Anole: A Case for Energy-Aware Mobile Application Design

Hui Chen, Bing Luo and Weisong Shi
Computer Science Department
Wayne State University
Detroit, United States
huichen, luobing, weisong@wayne.com

Abstract

Battery lifetime, which is one of the most significant user experiences for mobile devices, strongly restricts the functional design of hardware architecture and applications. Among all the aspects of energy saving for mobile devices, energy-aware application design is one of the main areas that has not yet been explored comprehensively. In this paper, we argue for the case for energy-aware mobile application design since there is pretty large space for energy saving on applications and we believe this is a promising area for future energy saving on mobile devices. To support energy-aware mobile application design, we propose a framework called *Anole*, aiming to add an energy adaptation layer by providing a set of APIs and adaptation policies. Our experiments with two applications on Android show that *Anole* is able to save a large amount of energy by triggering applications to change states accordingly. ***Keywords*-Anole; energy-aware; adaptation;**

I. Introduction

Accompanied with the prevalence of personal mobile devices, more and more applications are developed for those mobile devices, such as smart phones and tablets. At the same time, we notice that the battery lifetime for current mobile devices is a big challenge, limiting the usage of smart phone considerably. Researchers have proposed new ideas to extend the battery capacity; however, we do not expect the new battery technique will hit the market in the near future. How can we extend the battery lifetime from software perspective becomes a crucial issue to the systems research community.

Even though a lot of improvements have been made in the past [3], [4], [9], [10], [13], [14], [17], the low battery lifetime problem is still a main concern of our society. To further optimize the energy consumption for mobile devices, we propose the *Anole* framework for energy-aware mobile application design, since previous results [7], [8], [12] show that applications have large space for energy optimizations. The **rationale** is that application developers know much better about how to control the behavior of applications.

This paper has three contributions. First, we propose the notion of energy-aware mobile application design by introducing an energy-adaptation layer. Second, we design and implement *Anole* on Android platform with a set of APIs and two energy-aware policies: *service adaptation* and *hardware adaptation*. Last, we evaluate *Anole* using two case studies: web browser and video player, on Google Nexus One running Android 2.3.

The rest of this paper is organized as follows. In Section II, we describe several previous efforts that are most related with our work. We will give the design of the *Anole* framework in Section III. Section IV shows how we implement this framework on Android. In Section V, we evaluate the effectiveness of *Anole* with two typical applications. Finally, we draw our conclusion and discuss the future work in Section VII.

II. Related Work

In the last decade, energy management of mobile devices is a hot research topic in the system and architecture community. There are a large amount of related efforts [7], [13], [14], [1] having been done.

Flinn *et al.* found that the energy consumption of applications could be significantly influenced by the way how they worked [7], [8]. They argued that software behavior/energy consumption tradeoffs were as important as hardware behavior/energy consumption tradeoffs. In addition, Roy *et al.* also proposed there was large room to save energy by changing application behaviors [13].

Currently, energy-aware application design has not been globally studied; most researchers are trying to build a global energy management strategy. Several publications [5], [7], [15] tried to build evaluation platforms to analyze the energy requirement of applications. Flinn *et al.* built a platform called *PowerScope* to evaluate the energy consumption of program blocks [7]. Chang *et al.* used an energy-driven method to detect software hotspots [5]. These previous publications were used to provide energy-related suggestions to application designers. Different from their work, our *Anole* framework tries to tell application designers when should they change the energy consumption states of applications.

III. Anole Design

To support energy adaptation, we propose the *Anole* framework for mobile application designers and operating system designers. *Anole* supplies a set of APIs to trigger applications and system to change to different power states when certain energy events happen.

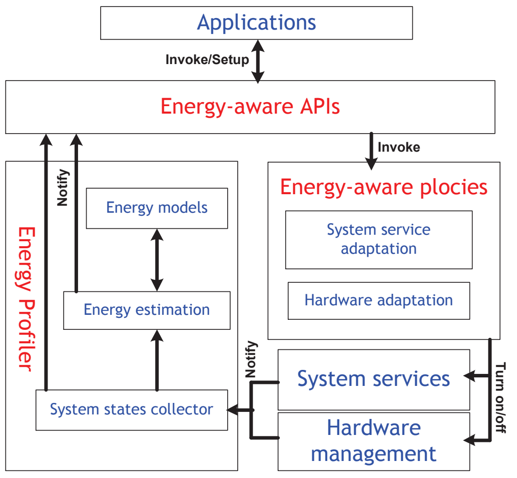

Figure 1. The Anole framework.

A. Energy Adaptation

Nowadays a common experience is that we can not use our mobile devices, such as tablets and smart phones, for more than 24 hours without charging. We attribute this limitation to the fact that energy consumption is rarely considered during the application design stage.

To this end, we propose energy adaptation in which application and system states should be dynamically changed based on the energy status and the user expectation. Energy adaptation is inspired by the idea of application adaption [11] , but with the focus on the bad user experience caused by the short battery life as more and more functions are added.

In addition, the idea of energy adaptation is in accordance with most users' habit. Usually, we want to keep our smart phones always on because telephone service and message service are still the two dominant functions.

B. The Anole Framework

As shown in Figure 1, the *Anole* framework includes three modules: energy profiler, energy-aware APIs and energy-aware policies. Energy profiler is a module that collects system states and estimates application energy consumption. Energy-aware APIs includes a set of interfaces for notifying applications and energy-aware policies by using the energy information. The concrete approaches to respond to the system energy adaptation should be designed by application developers in order to save energy consumption of their applications.

The energy-aware policy module executes the energy adaptation of operating system, which is also triggered by the energy-aware APIs. In this module we design two policies for the energy adaptation of operating system: *service adaptation* and *hardware adaptation*.

Finally, in our design we also highly consider user requirements. Therefore, we add several interfaces in the energy-aware APIs for the end users to setup the parameters of the *Anole* framework, such as adaptation point, which tells when the system should change to the adaptation mode, and adaptation step, which decides when the adaptation level will increase.

C. Energy Profiler

To support the energy-aware APIs, we implement an energy profiler module. The energy profiler module includes three parts: state collector, energy models and energy estimator. The state collector collects the states of the system and applications. Some of the collected information will be used to trigger energy adaptation directly. Other information will be used by energy estimator to estimate the energy consumption of applications, which tells the energy consumption information of each application. The application energy information is mainly used by fine-grained energy-aware APIs.

The method we use to estimate the energy consumption of applications is based on resource utilization. For more information on this, please see our previous work on pTop [6].

D. Energy-aware APIs

To support energy-aware mobile application design, we provide a set of APIs for application designers. Application designer should define several energy states and set the application to different energy states when one of a specific energy event is triggered. In this section we describe the functionality of these functions.

First, we define several functions for coarse-grained energy adaptation. These functions are triggered by system level events, such as battery level and battery temperature. Second, we define two fine-grained energy adaption functions, which are triggered by application energy consumption information. Finally, several functions are designed for end-users to setup the *Anole* framework. We implement several system-level setup functions, and leave application-level setup functions to be implemented by application developers. Several main functions are shown in Table I.

E. Energy-aware Policies

When the battery level is low, *Anole* will also notify operating system to adapt the energy status. We implement this function by adding the energy-aware policy module and some specific energy-aware policies to control the states of operating system. In this paper, we propose two basic energy-aware policies: *service adaptation* and *hardware adaptation*. Most importantly, we design this module with extensibility in mind. In this way, other researchers can easily add their customized energy-aware policies to the framework.

We observed that operating system services themselves generate a lot of activities even when the battery level is

Anole APIs	Description
`onBatteryLow(Level)`	Notify applications and policies to change energy consumption states.
`onPlugged(Plug/Unplug)`	Notify applications and policies to disable energy adaptation when the device is plugged or enable adaptation stage when the device is unplugged.
`onTemperatureHigh(Level)`	Notify applications when battery temperature is higher than a point.
`onBatteryHealth(Status)`	Notify applications when the battery is unhealthy.
`onEnergyConsumption(Level)`	Notify an application when the power consumption which is higher than a default value.
`requrestEnergy(Application)`	Request energy allocation for an application.
`onEnergyAllocationExhaust`	Notify applications when their allocated energy is exhausted.

Table I
THE ANOLE APIS FOR ENERGY-AWARE APPLICATION DESIGN.

low. This is one of the main reasons that causes the high idle energy consumption on the whole system, usually accountis more than 50% of the whole system energy consumption [2]. However, most of the services in the operating system is not necessary or critical. Thus, it is reasonable to stop them when the remaining battery energy is low. To implement this policy, we first classify the system services into three categories: *highly required*, *required* and *optional*. Then we define the policy: which services should be stopped when the system enters into an energy adaptation level. Dependencies between services are not considered, and a service should be stopped will be kept on only when the foreground application relies on it.

The idea of *hardware adaptation* is similar to *service adaptation*. When the remaining battery energy is lower than a threshold, we will directly control the state of devices. To design this policy, we need to know the usage pattern of each hardware and the power profile. Usage pattern is important for deciding which application should be closed on an energy adaptation level. Finally, a group of schemas are defined to set up this policy when the system enters into an energy adaptation level. End users can change the threshold value and the schemas of each level.

IV. IMPLEMENTATION

We implement the *Anole* framework on Android 2.3, which is open source to the society. The implementation mainly includes four parts: pTop service, *Anole* service, energy-aware policy service and application energy-aware support.

A. pTop Service

Energy profiler is implemented as a system service called pTop service, which has three functions: collecting system information, estimating application energy consumption, managing energy consumption information. The *Anole* service will be notified when the system states change or new energy information is available. The communication of pTop service and *Anole* service uses `Intent` messaging, which is a facility for late run-time binding between components, supplied by Android system.

The pTop service needs to maintain system states and application energy consumption information. We save these data into a hash table data structure. For application energy consumption, we only save the total energy consumption and the estimated energy consumption history in a time interval because the memory space is limited. Also, when an application is closing, the corresponding history energy consumption information in hash table will be deleted to save memory.

B. Anole Service

The *Anole* service, which is the core of this framework, triggers applications and operating system to change states. For each energy-aware option, we define a specific `Intent` message. Components that had registered this message will be notified when the *Anole* service generates one of this kind of message. These components includes application manager, service manager of Android system. Whenever it receives an system states change information from the pTop service, it will check which `Intent` message should be generated. These `Intent` messages are in accordance with the interfaces we define in in Section III.

Seven energy adaptation levels from level 1 to level 7 are used in the implementation. A higher level number means more actions should be taken to decrease the energy consumption. This service could dynamically map these seven levels onto user defined energy adaptation point and energy adaptation step. Energy adaptation point is a value that refer to the percent of remaining battery energy to trigger the whole system enters into energy adaptation mode. Energy adaptation step is a value that decides how many percent of energy drop will increase the energy adaptation level.

Finally, we define all the functions into a uniform energy adaptation interface object with some predefined constants, such as energy levels. This interface object is implemented by the super class of the `Activity` object, which manages the life cycle of applications on Android. Energy-aware policies are also required to implement this interface object.

C. Energy-aware Policy Service

We modify the service manager class, which registers all the energy-aware `Intent` messages, to support the design of energy-aware policies. Each policy is defined as a service of Android. Different with other normal services, these services must implement the energy adaptation interface. Then, service manager could invoke a specific method of an energy-aware policy when an `Intent` message was received by the service manager.

The service adaptation policy will stop some system services when energy adaptation enters into a level. In our design, we simply classify system services into three categories: *highly required*, *required* and *optional*. The default energy adaptation level to stop optional and required services are 5 and 2. Currently, we classify system services and define the default energy levels based our personal experience.

Similar to service adaptation policy, hardware adaptation policy is also implemented as a service that implements the energy adaptation interface. Also, we predefine the energy adaptation levels, at which a hardware device should be closed. For example, the bluetooth service will be closed when the energy adaptation level is 1 and network service will be closed when energy adaptation level is 6.

D. Application Energy-aware Support

To support application energy adaptation, we modify the `Activity` class of Android system, which must be extended by all the applications, by implementing the energy adaptation interface. Also, we modify this class to listen the `Intent` messages generated by the *Anole* service. When it receives an energy-aware message, the related function of each application will be invoked to change the energy consumption state of this application. By default, if the method is not overridden by the application designer, we will take some default energy-saving strategies at different energy adaptation level.

V. Evaluation

To evaluate the effectiveness of our work, we generate an Android image with our implementation for Google Nexus One and do a group of experiments on this platform.

A. Method

We use a Nexus One smart phone, the processor of which supports 12 power steps, and a RadioShack digital multimeter to measure the energy consumption of the system. First, we connect a resistor, about 0.35 Ohm, between battery and the mobile phone. Then, we measure the voltage of the resistor. Finally, we can compute the power consumption of the whole system with the voltage measured.

To evaluate the effective of application energy adaptation, we use two applications, video player and web browser, to do the experiment. First, we define a simple energy adaptation strategy for each of them. Then, we implement these strategies by overriding onBatteryLow method in all these two applications. When we evaluate on the applications, we close all the other applications.

In addition, when we evaluate the effectiveness of energy-aware policies, we change the system into energy adaptation mode by hand in order to avoid the long time waiting. Since, we only want to see whether the energy requirement drop or not, it make sense for us to do in this way. We use the measured system power to do analysis. To do this, we need to assume the power is stable in the sampling interval, which is 1 second. And then, we compute the energy consumption by accumulate the energy consumption of each sampling interval.

B. Case Studies

To demonstrate the feasibility of our approach, we did a group of experiments with the two applications that we modified. Energy-aware application design requires application designers to make decision to change application behavior based on system energy status. Our results show that there is an average of 30% potential to save energy wasted by applications and operating systems.

1) Video Player: We modify the MediaPlayer framework of the Android platform. For video player, we change the resolution of the video when energy adaptation is triggered by changing the number of frames played per-second. This will decrease the computation required for video data processing. We use three short videos to test. From Figure 2 we can see that this method is effective to decrease the energy consumption. On average, about 27.2% of energy could be saved.

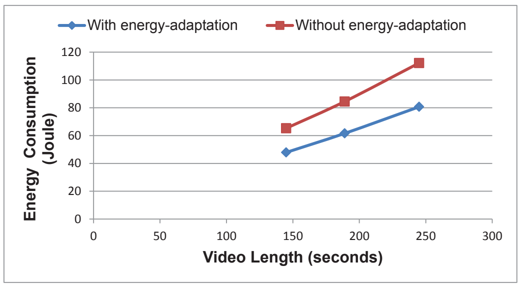

Figure 2. The energy consumption of video player when use energy adaptation and without energy adaptation.

Figure 3 shows the active power of the system when the system is entering into energy adaptation mode. In this figure, we see clearly that the power dissipation of the system drops when the application starts to execute energy adaptation. On average, there is a drop of about 40 mw on power.

2) Web Browser: We change the web browser application, which is supplied with the Android source code, and the WebCore library. In this application, an image is downloaded only when a direct user request is received. This is close to human behavior because we are usually only interested in several pictures of a web page. In the experiment, the web browser automatically open a group of web pages one by one (after the past one is fully loaded). Then we compare the energy consumption both with energy adaptation and

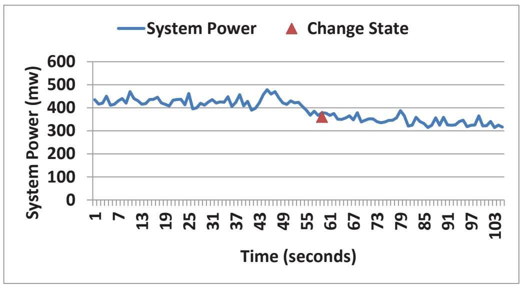

Figure 3. The active power of the system when video player changes to energy adaptation mode.

without energy adaptation. The result is shown in Figure 4, from which we can see that the case of using energy adaptation consumes much less energy than the other case. This is because image downloading requires a large amount of wireless communication. In this example, about 53.4% percent of energy could be saved.

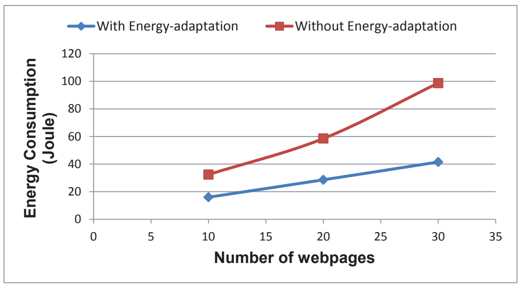

Figure 4. The energy consumption of web browser when use energy adaptation and without energy adaptation.

Even though we cannot see a a clearly drop of system power when we did experiment for web browser. The time used for loading web content is decreased significantly because the downloading of images account for a large amount of network traffic.

C. Energy-aware Policy

In addition to using applications, we also use two energy-aware policies to evaluate the framework. When we did experiment on these two applications, we enable one policy.

1) System Service Adaptation: System service adaptation policy stops several background services of the system when energy adaptation mode is entered. In this experiment, we measure the change of system power when the system change from normal status to energy adaptation status. In this stage, several optional services, such as alarm manager service, bluetooth service, clipboard service, mount service, backup manager service and wallpaper manager service will be closed. The experiment is shown as Figure 5, in which we can clearly see the drop of power.

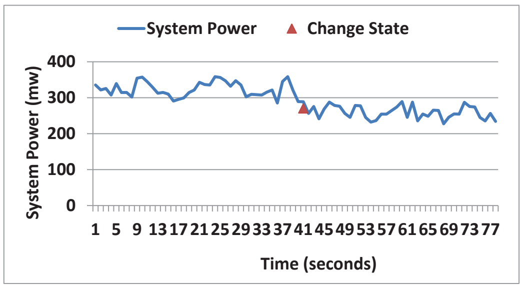

Figure 5. The active power of the system when service energy adaptation is triggered.

2) Hardware Adaptation: As we have mentioned before, the hardware adaptation policy turns off hardware device directly when system change to different energy adaptation levels. In our experiment, we measure the power of the system during 2 energy adaptation changes. In the first change, the system enters into energy adaptation mode, and we turn off GPS, Bluetooth, and accelerometer. Also, we decrease the brightness of the screen. In the second change, we mainly turn off the wireless network. As we could see from Figure 6, the active power of system clearly shows three stages.

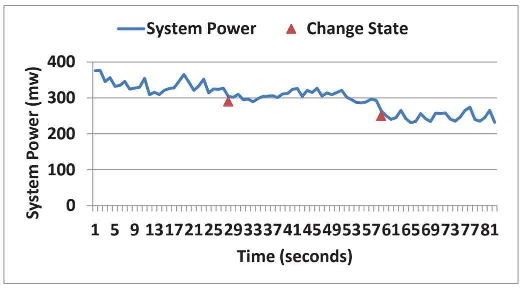

Figure 6. The active power of the system when hardware adaptation is triggered.

Experiment results not only show that these two policies can effectively decrease the energy consumption, but also verify that *Anole* is capable of correctly changing the system into different working modes.

VI. Future Work

In this paper, we investigated enabling technologies for energy-aware mobile application design. Our long term objective is to extend the battery life of mobile devices significantly via integrated cross-layer approaches. There are a lot of future works need to do enable to complete the function of energy adaptation. In this section, we talk about the future works in this area.

1) *User behaviors.* Research on user behaviors is highly required for us to design the default energy-aware policies. In the future, how users setup energy adaptation parameters and how will they apply energy adaptation

level to each services and hardwares should be studied. Also, user experience is highly required for application designers to make up energy adaptation strategies.

2) *Energy-aware policies.* More energy-aware policies should be designed to support energy adaptation. First of all, we need to design an energy-aware scheduling algorithm, which has two objectives: controlling the energy consumption of applications that did not implemented the energy-aware APIs and saving energy while at the same time maintain a good user experience. Also, is is helpful to design a synchronized packet scheduling algorithm to optimize the energy consumption of wireless communication. The polices should not only be good at decreasing energy consumption but also considering the difference of each application.
3) *Analyze application usage patterns*. Application usage pattern is helpful for design user-oriented energy-aware policies. The difficult part are pattern representation and pattern recognition. We could leverage SPAN [16] and data mining techniques to recognize the patterns of applications in terms of their energy consumption. The difference between application usage pattern-based energy-aware policies and other existing energy-aware policies is that user experience and energy saving are considered at the same time.

VII. Conclusion

In this paper, we propose energy adaptation and implement it as the *Anole* framework on Android platform. We evaluate our framework with two applications, the result shows an average energy-saving space about 30%. Also, the experiment shows that the *Anole* framework effectively supplies energy adaptation support to the operating system. The experiments on applications and energy-aware policies clearly show that Anole could trigger energy adaptation and save energy.

Of course, our method will influence user experience of applications, especially those energy-heavy applications. But, we noticed that a lot of functions are infect not always used for most of time. However, we still run background services and turn on related hardware to support these functions. The is the main reason that cause the energy consumption because a large amount of activities will be generated by them.

Acknowledgements

Weisong Shi is in part supported by the Introduction of Innovative R&D team program of Guangdong Province (NO. 201001D0104726115) and NSF grants CCF-0643521 and NSF CNS-1205338.

References

[1] Manish Anand, Edmund B. Nightingale, and Jason Flinn. Ghosts in the machine: interfaces for better power management. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, MobiSys '04, pages 23–35, New York, NY, USA, 2004. ACM.

[2] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40:33–37, December 2007.

[3] Pat Bohrer, Elmootazbellah N. Elnozahy, Tom Keller, Michael Kistler, Charles Lefurgy, Chandler McDowell, and Ram Rajamony. *The case for power management in web servers*, pages 261–289. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[4] Thomas D. Burd, Trevor A. Pering, Anthony J. Stratakos, and Robert W. Brodersen. A dynamic voltage scaled microprocessor system. *IEEE Journal of Solid-State Circuits*, 35:1571–1580, 2000.

[5] Fay Chang, Keith Farkas, and Parthasarathy Ranganathan. Energy-driven statistical sampling: Detecting software hotspots. In Babak Falsafi and T. Vijaykumar, editors, *Power-Aware Computer Systems*, volume 2325 of *Lecture Notes in Computer Science*, pages 105–108. Springer Berlin/Heidelberg, 2003.

[6] Thanh Do, Suhib Rawshdeh, and Weisong Shi. ptop: A process-level power profiling tool. In *Proceedings of the 2nd Workshop on Power Aware Computing and Systems (HotPower'09)*, oct 2009.

[7] Jason Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *WMCSA '99: Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, page 2, Washington, DC, USA, 1999. IEEE Computer Society.

[8] Jason Flinn and M. Satyanarayanan. Managing battery lifetime with energy-aware adaptation. *ACM Trans. Comput. Syst.*, 22:137–179, May 2004.

[9] Jerry Frenkil. Tools and methodologies for low power design. In *Proceedings of the 34th annual Design Automation Conference*, DAC '97, pages 76–81, New York, NY, USA, 1997. ACM.

[10] Andreas Merkel and Frank Bellosa. Balancing power consumption in multiprocessor systems. *SIGOPS Oper. Syst. Rev.*, 40:403–414, April 2006.

[11] Brian Noble, M. Satyanarayanan, and Morgan Price. A programming interface for application-aware adaptation in mobile computing. In *Proceedings of the 2nd Symposium on Mobile and Location-Independent Computing*, pages 57–66, Berkeley, CA, USA, 1995. USENIX Association.

[12] Michael D. Powell, Arijit Biswas, Joel S. Emer, Shubhendu S. Mukherjee, Basit R. Sheikh, and Shrirang Yardi. Camp: A technique to estimate per-structure power at run-time using a few simple parameters. In *IEEE 15th International Symposium on High Performance Computer Architecture, 2009. HPCA*, pages 289–300, 2009.

[13] Arjun Roy, Stephen M. Rumble, Ryan Stutsman, Philip Levis, David Mazières, and Nickolai Zeldovich. Energy management in mobile devices with the cinder operating system. In *Proceedings of the sixth conference on Computer systems*, EuroSys ’11, pages 139–152, New York, NY, USA, 2011. ACM.

[14] David C. Snowdon, Etienne Le Sueur, Stefan M. Petters, and Gernot Heiser. Koala: a platform for os-level power management. In *Proceedings of the 4th ACM European conference on Computer systems*, EuroSys ’09, pages 289–302, New York, NY, USA, 2009. ACM.

[15] Shinan Wang, Hui Chen, and Weisong Shi. Span: A software power analyzer for multicore computer systems. *Elsevier Sustainable Computing: Informatics and Systems*, page In press, 2011.

[16] Shinan Wang, Hui Chen, and Weisong Shi. SPAN: A software power analyzer for multicore computer systems. *Sustainable Computing: Informatics and Systems*, 1(1):23 – 34, 2011.

[17] Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat. Ecosystem: managing energy as a first class operating system resource. *SIGPLAN Not.*, 37(10):123–132, 2002.