Journal of Systems Architecture 59 (2013) 794–800

Contents lists available at SciVerse ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Virtual Battery: A testing tool for power-aware software

Youngjoo Woo [a], Seon Yeong Park [b], Euiseong Seo [a,*]

[a] Sungkyunkwan University, Suwon 440 746, Republic of Korea
[b] Korea Advanced Institute of Science and Technology, Daejeon 305 701, Republic of Korea

ARTICLE INFO

Article history:
Received 15 November 2012
Received in revised form 20 June 2013
Accepted 26 June 2013
Available online 5 July 2013

Keywords:
Battery management
Emulation
Software testing
Virtualization
Virtual machines

ABSTRACT

Virtualization is an inexpensive and convenient method for setting up software test environments. Thus it is being widely used as a test tool for software products requiring high reliability such as mission critical cyber-physical systems. However, existing virtualization platforms do not fully virtualize the battery subsystem. Therefore, it is difficult to test battery-related features of guest systems. In this paper, we propose Virtual Battery, a battery virtualization scheme for type II full virtualization platforms. Virtual Battery takes the form of an ACPI-compatible battery device driver dedicated to each virtual machine, which virtualizes a target system. Through Virtual Battery, developers can easily manipulate the charging and battery status of each virtual machine (VM), regardless of the existence or current status of the host system's battery. In addition, Virtual Battery emulates the behavior of batteries by discharging the virtual batteries according to the resource usages of their VMs. This feature enables VMs to act as battery resource containers. Three case studies demonstrate the effectiveness of the proposed scheme.

1. Introduction

To extend battery lifetime and improve the user experience, most modern operating systems (OSs) employ low-power operating modes that change diverse system behaviors according to battery status, such as the charging/discharging condition and remaining energy. In addition, to protect user and critical system data, most systems automatically save their memory contents to permanent storage devices, suspending their operations until the battery systems are recharged. This is called hibernation. As the importance of battery management in mobile cyber-physical systems is ever increasing, such battery-related software components are becoming more powerful, complicated, and error-prone.

Although many debuggers and profilers for application development have been introduced, there are relatively few tools for system software development; therefore, testing and resuscitating specific behaviors exhibited by system software or device drivers remain difficult problems.

Emerging virtualization technology significantly relieves this difficulty. By making target systems run on virtual machines (VMs) and by manipulating these VMs, developers can easily observe the behavioral characteristics of target systems as they react to hardware status changes [1,2]. In addition, virtualized target systems enable developers to run multiple target systems or to have different target OSs on a single development workstation at the same time.

For a hardware component to be used in a VM, the virtualization platform must provide the component to the VM as a virtual device. For example, a battery subsystem has to be virtualized and provided to VMs if developers want their VMs to have a battery subsystem. However, despite the ever-increasing popularity of battery-powered mobile systems, rechargeable batteries are not fully virtualized on almost all commodity virtualization platforms, including QEMU [3], Xen [4] and VirtualBox [5]. These platforms typically provide only bridged battery device drivers that just reveal the host system battery status to VMs.

The lack of battery virtualization or emulation hinders system software developers from testing their battery-related features, such as low-power operating modes, hibernation, system status backup, and energy-aware background job scheduling. For example, if the battery status information provided to VMs is directly transferred from the status information of the host system's battery, as in Xen or VirtualBox, repeated cycles of discharging host system's battery and recharging it are required to confirm proper hibernation of the virtualized testing system. Even worse, if the host system is not battery-powered, developers will have no means to test battery-related features.

A battery emulation layer for VMs is also useful for end users. Besides the enormous success of virtualization in servers, laptops and other mobile computers are currently using virtualization technology to allow users to run different OSs in a device or to sandbox unverified applications. In such cases, VMs act as resource containers that can precisely control the resource usage of

* Corresponding author. Tel.: +82 10 8792 7678.
E-mail addresses: mongmio@csl.skku.edu (Y. Woo), parksy@calab.kaist.ac.kr (S.Y. Park), euiseong@skku.edu (E. Seo).

1383-7621/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.sysarc.2013.06.006

virtualized systems. However, current VMs have no control over energy usage [6]. Thus, VMs in mobile devices also require a battery emulation layer that provides VMs with independent energy containers.

In this paper, we introduce Virtual Battery, a battery virtualization scheme for type II full virtualization platforms that allows each VM to have a separate and independent battery that can be manipulated.

The battery emulation layer in Virtual Battery delivers hypothetical information regarding battery capacity, current status, and more to the VM. Each VM receives this set of information on its own, and each Virtual Battery can have different data from another. Consequently, VMs appear as independent battery-powered systems. The suggested Virtual Battery scheme also includes a discharger component that gradually drains Virtual Batteries in proportion to processor utilization by their corresponding VMs.

The remainder of the paper is organized as follows. We propose the battery virtualization scheme in Section 2 and present two examples in Section 3. After describing related work in Section 4, we conclude in Section 5.

2. Our approach

Virtualization technology can be categorized as type I or type II [7] virtualization, as illustrated in Fig. 1 In type I virtualization, the VM monitor (VMM) is located on bare-metal hardware. VMs run on the VMM and access the physical hardware through the VMM. No OSs are required to run the VMM. By contrast, the VMM in type II virtualization runs on a conventional OS, which sees the VMM as a normal application. VMs run through the VMM and the system access requests from the VMs pass through the VMM to the underlying OS. The OS actually deals with the requests on behalf of the VMs.

Since the type I approach has less overhead than type II, type I virtualization technology is currently the de facto virtualization standard for cloud computing and server consolidation, whereas workstation, PC and laptop users favor type II virtualization because the roles of VMs in those systems are subsidiary to the main OS or system that the VMM runs on. In this research, we focus on a battery emulation layer for type II virtualization.

Most modern OSs monitor and control power sources through the advanced configuration and power interface (ACPI) [8]. ACPI is an interface specification that facilitates power management of both peripheral devices and entire systems. ACPI includes standards for power management and configuration of hardware components, plug-and-play devices, and diverse system event handling. Battery management is also a function provided by ACPI.

An ACPI-compatible battery system has an embedded controller interface that communicates with a standard ACPI driver [9]. The ACPI driver collects a set of information, including the designed capacity, latest fully charged capacity, and remaining charge of the battery system. The driver also provides interfaces to adjust low-battery warning trip points through the controller so that the controller signals when the battery discharges below that point. In the case of Linux, the standard ACPI driver collects current values for capacity, voltage, temperature, and many other measurements. This information is regularly updated in human-readable files in the *proc* file system.

Virtual Battery, our battery emulation layer, is designed to be ACPI-compliant so that any OS with ACPI battery drivers can easily accommodate it. In addition, Virtual Battery is targeted for full virtualization platforms so that guest OSs do not need to be modified to use it. To satisfy these requirements, the prototype is designed to fit into VirtualBox [5], which is an open-source, type II full virtualization platform.

In general, VMMs are responsible for providing virtual I/O devices to VMs. There are two approaches to virtualization of I/O devices: a virtual device can be implemented either as an emulation of a physical device or as a direct pass-through of a physical device. While VMMs employ both of these approaches, depending on the device characteristics, ACPI battery drivers are currently implemented for direct pass-through of selective battery status information to the host system batteries.

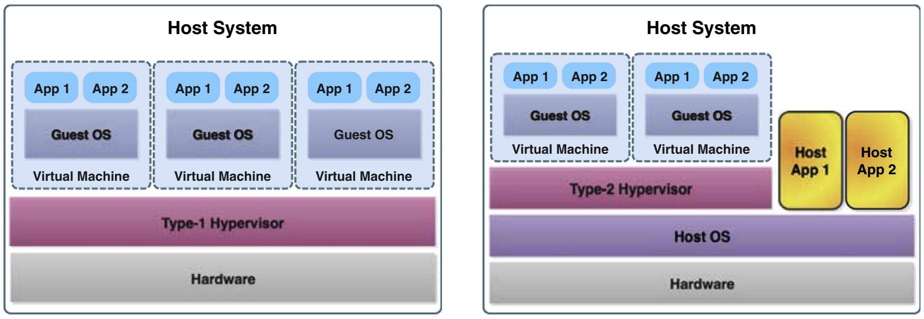
As illustrated in Fig. 2(a), the ACPI driver of VirtualBox is implemented as two separate drivers: a front-end driver and a back-end driver. The front-end driver is embedded in the VM code, while the back-end driver is located in the VMM code. The front-end driver merely transfers I/O requests to the back-end driver, which actually handles the requests in the host system. The ACPI drivers of guest OSs recognize the front-end driver as a standard ACPI controller.

Because the ACPI driver simply transfers status information for the host system's battery, all VMs always share the same information. To provide independent battery status to each VM, the back-end driver manages per-VM battery status information as illustrated in Fig. 2(b).

A part of the Virtual Battery emulation layer is also included in the back-end driver. This creates a Virtual Battery file that contains information about fully charged capacity, present remaining capacity, current voltage, and many other measurements when a VM is initialized. Each VM has its own Virtual Battery file. As a result, the battery status of a VM no longer depends on either the host battery or the batteries of other VMs.

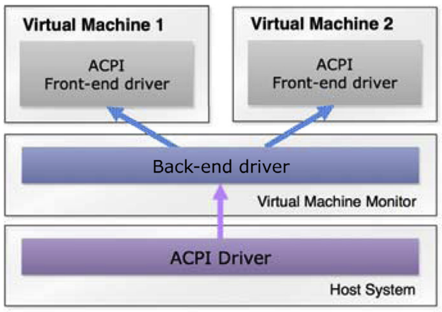
The function provided by the battery emulation layer is relatively simple in comparison to other types of devices. The battery status is changed not in response to OSs, but by user behavior, such as connecting the system to an AC power source.

Therefore, the back-end Virtual Battery device driver only handles inquiries for battery status. While the original back-end driver regularly polls host battery information through the host ACPI

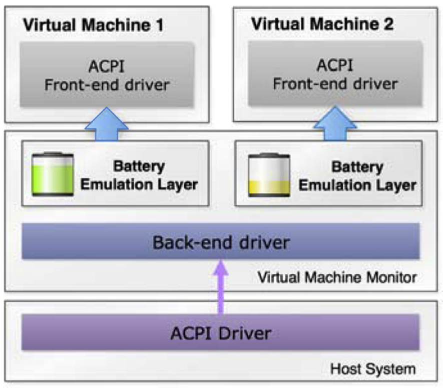


(a) Type I virtualization architecture (b) Type II virtualization architecture

Fig. 1. Two approaches to virtualization.



(a) Original VirtualBox ACPI driver structure



(b) Virtual Battery ACPI driver structure

Fig. 2. Virtual Battery provides separate battery emulation to each virtual machine, independently from the underlying host ACPI driver.

driver, the Virtual Battery back-end driver regularly fetches virtual battery information from the Virtual Battery status files.

Besides the front- and back-end device drivers and the battery emulation layer, Virtual Battery has two more components, the discharger and *Virtual Battery manager*, as shown in Fig. 3. The discharger is a thread in charge of mimicking battery drainage due to system activities. It decreases the remaining charge of a VM battery in proportion to its allocated processing time. Virtual Battery manager is a GUI application that displays and manipulates the status of Virtual Batteries.

The back-end ACPI driver has a thread to manage the battery status information for a VM. A VM instance in the type II virtualization scheme is commonly implemented as a process on the host system. Thus, in our approach the threads identify VMs by their process IDs (PIDs).

The Virtual Battery status files have the same format as the original battery status files in the *proc* file system. However, to identify information from different VMs, the file names contain the PID of the corresponding VM. The battery emulation layer finds the Virtual Battery status information files assigned to a specific VM by using its PID. Status information is updated by both the Virtual Battery manager and the discharger threads.

Users monitor and control the Virtual Battery status through the Virtual Battery manager. Fig. 4 shows a screen shot of the Virtual Battery manager GUI. This application can show the status of up to four Virtual Batteries at once. On the screen, remaining capacity is displayed in terms of mWh, which is the unit of battery charge. The GUI also shows the ratio of remaining charge to battery capacity as a horizontal bar graph. The charging status is shown as a pair of radio buttons. The battery capacity and the current charge

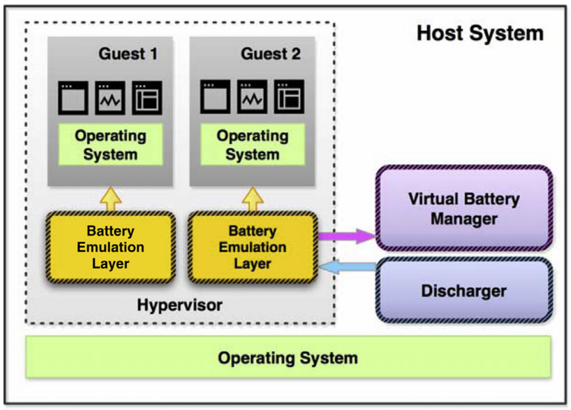


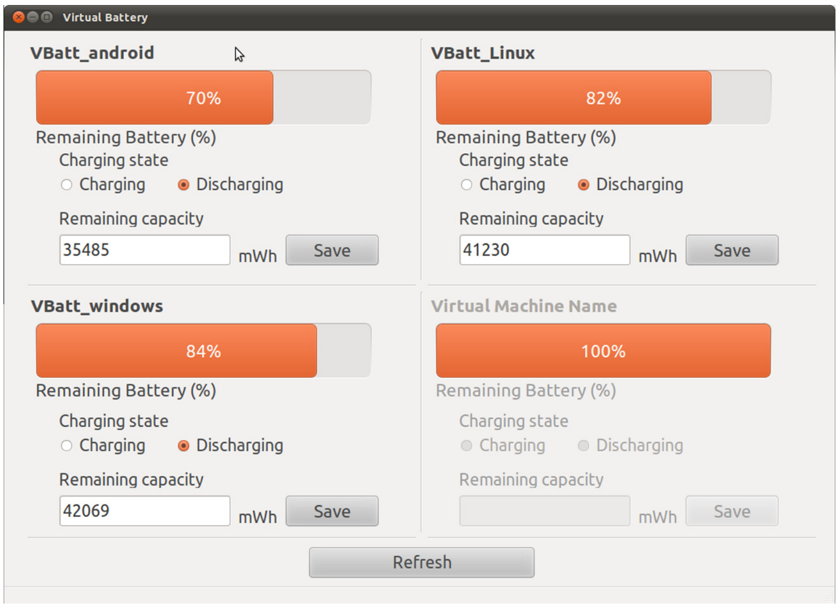
Fig. 3. Virtual Battery consists of three components: the Virtual Battery device layer, a management application, and a discharger thread.

of a hypothetical battery are configured simply by entering desired numbers. The battery charging status can also be set by clicking the radio buttons. When a user requests status changes through the manager, the manager writes the updated information over the applicable files. Then the updated information is collected by the back-end ACPI driver. The configuration and information update operations are regularly conducted every second.

In actual systems, batteries discharge in proportion to system energy consumption. Virtual Battery mimics battery discharge through the discharger thread. Fig. 5 shows the workflow for the discharger. A discharger thread is periodically activated and collects resource usage information for the corresponding VM from the *proc* file system. Based on the underlying energy consumption model and the amount of resources used in the last time interval, the discharger calculates the amount of energy to be deducted from the current charge value of the virtual battery. It updates the remaining charge in the status file and returns to the sleep state. The activation interval is by default set to be 1 s. However, the activation interval is easily configured with a parameter.

Currently, our model simply multiplies processor time by a user-defined constant. Because each VM runs as a task in the host system, the host system keeps track of the resource usage information for each VM in the *proc* file system, which is easily accessible from the discharger thread. Although in this model batteries seem to gradually discharge over time, its accuracy is far from that required for battery simulation or to profile the energy usage of a VM because the usable battery capacity heavily depends on diverse parameters including the discharge rate and system load [10]. However, modeling battery characteristics is beyond the scope of our research and capability because accurate modeling of battery behavior requires intense knowledge in electrical and chemical engineering. We believe that existing research results for estimating VM-level energy consumption [6,11] and for modeling battery characteristics can be easily incorporated into the discharger since the discharging and battery modeling component of our prototype is implemented as an independent module so that the proposed layer operates as a platform for applying the diverse existing battery models when it is necessary.

The front- and back-end drivers use processor time only when the guest OSs request battery status information or notification events occur due to status changes or low battery levels. Because these conditions rarely occur and the complexity of both drivers is as simple as in conventional bridge drivers, their impact on performance is negligible. The discharger thread is the only component that requires additional computing resource since it wakes up and updates the battery status files every 10 s or once in a user-defined interval. However, in comparison to the processor cycles that a VM consumes, the additional processor time for the VM


Virtual Battery

VBatt_android
70%
Remaining Battery (%)
Charging state
Charging [✓ Discharging]
Remaining capacity
35485 mWh Save

VBatt_Linux
82%
Remaining Battery (%)
Charging state
Charging [✓ Discharging]
Remaining capacity
41230 mWh Save

VBatt_windows
84%
Remaining Battery (%)
Charging state
Charging [✓ Discharging]
Remaining capacity
42069 mWh Save

Virtual Machine Name
100%
Remaining Battery (%)
Charging state
Charging Discharging
Remaining capacity
mWh Save

Refresh

Fig. 4. The Virtual Battery manager displays information for up to four virtual batteries at the same time.

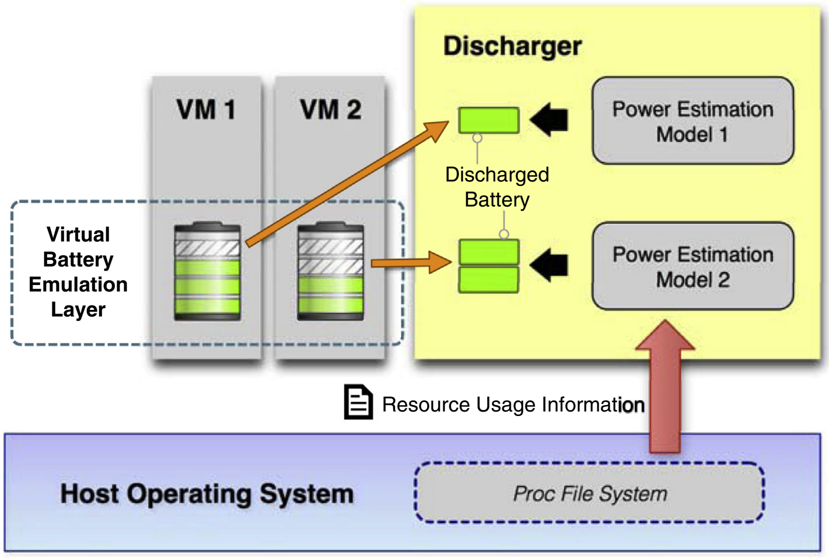


Fig. 5. The discharger periodically determines the amount of energy to be deducted from the virtual batteries based on the amount of resources used by virtual machines.

discharger thread is insignificant. In fact, no statistically significant differences in the performance of VMs and the host system were found when we conducted a series of benchmark tests.

3. Case studies

To demonstrate the effectiveness of Virtual Battery, we describe three examples in this section: enforcing low-power resource management policies on the guest system, testing hibernation initiated by a low-battery condition, and allocating energy budget to VMs.

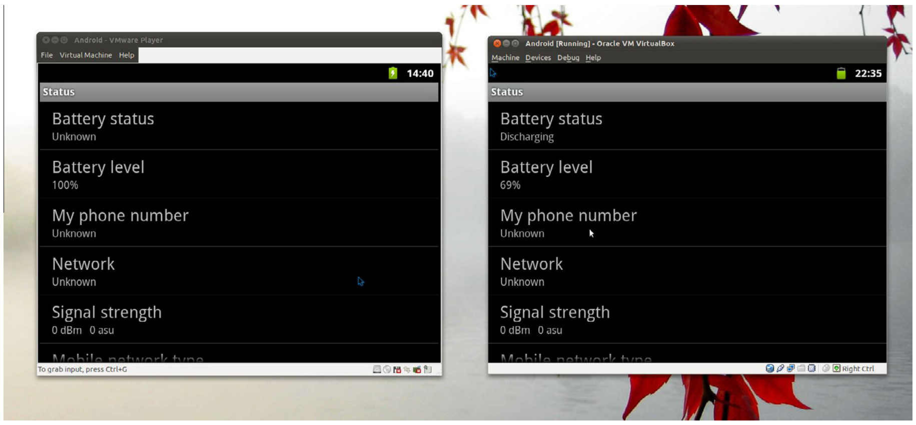
3.1. Testing environment

Because Virtual Battery is implemented as an ACPI-compliant battery device, any guest OS equipped with an ACPI battery driver can be configured with Virtual Battery. To prove this, we created VM images of diverse guest OSs. The OSs used in our research were various Linux distributions including Ubuntu, Microsoft Windows XP and Windows 7, and Android x86. All OSs were automatically configured with a conventional ACPI bridge driver during their installation. The VMM was reconfigured to use Virtual Battery after installation of the guest OSs, and then all VMs successfully recognized Virtual Battery.

As shown in Fig. 6, when they were configured with a conventional bridge driver, the guest OSs displayed their battery status as unknown since the host system has no battery installed. After they were configured to use Virtual Battery, the battery status information showed both the charging status of a battery and its remaining charge. In addition, when the battery manager changed the charging status or remaining charge for the battery, the changes were instantly reflected in the battery status information of the guest system.

3.2. Enforcing battery-mode resource management on VMs

End users, but not developers, may need virtualization when using multiple OSs in a single system or sandboxing unverified


Android - VMware Player
File Virtual Machine Help
Android [Running] - Oracle VM VirtualBox
Machine Devices Debug Help

14:40
Status
Battery status
Unknown
Battery level
100%
My phone number
Unknown
Network
Unknown
Signal strength
0 dBm 0 asu
To grab input, press Ctrl+G

22:35
Status
Battery status
Discharging
Battery level
69%
My phone number
Unknown
Network
Unknown
Signal strength
0 dBm 0 asu
Right Ctrl

Fig. 6. Two Android x86 VMs are running side by side on an AC-powered desktop PC. The left one is configured with a conventional bridge driver and the right one with Virtual Battery.

applications. Modern OSs carry out a lot of housekeeping tasks in the background such as backing up, system software updating, and system status logging. In addition to OSs, applications also conduct many background operations including application updates, data reorganization, data backup, and data fetching from remote sites. Virtualization inherently induces a significant overhead. Therefore, such background tasks can have a significant negative effect on user experience. What users want from type II virtualization is to run a specific set of applications inside VMs, so it is highly desirable to allow only essential tasks in a VM and to limit non-urgent background activities.

Most modern OSs provide battery-mode resource management policies, which minimize energy consumption using various means, including suspension of housekeeping tasks. Virtual Battery can enforce these power-aware resource management policies on VMs by making them believe that they are working on battery power. In addition, many applications adapt their behavior to the system energy status. Thus, making them believe that the system is working on battery power would limit their activities to only crucial operations.

All three guest systems indicated that they were working on battery power when Virtual Battery entered the discharge mode. In addition, we observed that the Windows 7 VM halted system update operations when in discharge mode. We also verified that an antivirus program started to download update files immediately after we changed the status to "being charged". According to its manual, the software downloads only when the system is connected to external power.

3.3. Testing OS reactions to the low-battery condition

Virtual Battery can be useful when developers want to test software components that react to battery status changes. For example, some OSs conduct housekeeping tasks, such as backing up, disk defragmentation and malware detection, only when the system is connected to AC power. Another example is for testing hibernation. Hibernation is a power-down mode supported by most modern OSs. When the remaining battery charge reaches its low threshold, hibernation saves the contents of the entire memory to non-volatile storage and shuts down the system. The saved contents are loaded back into memory to resume the system after a sufficient amount of energy is secured.

Since it takes a significant amount of time to discharge a battery to the low threshold and to recharge it to a sufficient level for resumption, testing of hibernation features is time-consuming Virtual Battery can expedite testing procedures for hibernation. We observed behaviors exhibited by the Android x86, Ubuntu, and Windows XP VMs after we changed the remaining charge level to 3% of the battery capacity. The host system was a workstation equipped with no batteries. The Ubuntu and Windows XP VMs successfully initiated and finished hibernation immediately after we changed the battery level as shown in Fig. 7. Since Android does not have a hibernation feature, the Android VM halts all applications and shuts down automatically. These same behaviors were observed when the discharger caused the remaining charge to reach the low threshold.

3.4. Allocating energy to VMs

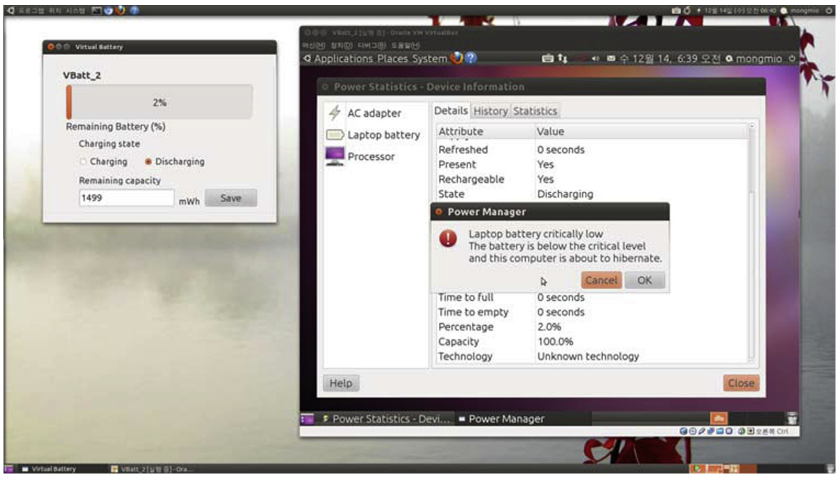
When multiple VMs of different importance are running, the user can control the amount of processor time for a VM by using Virtual Battery.

We set the same amount of remaining battery charge for two VMs and ran a movie player continually on one of the VMs. The discharger threads for both VMs were activated and drained their batteries according to their processor times. The movie player VM discharged its battery 4.5 times faster than the other VM. Finally, when the remaining charge reached 10%, the movie player VM popped up a warning message and hibernated, while the other VM was still in operation.

Because the current discharger model is built on the simple assumption that battery consumption is proportional to processor utilization, the calculated discharge does not accurately reflect actual energy consumption [12]. Despite this inaccuracy, we believe that our preliminary discharging model provides an intuitive interface for regulating VMs so that they consume resources according to their priorities because the amount of remaining charge would be recognized more intuitively than the processor time or other resource units and the remaining lifetime for a VM is automatically calculated by the guest OS as its remaining battery lifetime.

4. Related work

Cao et al. first coined the term Virtual Battery for an energy abstraction layer that provides battery abstraction to each application individually on sensor network systems [13]. It distributes the remaining energy to applications according to their priorities, and it prohibits applications from consuming more energy than their allocated share by estimating their energy consumption. The Virtual Battery proposed in this paper is intended to emulate and manipulate battery behavior as a development and testing tool, whereas the Virtual Battery of Cao et al. 13 is intended to



VBatt_2

2%

Remaining Battery (%)

Charging state

Charging [✓ Discharging]

Remaining capacity

1499 mWh

Attribute	Value
Refreshed	0 seconds
Present	Yes
Rechargeable	Yes
State	Discharging
Time to full	0 seconds
Time to empty	0 seconds
Percentage	2.0%
Capacity	100.0%
Technology	Unknown technology

Power Manager

Laptop battery critically low
The battery is below the critical level
and this computer is about to hibernate.

Fig. 7. After the charge of the virtual battery was set to a low value, the guest operating system initiated the hibernation procedure.

guarantee a proportional share of energy allocation in sensor network systems.

A battery emulation scheme must estimate the energy consumption of each processing entity. The basic proposition of power estimation at the software level is that greater resource usage causes higher power consumption [14]. This fundamental point has been the basis of a lot of research on energy efficiency optimization schemes, estimation models for power consumption and power management schemes [15].

It has been suggested that the power consumption of a VM might be measured or estimated based on the resources used [6,11]. These approaches commonly identify relationships between resource utilization and system power consumption. Based on these relationships, they estimate the power consumption contributed by a VM according to its resource usage. These estimation models are necessary for optimizing the energy usage of virtualized and consolidated enterprise workloads in data centers while preserving service quality [16]. However, to the best of our knowledge, this is the first work that utilizes a VM energy consumption estimation model to emulate battery behavior as a development tool.

A VM power management scheme has been proposed that provides a VM with computing resources in accordance with its requested power state and workload characteristics [17]. The aim of this research was to improve VM energy efficiency while maintaining service-level agreements. On the contrary, our research enforces the performance and energy demands of a host system or host users on VMs because, in type II virtualization, the system-wide throughput or energy efficiency is more important than that of a single VM.

Our approach to the battery discharger oversimplifies the power consumption model of a VM, since our primary goal was to implement a battery virtualization layer that can easily be manipulated. However, when accurate simulation of battery discharge characteristics is necessary, incorporation of existing models for estimating the power consumption of a VM will easily satisfy such demand.

The Cinder OS [18,19], which is a research OS for mobile devices, accurately monitors and controls the energy consumption of each application. In type II virtualization, a VM is implemented as a task or process on the host OS. Therefore, if the host OS employs the Cinder architecture and incorporates its monitoring and control features into our scheme, a VM can be seen as a unit of energy allocation. Thus, applications inside a VM can grasp the allowable energy for the VM through Virtual Battery and adjust their service fidelity to fit the energy allocation [20].

In practice, the OS emulator included in the development environment for Google Android provides battery subsystem emulation [21]. Developers can change the battery charging status and remaining battery capacity of the emulated machine, and watch the system behavior according to that change. Different from our approach, this is only applicable to the Android OS, and it does not provide the battery discharging simulation.

For web applications to determine the host system, the World Wide Web Consortium proposed an API named *battery status API* [22]. A web application is able to check the battery capacity of the host system by calling the API functions, and adapt its behavior to the current energy condition. An approach similar to Virtual Battery can be applied to create a testing environment for energy-aware web applications. By inserting a battery system emulation layer in between the host and the API, the developers can test the behavior of web applications according to the host battery state.

5. Conclusions

Because of the ever-increasing importance of battery management in the mobile cyber-physical systems, the development of battery-related software components is becoming crucial. However, because manipulation of battery status is a physical and slow task, testing of battery-related features with real batteries is usually cumbersome and time-consuming. A battery emulation layer for VMs that the user can manipulate would relieve this problem.

We proposed and implemented Virtual Battery, a battery virtualization scheme for type II full virtualization platforms. Virtual Battery provides each VM with a separate hypothetical battery that is monitored and controlled through a GUI application. In addition, we suggested a discharger for Virtual Battery that gradually decreases the remaining energy of each hypothetical battery according to the amount of processing time allocated to the corresponding VM. To show the effectiveness of our scheme, we carried out tests.

The proposed scheme is suitable for testing procedures for battery- and power-related OS or application features. Therefore, it will be helpful in improving the energy efficiency of software. In addition, when end users run multiple VMs for various purposes, our scheme can be used to control VM energy consumption by setting the remaining charge for each VM.

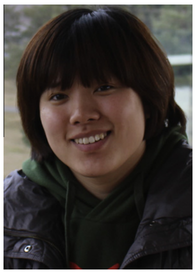
The prototype currently supports only x86-based VMs since the virtualization platform that the prototype was built upon is dedicated for x86 virtualization. However, considering the growing popularity of mobile embedded systems, porting VirtualBox to virtualization platforms for embedded hardware architecture, such as ARM or MIPS, is highly desirable. We believe that QEMU, a platform-independent virtualization tool, would be an appropriate base for this purpose, and we also expect that porting Virtual Battery to QEMU would require marginal effort since QEMU supports ACPI emulation and the ACPI emulation component of QEMU is well modularized similarly to VirtualBox.

Acknowledgement

This research was supported by Next-Generation Information Computing Development Program (2012-0006423) and Basic Science Research Program (2012R1A1A2A1A10038823) through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology.

References

[1] X. Xie, H. Jiang, H. Jin, W. Cao, P. Yuan, L.T. Yang, Metis: a profiling toolkit based on the virtualization of hardware performance counters, Human-centric Computing and Information Sciences 2 (2012) 1–15.
[2] X. Wang, Y. Sang, Y. Liu, Y. Luo, Considerations on security and trust measurement for virtualized enviroment, Journal of Convergence 2 (2011) 19–24.
[3] F. Bellard Qemu, a fast and portable dynamic translator, in: Proceedings of the USENIX, Annual Technical Conference vol. 2005, 2005, pp. 41–46.
[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, in: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, 2003.
[5] Oracle, Virtualbox, <https://www.virtualbox.org/>.
[6] N. Kim, J. Cho, E. Seo, Energy-based accounting and scheduling of virtual machines in a cloud system, in: Proceedings of the 2011 IEEE/ACM International Conference on Green Computing and Communications, 2011.
[7] R.P. Goldberg, Architecture of virtual machines, in: Proceedings of the Workshop on Virtual Computer Systems, ACM, 1973.
[8] Hewlett-Packard, Intel, Microsoft, P. Technologies, Toshiba, Advanced configuration and power interface specification, 2006. <http://www.acpi.info/>.
[9] D.C. Snowdon, E. Le Sueur, S.M. Petters, G. Heiser, Koala: a platform for os-level power management, in: Proceedings of the 4th ACM European Conference on Computer Systems, 2009.
[10] F. Bellosa, The benefits of event: driven energy accounting in power-sensitive systems, in: Proceedings of the 9th ACM SIGOPS European Workshop, 2000.
[11] A. Kansal, F. Zhao, J. Liu, N. Kothari, A.A. Bhattacharya, Virtual machine power metering and provisioning, in: Proceedings of the 1st ACM Symposium on Cloud Computing, 2010.
[12] M. Li, F. Wang, D. Wang, Preheating simulation of temperature rising of power battery, Journal of Convergence 2 (2011) 9–12.
[13] Q. Cao, D. Fesehaye, N. Pham, Y. Sarwar, T. Abdelzaher, Virtual battery: an energy reserve abstraction for embedded sensor networks, in: Real-Time Systems Symposium, vol. 2008, 2008, pp. 123–133, http://dx.doi.org/10.1109/RTSS.2008.41.
[14] V. Tiwari, S. Malik, A. Wolfe, M.T.-C. Lee, Instruction level power analysis and optimization of software, in: Proceedings of the 9th International Conference on VLSI Design: VLSI in Mobile Communication, 1996.
[15] B. Singh, D.K. Lobiyal, A novel energy-aware cluster head selection based on particle swarm optimization for wireless sensor networks, Human-centric Computing and Information Sciences 2 (2012) 1–28.
[16] J. Stoess, C. Lang, F. Bellosa, Energy management for hypervisor-based virtual machines, in: 2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference, 2007.
[17] R. Nathuji, K. Schwan, Virtualpower: coordinated power management in virtualized enterprise systems, in: Proceedings of twenty-first ACM SIGOPS Symposium on Operating Systems Principles, 2007, pp. 265–278.
[18] S. Rumble, R. Stutsman, P. Levis, D. Mazières, N. Zeldovich, Apprehending joule thieves with Cinder, ACM SIGCOMM Computer Communication Review 40 (1) (2010) 106–111.
[19] A. Roy, S.M. Rumble, R. Stutsman, P. Levis, D. Mazières, N. Zeldovich, Energy management in mobile devices with the Cinder operating system, in: Proceedings of the 6th European Conference on Computer Systems, 2011, pp. 139–152.
[20] J. Flinn, M. Satyanarayanan, Energy-aware adaptation for mobile applications, in: Proceedings of the 17th ACM Symposium on Operating Systems Principles, 1999.
[21] Android, Using the Android emulator, <http://developer.android.com>.
[22] W3C, Battery status API, 2012, <http://www.w3.org/TR/battery-status>.



Youngjoo Woo received the B.S. degree in electrical engineering from Inha University in 2009 and the M.S degree in computer science from Ulsan National Institute of Science and Technology in 2012. Currently, she is a Ph.D. candidate in the department of electrical and computer engineering at Sungkyunkwan University. Her research interests include system software, virtualization, power-aware computing and embedded systems.

Seon-Yeong Park received the B.S. degree in computer science from Chungnam National University and the M.S. and Ph.D. degrees in computer science from KAIST in 2001 and 2011, respectively. From 2001 to 2003, she was a researcher at ETRI. From 2011 to 2012, she was at KAIST as a post-doctoral researcher. Currently, she is a researcher at National Security Research Institute, Korea. Her research interests include flash memory filesystems, power-aware systems and system security.

Euiseong Seo received his B.S., M.S., and Ph.D. degrees in com- puter science from Korea Advanced Institute of Science and Technology (KAIST) in 2000, 2002, and 2007, respectively. He is currently an assistant professor in College of Information and Communication Engineering at Sungkyunkwan University, Korea. Before joining Sunkyunkwan University in 2012, he had been an assistant professor at Ul- san National Institute of Science and Technology (UNIST), Korea from 2009 to 2012, and a research associate at the Pennsylvania State University from 2007 to 2009. His re- search interests are in power-aware computing, real-time systems, embedded systems, and virtualization.