5] S. Datta, C. Bonnet, and N. Nikaein. Power monitor v2: Novel power saving android application. In *Consumer Electronics (ISCE), 2013 IEEE 17th International Symposium on*, pages 253–254, June 2013.
[6] S. Datta, C. Bonnet, and N. Nikaein. Personalized power saving profiles generation analyzing smart device usage patterns. In *Wireless and Mobile Networking Conference (WMNC), 2014 7th IFIP*, pages 1–8, May 2014.
[7] Y. Hirate and H. Yamana. Generalized sequential pattern mining with item intervals. *Journal of computers*, 1(3):51–60, 2006.
[8] V. Pallipadi and A. Starikovskiy. The ondemand governor. In *Proceedings of the Linux Symposium*, volume 2, pages 215–230. sn, 2006.
[9] R. A. Powers. Batteries for low power electronics. *Proceedings of the IEEE*, 83(4):687–693, 1995.
[10] M. Schuchhardt, S. Jha, R. Ayoub, M. Kishinevsky, and G. Memik. Caped: Context-aware personalized display brightness for mobile devices. In *Proceedings of the 2014 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '14, pages 19:1–19:10, New York, NY, USA, 2014. ACM.
[11] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *High-Performance Computer Architecture, 2002. Proceedings. Eighth International Symposium on*, pages 29–40. IEEE, 2002.
[12] A. Shye, B. Scholbrock, G. Memik, and P. A. Dinda. Characterizing and modeling user activity on smartphones: Summary. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '10, pages 375–376, New York, NY, USA, 2010. ACM.
[13] W. Song, N. Sung, B.-G. Chun, and J. Kim. Reducing energy consumption of smartphones using user-perceived response time analysis. In *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications*, HotMobile '14, pages 20:1–20:6, New York, NY, USA, 2014. ACM.