The 9th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2018) KaliGreen: A distributed Scheduler for Energy Saving

Hernán Álvarez-Valera, Philippe Roose, Marc Dalmau, Christina Herzog, Kyle Respicio

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2018) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

The 9th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2018)

KaliGreen: A distributed Scheduler for Energy Saving

Hernán H. Álvarez-Valera[a], Philippe Roose[a,*], Marc Dalmau[a], Christina Herzog[b], Kyle Respicio[a]

[a] *LIUPPA - Computer Science Laboratory - University of PAU, Allée du Parc Montaury, 64600 Anglet, France*
[b] *EFFICIT SAS, 6 route de Carpentier Cabirol, Mauzac, France*

Abstract

A commonplace issue with portable technology is battery efficiency. While many industries are trying their best to improve battery life without sacrificing a product's quality and efficiency, we believe that further can be done to improve battery consumption on one's mobile device—from tablets to smartphones to laptops to everything else. Many applications on these devices are based on a microservice architecture.

In this article, we introduce a new algorithm KaliGreen that can maneuver the microservices within a network of devices in order to maximize the run-time of a microservice-based application; moreover, KaliGreen allows a 54% increase in the average run-time of an application by shifting microservices from 6 devices (as example) with low battery or inefficient processing ratios to devices in better conditions. To achieve this, KaliGreen utilizes KaliMucho middleware, which is able manipulate microservices in run-time. This algorithm provides a plausible solution to maximizing energy consumption within a network of devices.

Keywords:
, Green Computing, Distributed applications, Microservices, Smartphones

1. Introduction

The importance of green computing has accelerated due to humanity's augmented consumption of energy. It's possible to define green computing as the study of computational methods aimed at minimizing required energy levels and the emission of co-pollution [15]. Pargman [13] and his colleagues point out that humanity consumes 50 % more of the resources that our planet can renew; thus, it is easy to believe that in a few years, out planet will be indefinitely unsustainable. In general, continuous heat from devices, waste from discarded devices, and the high level of energy

* Corresponding author. Tel.: +33 5 59 57 43 48
E-mail address: philippe.roose@iutbayonne.univ-pau.fr

that devices require immensely, degrades the environment and contributes to greater issues such as pollution and global warming[1] [2].

For this reason, many researchers are looking for new methods that minimize energy consumption in different levels of work such as the cloud, data centers and the single devices environments.

Furthermore, at the level of distributed applications, scientists and architects have designed new ways to implement an application, achieving reusability of functions, scalability, better parallelism, and better use of resources through modularity [7]. Service-oriented architecture is a way of conceiving an application as a set of services that can be used by multiple devices. Based on this model, the idea of conceiving an application as a set of microservices arises; in other words, each microservices runs on its own process and communicates with lightweight mechanisms [8].

We believe that it is possible to save energy in the typical devices of a common user such as smartphones, tablets, laptops, etc., by correctly organizing the interactive applications programmed based on microservices. KaliGreen is driven by this idea. It is a non-centralized scheduler with the objective of efficiently distributing microservices of an interactive application among the devices a common user. The basic idea is to allow a device with low battery, bad bandwidth, or slow processing speeds to request aid from other connected devices in order to shift energetically problematic microservices to a more efficient device.

The works related to this article will be explained in Section 2, then the Kaligreen architecture and algorithm will be explained in Section 3; the implementation and testing will be explained in Section 4 and we conclude in Section 5.

2. Related Works

In order to adequately understand the justification of their proposal, it is necessary to explain two fundamental concepts:

- How scientists try to save energy at different levels of work (Cloud, Grid and single devices).
- The tendency to use microservices instead of monolithic applications.

These two elements are detailed below.

2.1. The Green Perspective

In this paragraph, we will detail the three levels in which we believe that energy saving techniques are applied, after having studied the state of the art.

- To save energy at the cloud level, many authors focus on how to distribute the load around their data centers. However, some works like Hassan's and his colleagues [11] propose a higher degree of abstraction, in which energy saving can be understood as a set of services in the cloud. For a company that provides cloud services that is comprised of datacenters, there is
  - A service that provides green methods like guidelines, measurements, monitors and functionalities such as energy management, carbon emission of the set of datacenters.
  - A monitoring service, which verifies that the energy results obtained are always correct, so that new green services can be offered by the green service provider.

  Lastly, the set of datacenters will have new green methods and a permanent monitoring service to work in an energy efficient and non-polluting way.
- At the datacenter level, a good idea is to divide its hosts into clusters, taking into account the needs of users in the cloud. Then, the challenge is to locate or migrate the virtual machines that make up the environment in

[1] https://www.greenpeace.org/international/press-release/7612/smartphones-leaving-disastrous-environmental-footprint-warns-new-greenpeace-report/

[2] https://www.greenpeace.org/usa/research/from-smart-to-senseless-the-global-impact-of-ten-years-of-smartphones/

the correct hosts in such a way that some hosts can be suspended, so that others are alleviated from overload. Azmy, Nour M [1] and his colleagues apply this idea while considering SLA violations. Siddavatam and his colleagues [14] have a similar idea, but take into account processes and the thresholds of minimum and maximum use of the processor to save energy.

- At the level of a single device, scientists focused their efforts on reducing the amount of time and intensity in which a hardware component works. For example, at the CPU level (the component of greatest concern), it is possible to apply dynamic voltage and frequency scaling taking into account the type of process [9]. As for the hard drive, it is possible to apply a technique called "dynamic power management," which consists in putting a hard disk in a suspended state. Chen and his colleagues [2] predict disc accesses to apply this technique and make it possible the suspension time to be long enough to save energy. By evaluating other types of components, it is also possible to reduce energy consumption. For instance, by sharing the GPS location of a single device to other nearby devices, we can have a single GPS sensor for devices within a certain proximity [16].

Moreover, it is important to think about saving energy in common everyday devices. According to greenspace NGO, these devices represent a high degree of pollution in the world every day. Ideally, users should use the least amount of energy stored in batteries so that they save more energy and money while generating less heat and pollution. There are works that seek to deduct energy consumption by taking into account the routines [5] and the source code [10] of applications in mobile devices. The objective is to allow programmers and designers to make better decisions about code structuring and the implementation of certain application components; for example the more CPU an application uses and the longer an application component is executed, the more energy it spends [4].

Other works try to reduce the energy of mobile devices by making direct modifications to the operating system: less employment of the CPU and lower voltage levels. Some have tried manipulating the task scheduler by grouping processes according to categories for intelligent execution [3].

Many android developers have installed battery-friendly components to their applications to combat low levels of battery. Some techniques include putting the screen in black and white, lowering the brightness of the screen, closing dormant applications that consume too much energy. Following this idea, there are applications such as *ccleaner*[3], which detect high-consumption applications (regardless of the current state of the battery) and offer the user the option to close them.

However, there is not much literature that attempts to reduce energy expenditure from the intelligent distribution of services between mobile devices. Most focus on distributing processes between homogeneous devices [12] with fixed energy sources, using centralized scheduling algorithms. Nevertheless, E. Ilavarasan and colleagues [12] propose to execute a distributed application, abstracting it as a directed graph of dependencies and taking into account a set of mobile and heterogeneous devices. Their proposal is based on categorizing application nodes in similar devices, in terms of CPU, RAM, etc. in such a way that if one of the devices stops working, the process can be continued to executed on another similar one.

Thus, it is easy to think that there is a direct relationship between the number of devices turned on, the number of hardware components in use, and the level of saturation of said hardware components and the level of energy spent combined with the production of pollution. In our work, we will use the principles of the techniques described above, but oriented to microservices of an application. The microservices and their context will be explained in the following paragraph.

2.2. *The Microservices Architecture*

Following Paolo Di Francesco, microservices are small services that make up an application, each running in its own process and communicating with lightweight mechanisms [8]. With this concept, applications can be programmed in a modular way. Martin Fowler claims [4], for the following advantages:

[3] https://play.google.com/store/apps/details?id=com.piriform.ccleaner
[4] https://martinfowler.com/articles/microservices.html

- Easy updates via modules. If a user wants to update only one functionality of the application, then the user only needs to update the microservice that does this functionality.
- Decoupling and Cohesion. High cohesion and low coupling lead to a more stable and functional software program.
- Heterogeneity. This architecture facilitates application deployment between different types of devices and heterogeneous platforms.
- Decentralized storage operations. This adds more security and functionality overall.

With this, some companies have started to use this form of architecture for their applications (some may use this architecture without including the term microservices). Some examples are Amazon, Netflix or the UK Government Digital Service.

In this article, we will adopt the same definition of microservice as Paolo Di Francesco [8] explained, but in the context of an interactive application that works on a mobile device or on a desktop PC. We consider an application that can be understood as a set of small services that work together. A sample application could entail a game that includes one microservice that takes care of the calculation part (artificial intelligence of a character, for example), another microservice that focuses on graphical interface part, and another microservice for saving the player's progress on the cloud.

Up to this point, we have explained some of the different techniques for saving energy at different work levels (Cloud, Grid and a single device). All of them have shown that it is possible to decrease power consumption depending on how load balancing is performed between devices and how the new hardware features of shutdown or frequency reduction are used.

We have also explained the definition and advantages of using microservices in the development of distributed applications and how an interactive application for end users can be understood through microprocesses.

From this, we can now further explain KaliGreen.

3. The structure of KaliGreen

KaliGreen works across multiple devices; however, this network of devices is not guaranteed to be connected to other devices at any time (i.e. they are turned off, the connection signal is weak, etc.). This capricious environment explains the non-centralized nature of the algorithm. We believe that the natural way to distribute one or more microservices from an application in this type of environment is in a collaborative way. That is, each device offers or requests resources to others that are connected. So, KaliGreen, in general terms, is a non-centralized scheduling algorithm that makes it possible for a set of devices to intelligently exchange the microservices that make up their applications in such a way that the greatest amount of energy is saved, in particular for devices that depend on a battery. Broadly speaking, the following operations occur when a device is in an energy inefficient situation (for example, a cell phone that is about to finish its battery):

1. Find the microservice that consumes the most energy.
2. Extract the metadata of said microservice (CPU that it demands, size, network that uses, etc.) and then insert them into a data structure of a fixed size (which is sensible since the metadata fields will be the same throughout the algorithm) and that is naturally designed for the storage and exploration of small amounts of data. For these specifications, it is reasonable to utilize vectors.
3. Send the vector to all connected devices at that moment. The devices will evaluate if they can continue processing the microservice without any problem, ensuring that they are in an energy-safe situation. If a receiving device can handle the microservice, it will send a new vector back to the original device.
4. Evaluate existing candidates-devices and note which of them is the most energy efficient. This evaluation is done by analyzing the vectors that each candidate-device sends.
5. Move the microservice to the best candidate-device.

To achieve these operations, we have chosen Kalimucho middleware [6] because it allows to move / stop / start components of an application efficiently and transparently. Thereby, we propose that the devices of a user's daily life can interact with each other automatically such that the microservices that make up the applications are executed in the most appropriate place for saving energy, without harming the experience of the user. Thus, it is necessary that the application's microservices can be moved across devices, stopped or restored during ideal moments. In this Section the elements of the interaction achieved in KaliGreen will be explained. Then, in the final Subsection 3.6, the main algorithm will be explained in more detail. These elements are:

1. The Kalimucho middleware.
2. Set of devices of a user $D = [D_1...D_n]$.
3. The set of the user's applications $APPS = [App_1...App_n]$. Each one is composed of a set of microservices: $msAPP_i = [msApp_i...msApp_n]$.
4. The communication vector that is denoted V. Name V_i for each device D_i.
5. One monitoring microservice called $monDS = [monD_i...monD_n]$. This will find when a device is in a emergency energy situation.

3.1. Kalimucho Middleware

Da, Keling et al. [6] designed and programmed *Kalimucho*. This middleware reads and understands Java applications as a set of interconnected components installed in Android and PC devices. Each component is connected to others through special *connectors*. These connectors can be in active or inactive state. This means that a component can have many connectors with several other components without saturating the network since only the necessary connectors will be active.

Additionally, Kalimucho is able to move components from one device to another transparently without losing its state of execution. If a component requests to be moved to another device, Kalimucho will move it by shifting its java source code along with its current state of execution (special state storage files).

In KaliGreen, Kalimucho is the main engine that moves the microservices of a program to achieve an energy-safe load balance between user devices. In the following paragraph we will define user devices and their categories.

3.2. Devices of a User

According to several studies, like *Pew Research Center*[5] or *Newzoo's*[6] it has been determined that by 2016, at least 70% of people in countries like South Korea have a mobile device (Phones, tablets, etc). Then more than 80% of people in the world use mobile devices with android operating system [7].

Another study published by theverge[8] shows that there are one billon of personal computers running with windows, 100 million of active Mac users between desktops and smartphones and 1.6% of the desktops are performed by GNU Linux [9].

Thus, in this article, it is assumed that each user usually has devices such as a smartphone, a tablet, a desktop PC, a laptop, etc. Kaligreen uses Kalimucho, therefore manages components programmed in java, which are compatible with PC and Android.

Finally, as the first criterion of energy saving and sustainability, Kaligreen will differentiate user devices as follows:

- Battery-dependent devices: laptops, smartphones or tablets.
- Non-Battery-dependent devices: desktop computers.

[5] http://www.pewresearch.org/fact-tank/2017/01/12/evolution-of-technology/
[6] https://newzoo.com/
[7] https://www.idc.com/promo/smartphone-market-share/os
[8] https://www.theverge.com/2017/4/4/15176766/apple-microsoft-windows-10-vs-mac-users-figures-stats
[9] https://www.theverge.com/2014/9/21/6661393/tell-me-why-you-use-linux-everyday

Battery dependent devices will have higher priority to be freed from process load and lower priority to receive new processes (see section 3.6). The objective is to avoid battery deterioration and minimize the waste of energy.

3.3. Applications and Microservices

As we mentioned in Section 2.2, an interactive application can be composed of a set of microservices working together. One of the criteria that we apply in KaliGreen is how to categorize these microservices, in such a way that the user's natural interaction with their applications is not damaged. For us, an application may contain:

- Microservices that manage the graphical user interface (interactive part of the application): Kaligreen will not move this type of microservices, since it is impossible for us to know the will of the user.
- Microservices that manage the sensors of the devices: These microservices will not be moved by KaliGreen since we cannot remove these microservices in the user's devices without losing meaningful functionality.
- Microservices that control part of the applications: Here they enter to carve the microservices that perform calculations, for example, artificial intelligence, compression of images, etc. This type of microservices will be moved by Kaligreen among the user's active devices in order to save as much energy as possible.

Thus, Kaligreen will verify the type of operation that a microservice does before attempting to move it. Only this kind of microservices will be taken into account with the term $msAPP_i$.

3.4. The help request Vector

In order to achieve a non-centralized scheduler, it is necessary that each device has the ability to send relevant information to other devices in such a way that it can intelligently exchange information with them. In *Kaligreen* this information will be written in a vector called V. The objective is that each device D_i can build the vector V_i with the amount of resources (in terms of CPU, RAM, etc) that its heaviest microservice needs to work. Then, this vector will be sent to the others user's devices at important moments in order to try to move its represented microservice to another device that will execute it wasting less energy (It will be detailed in the next Section 3.6). Then, the devices can reach consensus and move all necessary microservices between them to achieve energy savings. Thus, for our proposal, the structure of the "Help Request Vector" will have the following information fields:

1. The unique identifier of the vector.
2. The identifier of the device D_i where it was created.
3. The average of CPU required in terms of Gigahertz.
4. The average of RAM required in terms of Mb.
5. The average of Storage Devices required in terms of Mb/S.
6. The GPS position in order to try share it by network and turn the sensor off, like *ErdOS* [16] do.
7. The screen resolution if the microservice wastes too much energy on it.

Finally, the entity responsible for choosing the heaviest microservice, building the vector that represents it and sending this vector to the other user devices is called "Monitoring Microservice," which will be detailed below.

3.5. The monitoring Microservice

To be able to start the scheduling algorithm explained in the next Section 3.6, it is necessary that each device has its own energy awareness, which will allow it to self-evaluate from the point of view of energy consumption and then be able to move or stop any of it's current microservices with the objective of lowering it's current energy consumption.

To achieve this goal, a microservice called *monitor* will be installed on each device (each monitor in each devices is called $monD_i$). Each $monD_i$ has the following objectives:

- Store the list of microservices ($msApp_i$) of each user applications that are currently running on the D_i device.

- Calculate and store the level of impact that each microservice has on the consumption of each device D_i. Then, organize them by order.
- Create and send the communication vector called V to all the other devices. It will be detailed in the next Section 3.4.
- Receive and store communication vectors from other candidates-devices V_i for every device D_i.
- Periodically evaluate the level of consumption of each device's components (CPU, RAM, Screen, Etc).

Each monitor $monD_i$ of each device D_i can create an information vector V_i with the metadata of the heaviest Application Microservice $msAPP_i$ in order to send it to connected devices. Then, a $monD_i$ will be able to receive and analyze every vector V_j that every candidate-device sends. Finally, the monitor $monD_i$ will select the best candidate device D_i and move the heaviest Microservice $msAPP_i$ to it. This operation will be detailed in the next paragraph *The Scheduling Algorithm*.

3.6. The Scheduling Algorithm

As mentioned in the Section 3, KaliGreen consists of a non-centralized algorithm for moving microservices between connected devices of a user; that is, there is no single device that executes this scheduling algorithm. Instead, each device will work collaboratively with the other user connected devices. With everything explained above, it is easy to deduce that there are two main participants in the scheduling algorithm: devices that want to move their microservices and the devices that wish to offer their resources to run microservices. We now detail the necessary algorithms for each of the participants.

3.6.1. Algorithm for devices that need support

We define that a device needs support when it enters one of the following situations:

- CPU overload. Taking into account our own experience, we have decided for this study that a CPU is in overload when the device is executing procedures at more than 80% [10] of its capacity.
- RAM overload. Taking into account our own experience, the RAM memory is in overload when it is full to more than 80% of its capacity.
- Network overload. Taking into account our own experience, the network is overloaded when data is being transmitted at a speed of more than 80% of the network component.
- Significant battery depletion. In android devices, a warning state is when battery is is below 15% and critical when it's below 5%.

This situation will be validated by a function called $isEmergencySituation()$.

Once the device enters a state of emergency, it is necessary to know which of the microservices should be moved, for this a function called $selectHeaviestMicroservice()$ will be invoked. With it, the microservice that has the greatest impact on the device will be chosen based on CPU use, RAM use, and network use. For us, the network is the most important component to analyze, then the CPU and finally the RAM memory. From observations we find energy consumption ratios of 6:3:1 respectively (there is no mathematical justification and will continue as our future works).

Once the microservice is selected, it is necessary to extract its metadata and write it in a vector, which will be sent to the other devices to be analyzed. This will be done in a function named $sendVector(V)$. Then, the device will wait a reasonable time to get candidate devices to move the microservice. This time, for us, will be $1ms$, like a normal *ping* operation do.

Then, it is necessary to know which is the best candidate device to send the heaviest microservice. For this experiment, the best candidate device is the first one that answer.

Finally, the chosen candidate device will be notified with the function $sendConfirmation()$, and then the heaviest microservice will be transferred over. Other candidate devices will also be notified that another device has been selected to said microservice via function $sendNegativeConfirmation(D_i)$.

[10] This value can be changed if needed

The complete algorithm is presented below.

Algorithm 1 Algorithm running in an emergency state device D_i

while $isEmergencySituation() == true$ **do**
  for all Microservices $msApp_i$ **do**
    $EvaluateEnergyImpact()$
  end for
  $msApp_h = selectHeaviestMicroservice()$
  build Vector V_h with metadata of $msApp_h$
  for all Devices connected D_j **do**
    $sendVector(V_h)$
  end for
  $WaitForCandidatsReponses()$
  for all candidates Devices $canD_j$ **do**
    $D_b = selectBestCandidate()$
  end for
  $sendConfirmation(D_h)$
  $ReceiveConfirmation(D_h)$
  for all Non selected candidates Devices $canD_u$ **do**
    $sendNegativeConfirmation(canD_u)$
  end for
  $moveMicroservice(msApp_h, D_b)$
end while

3.6.2. Algorithm for devices that offer support

We define that a device can offer help when it has sufficient computational capacity to run a microservice without putting itself in a state of emergency. Each user's device will have a list of devices that can assist at any given moment. The following algorithm will be executed:

Algorithm 2 Algorithm running in a offering help device D_i

while $isEmergencySituation() == false$ **do**
  $RecieveNewVectorFromDevice()$
  for all Vectors V_j of each needed support device D_j **do**
    $EvaluatePossibleEnergyImpact(V_j)$
  end for
  $V_c = selectHeaviestMicroserviceVector()$
  $sendConfirmation(D_c)$
  $receiveMicroservice(msApp_c)$
  if $isEmergencySituation() == true$ **then**
    $break$
  end if
end while

4. Implementation and Tests

To be able to test our algorithm, it is necessary to have an environment where several devices of a normal user can run microservices and move them. To achieve this with ease, we created a simulator utilizing the JAVA language that allows us to see the interaction of several devices in real time such that we can see the load level that each device has before and after running our algorithm.

This simulator mainly has the following entities:

- User. It is responsible for creating new devices (smartphone or PC), and turning them on and off. The user also has the ability to open new applications and create microservices for each application. Each user can turn on their devices at the same time with the button ”Start all devices.” This way all the devices can execute the algorithm at the same time (as it would happen in real life).
- Device. This represents a host that executes applications with its microservices. Each device shows the level of use of each component (CPU, NET and RAM) and its capacity. Then, each device shows its applications and its microservices in order to see the microservices movement from one device to another in real time. Finally, each device is able to update its state manually; for example, a smartphone may be able to put itself in a low-level energy situation.
- A Supervisor. This executes our proposed algorithm. It aims to monitor the current state of the device ad to deal with situations of overload or low battery. The supervisor requests and manages help vectors between the devices. The supervisor is also in charge of sending microservices and receiving them. This entity is in charge of simulating the operations carried out by Kalimucho.

With the simulator running, we created several amounts of devices with different amounts of microservices. The goal is to save the greatest amount of energy all devices use a battery in order to allow that applications run as long as possible. For us, we have assumed that when a device is using 100% of its CPU, RAM or NET capacity, its battery life is 2 hours. Thus, we can estimate the duration of execution of an application knowing the conditions of the devices.

From table 1, we see:

- The number of devices. We consider that is important to know how our algorithm works with the quantity of devices that a normal user has, so, we tested our algorithm from 1 to 6 devices.
- The number of microservices per device. It is important to know how our algorithm works, mainly when there are several microservices running on each device, causing several negotiation operations. We have testedup to 100 microservices running between all devices, with random resource costs.
- Average execution time of microservices before the algorithm. In each case, we evaluate how long the applications are executed, considering that the duration time of each device is directly related to its battery.
- Average execution time of microservices after the algorithm. We evaluate how long the applications are executed, after our algorithm is executed.

Number of Devices	MS per Dvice	Avg. Applications execution time before the Algorithm	Avg. Applications execution time after the Algorithm.
2	150	2.335	2.4
3	100	2,4	2,43
4	75	2.45	2.99
5	60	2,43	3.71
6	50	2.44	3.76

Table 1. Table of Experiments: The execution of Kaligreen in 1-6 devices with 100 MicroServices

5. Conclusions

In this article, we have presented KaliGreen.This is a middleware with a non-centralized conscious scheduling algorithm that monitors the energy levels of user’s devices with the objective of saving energy and battery in order to allow applications to run as long as possible.

In the continuous executions in our simulator, we saw with satisfaction that the devices that depend on the battery in conditions and overload of CPU or RAM or network were freed of load in an intelligent way, prioritizing devices with dangerous battery conditions. Thus, an application with a limited duration of time due to battery conditions, for example of 6 devices, can be redistributed and continue to run for 54% more of time on the other user devices if KaliGreen is launched.

6. Future Works

- One of the objectives of KaliGreen is not to affect the user experience when operating. In this article, it is considered that no matter how intelligent the heuristics are about knowing the behavior pattern of a person, there will always be dynamic situations in which he will be affected in efficiency and time. We consider future work, studying the best methods to move microservices without affecting the user experience.
- In order to select the most expensive microservice in energy, it is necessary to know its consumption ratio in terms of RAM, CPU and network. However, it is also necessary to find the proportional indexes of the consumption of each component (In the Section 3.6.1, rather than setting these values manually). A future work, is to find those coefficients dynamically depending on the hardware and operating system.

References

[1] Azmy, N.M., El-Maddah, I.A., Mohamed, H.K., 2016. Adaptive power panel of cloud computing controlling cloud power consumption, in: Proceedings of the 2Nd Africa and Middle East Conference on Software Engineering, ACM, New York, NY, USA. pp. 9–14. URL: http://doi.acm.org/10.1145/2944165.2944167, doi:10.1145/2944165.2944167.

[2] Chen, F., Zhang, X., 2008. Caching for bursts (c-burst): Let hard disks sleep well and work energetically, in: Proceedings of the 2008 International Symposium on Low Power Electronics & Design, ACM, New York, NY, USA. pp. 141–146. URL: http://doi.acm.org/10.1145/1393921.1393961, doi:10.1145/1393921.1393961.

[3] Corral, L., Georgiev, A.B., Janes, A., Kofler, S., 2015. Energy-aware performance evaluation of android custom kernels, in: Proceedings of the Fourth International Workshop on Green and Sustainable Software, IEEE Press, Piscataway, NJ, USA. pp. 1–7. URL: http://dl.acm.org/citation.cfm?id=2820158.2820160.

[4] Corral, L., Georgiev, A.B., Sillitti, A., Succi, G., 2014a. Can execution time describe accurately the energy consumption of mobile apps? an experiment in android, in: Proceedings of the 3rd International Workshop on Green and Sustainable Software, ACM, New York, NY, USA. pp. 31–37. URL: http://doi.acm.org/10.1145/2593743.2593748, doi:10.1145/2593743.2593748.

[5] Corral, L., Georgiev, A.B., Sillitti, A., Succi, G., 2014b. Method reallocation to reduce energy consumption: An implementation in android os, in: Proceedings of the 29th Annual ACM Symposium on Applied Computing, ACM, New York, NY, USA. pp. 1213–1218. URL: http://doi.acm.org/10.1145/2554850.2555064, doi:10.1145/2554850.2555064.

[6] Da, K., Dalmau, M., Roose, P., 2014. Kalimucho: Middleware for mobile applications, in: Proceedings of the 29th Annual ACM Symposium on Applied Computing, ACM, New York, NY, USA. pp. 413–419. URL: http://doi.acm.org/10.1145/2554850.2554883, doi:10.1145/2554850.2554883.

[7] Daugherty, P.R., 2009. The future of software architectures for large-scalebusiness solutions: Modularity, scalability, andseparation of concerns, in: Proceedings of the 8th ACM International Conference on Aspect-oriented Software Development, ACM, New York, NY, USA. pp. 1–2. URL: http://doi.acm.org/10.1145/1509239.1509241, doi:10.1145/1509239.1509241.

[8] Francesco, P.D., 2017. Architecting microservices, in: 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), pp. 224–229. doi:10.1109/ICSAW.2017.65.

[9] Ge, R., Feng, X., Sun, X.H., 2012. Sera-io: Integrating energy consciousness into parallel i/o middleware, in: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012), IEEE Computer Society, Washington, DC, USA. pp. 204–211. URL: https://doi.org/10.1109/CCGrid.2012.39, doi:10.1109/CCGrid.2012.39.

[10] Hao, S., Li, D., Halfond, W.G.J., Govindan, R., 2012. Estimating android applications' cpu energy usage via bytecode profiling, in: Proceedings of the First International Workshop on Green and Sustainable Software, IEEE Press, Piscataway, NJ, USA. pp. 1–7. URL: http://dl.acm.org/citation.cfm?id=2663779.2663780.

[11] Hassan, M.I., Bahsoon, R., 2014. Green-as-a-service (gaas) for cloud service provision operation, in: Proceedings of the 29th Annual ACM Symposium on Applied Computing, ACM, New York, NY, USA. pp. 1219–1220. URL: http://doi.acm.org/10.1145/2554850.2555182, doi:10.1145/2554850.2555182.

[12] Ilavarasan, E., Manoharan., R., 2010. High performance and energy efficient task scheduling algorithm for heterogeneous mobile computing system URL: https://airccj.org/csecfp/library/Search.php?title=high+performance+and+energy&x=20&y=8.

[13] Pargman, D., Raghavan, B., 2014. Rethinking sustainability in computing: From buzzword to non-negotiable limits, in: Proceedings of the 8th Nordic Conference on Human-Computer Interaction: Fun, Fast, Foundational, ACM, New York, NY, USA. pp. 638–647. URL: http://doi.acm.org/10.1145/2639189.2639228, doi:10.1145/2639189.2639228.

[14] Siddavatam, I., Johri, E., Patole, D., 2011. Optimization of load balancing algorithm for green it, in: Proceedings of the International Conference & Workshop on Emerging Trends in Technology, ACM, New York, NY, USA. pp. 1344–1346. URL: http://doi.acm.org/10.1145/1980022.1980321, doi:10.1145/1980022.1980321.

[15] Talebi, M., Way, T., 2009. Methods, metrics and motivation for a green computer science program, in: Proceedings of the 40th ACM Technical Symposium on Computer Science Education, ACM, New York, NY, USA. pp. 362–366. URL: http://doi.acm.org/10.1145/1508865.1508995, doi:10.1145/1508865.1508995.

[16] Vallina-Rodriguez, N., Crowcroft, J., 2011. Erdos: Achieving energy savings in mobile os, in: Proceedings of the Sixth International Workshop on MobiArch, ACM, New York, NY, USA. pp. 37–42. URL: http://doi.acm.org/10.1145/1999916.1999926, doi:10.1145/1999916.1999926.