Power-Aware I/O Virtualization

Kun Tian, Yaozu Dong
Intel China Software Center
{Kevin.tian, eddie.dong}@intel.com

ABSTRACT

Power consumption is one of the key concerns in modern computers within which I/O consumes a significant portion of power, from portable devices to servers. This concern has led to the development of various hardware and software techniques to improve the energy efficiency of I/O subsystems in the native platform. However, virtualization poses new challenges, preventing those techniques from achieving the desired level of energy efficiency.

In this paper, we analyze how I/O virtualization challenges impact energy efficiency, and propose a novel power-aware I/O virtualization architecture to tackle them. Our preliminary research on portable devices shows that new architecture can significantly extend battery life in a typical idle scenario, compared to existing solutions.

Categories and Subject Descriptors

C.0 [**Computer Systems Organization**]: General–*System architectures*; D.4.4 [**Operating Systems**]: Communications Management; D.4.7 [**Operating Systems**]: Organization and Design;

General Terms

Design, Management, Experimentation

Keywords

I/O virtualization, Power management

1. INTRODUCTION

The I/O subsystem consumes a significant portion of power in modern computers [7][9], leading to the development of various hardware and software techniques to help prolong battery life of portable devices or to reduce the costs around cooling servers.

In hardware, some of or all internal components can be regulated to consume less power. Specific components may be slowed down if I/O requests don't impose peak pressure, such as dynamic rotation speed of disk spindle motor [5]. Clock and voltage may be throttled to lower level. When there's enough idle duration among I/O requests, both clock and power can be gated to consume the least power, such as PCIe D3 state [1].

At the same time, the modern OS manages to use hardware low power states in an energy efficient manner, such as trying to finish more jobs with the same number of joules, or to consume less power for the same task. Usually a system-wide coordination framework [15][16][17] is deployed, which connects to both resource owners (such as the I/O drivers) and resource consumers (such as the user applications). Resource consumers report their requirements to, and also proactively create chances for, the I/O driver to make good use of low power states of the device. This is usually feasible because the OS controls the entire system with insights into all layers, from the hardware, device driver to the applications.

However, virtualization imposes new challenges to I/O energy efficiency regarding to system-wide coordination. With virtualization, resource consumers on I/O devices, such as various application workloads, are now consolidated into multiple separated virtual environments. The host OS doesn't have original insights into those workloads running in VM, because the guest OS now has full control of them. Thus, the host I/O device driver lacks enough inputs and chances to make an energy efficient decision. It's possible to rely on the I/O virtualization path, which may aggregate some indirect information about VM workload patterns. However, no I/O virtualization techniques provide such power awareness so far, as far as we know.

We provide a full anatomy about the challenges of adding power awareness into the I/O virtualization path. A novel power aware I/O virtualization architecture, called PAIOV, is proposed to reunite VM workloads to the host power management framework. PAIOV shows flexibility in integrating with various I/O virtualization methods. Finally, we propose several techniques on PAIOV to proactively create power saving chances, which can be well integrated with the host power management framework. Our preliminary research on portable devices shows that PAIOV can reduce the virtualization overhead of battery life in a typical idle scenario to be as small as 4%, while existing solutions incur ~15% overhead.

Section 2 introduces I/O power management in a native system. The full anatomy of I/O virtualization challenges are provided in section 3. Section 4 is our proposal of a power-aware I/O virtualization solution. Section 5 introduces our preliminary experiments, while Section 6 introduces related work. The last section is conclusion and future work.

2. I/O POWER MANAGEMENT

I/O power management architecture includes a Power Management Agent (PMA) in the driver and a Power State Agent (PSA) in the device. The PMA negotiates power management

This paper appeared at the Second Workshop on I/O Virtualization (WIOV '10), March 13, 2010, Pittsburgh, PA, USA.

policies, and then requests the PSA to regulate the device into various low power states, as shown in Figure 1.

Both the device and the driver in Figure 1 contain a "Functional Core" component, denoting all functions other than for power management purposes. For example, the functional core of the device would include circuits necessary for handling I/O requests, mode setting, interrupts, etc.

2.1 I/O PMA

An I/O PMA is the bridge connecting the I/O sub-system to the coordinated power management framework in the OS, as shown in Figure 2. In such a framework, all PMAs are organized in a layered style, with the higher level PMA holding a broader view of the power management requirements. For example, the PMAs in the lowest level directly manage low power state of a set of resources (CPU, memory, I/O devices, etc.), while the higher level PMA may proxy requirements directly from user level applications. Some I/O PMAs may be the parent of others, based on platform hierarchy, as shown in Figure 2 by a PCI bridge.

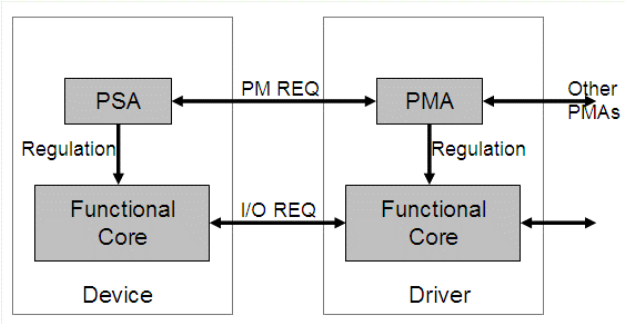

Figure 1: Architecture of the I/O Subsystem's Power Management.

Based on that coordinated framework, the I/O PMA could receive requirements from consumers using that device, and also propagate its own requirement on other system resources. Exchanged requirements include kinds of QoS metrics (latency, throughput, media quality, etc.), statistics (I/O pattern, resource use, etc.), power budget (watt, thermal, etc.), and some general policies (performance-oriented, power-oriented, etc.).

The I/O PMA makes energy efficient decisions based on negotiated policies, and then sends requests to the device's PSA. Besides hunting for power saving chances passively, the I/O PMA may proactively regulate the functional core of the driver to create more chances, such as burst request in [16] with negligible performance drop.

2.2 I/O PSA

An I/O PSA exposes those power-friendly capabilities to the I/O PMA in a well-defined interface. The I/O PMA specifies a low power state request and send sent to the I/O PSA. Based on the requested state, the I/O PSA then regulates the functional core of the device to consume less power. The actual regulation method is specific to the device implementation, as introduced in section 1.

Some I/O PSAs may include their own intelligence to trigger a transition to a low power state [5], when the I/O PMA is missing or less capable of making an optimal decision.

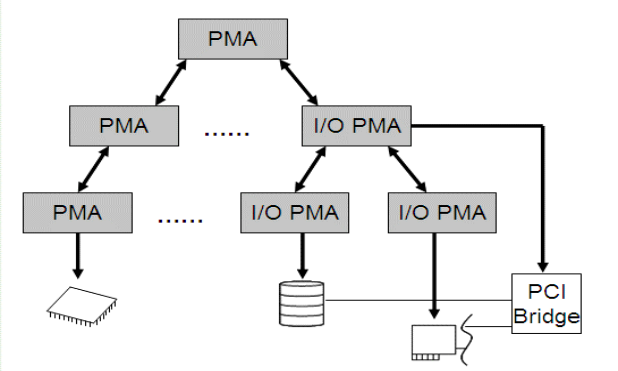

Figure 2: Power Management Architecture in the OS.

3. I/O VIRTUALIZATION CHALLENGES

Virtualization is bringing new challenges to traditional I/O power management, which haven't been noted in previous I/O virtualization research because previously the only goal was performance [2] [3].

3.1 Problem

The obvious challenge is the split power management framework; because multiple workloads are now consolidated into separate VMs. Each VM is a closed environment containing heterogeneous OS and application stacks. Workloads running within a VM are then disconnected from monolithic coordination framework on the host side. There's no channel to convey the workload requirements out of the VM to host I/O PMA, or to enforce power-friendly policies on the VM for creating more power-saving chances. Without such a tight coordination channel, the host I/O power management techniques are then either conservative to save power, or even apt to save power overly with an undesired drop in performance.

Obviously, new techniques are required to reunite a global coordination power management framework across separated VMs.

3.2 Option-A: high level PMA proxy

The intuitive option is to plant some cooperative PMAs into the proprietary power management framework in each VM. Those planted PMAs behave as the bridge between two frameworks of the host and the VM, to convey workload requirements and administrative policies to host I/O PMA. However the feasibility is not always there, regarding diverse virtualization usages:

- VM is not owned by the physical machine owner in the public cloud. The customers renting computing resources fully control their own proprietary VMs. They don't want to allow power saving customizations to help cloud service providers, if there is no return value.
- VM may contain a heterogeneous power management framework compared to the host side, even when external customization is allowed. Such heterogeneities are reflected

on API/ABI, PM capabilities, stability, and so on. In the extreme case, there's no power management awareness in the VM at all. Due to the heterogeneities involved, the option A method of enabling power savings results in increasing complexity and maintenance costs.

This limitation led us to think from another perspective, as follows.

3.3 Option-B: virtual I/O proxy

A more generic option, to reunite VM workloads with host I/O subsystem, is to introduce power awareness in the I/O virtualization path. The guest OS has intrinsic knowledge about internal workload activities, and thus is able to make intelligent decision to balance power and performance. The I/O virtualization path can expose virtual low power states to VM, and then deduce implicated workload requirements, based on the state chosen by VM. Even when VM is not able to specify virtual low power state, I/O virtualization path still allows for passive power-saving possibilities based on the access patterns of virtual I/O requests. This provides better flexibility than option A, given that the I/O virtualization path is well under the control of the host VMM.

However, there are also challenges with this option, which haven't been resolved yet:

- Few I/O virtualizations currently provide virtual low power states. This is echoed in Qemu emulated devices, the Xen frontend (FE) NIC driver [4] and the KVM virtio NIC driver [12]. VMs are even denied controlling the power of pass-through devices in many implementations.
- Diverse I/O virtualization approaches provide various chances to add power awareness. Some present VM with virtualized I/O resources; while others may grant VM with direct access to physical I/O. VM may reuse unmodified drivers, or load a new cooperative entity. No common architecture exists to the best of our knowledge.
- Virtual I/O semantics may cause VM power management technique to generate an undesired effect. Some techniques are designed upon physical I/O semantics. This doesn't hold true, however, for virtual I/O, such as those honoring disk spin-down while virtual disk has no spindle motor. There's still a long way from the virtual I/O to the physical I/O subsystem, so the benefits may be amortized to, instead, hurt performance without help on power. Such side effects have to be balanced carefully.
- I/O patterns from multiple VMs are out of step, which compresses power-saving room in physical I/O subsystems. Even when each VM proactively arranges I/O requests in bursts, those bursts are dispersed system-wide, with few idle periods.
- VMs may be prioritized based on signed SLAs. Differentiation must be given based on the extent of power management in the relevant I/O virtualization path.

This had led to our design for a new I/O virtualization architecture, capable of reuniting host the I/O subsystem with its clients to gain energy efficiency.

4. POWER AWARE I/O VIRTUALIZATION

We propose Power Aware I/O Virtualization (PAIOV) as the solution to improve I/O energy efficiency, which is designed with two goals in mind: 1) Modularized to allow component modification in various I/O virtualization implementations; 2) Enabled for host I/O power management in wide usage models, regardless of whether workload hints are directly available from the guests.

4.1 Architecture Overview

PAIOV architecture is shown in Figure 3**.** Ignoring the gray boxes, the vertical dotted line shows the split in environment before PAIOV is added, in which both the host and the guest may have their own power management frameworks. The two frameworks are disconnected, so the I/O PMA connecting to the host framework lacks a coordination channel and cannot determine what the workloads running in VM are.

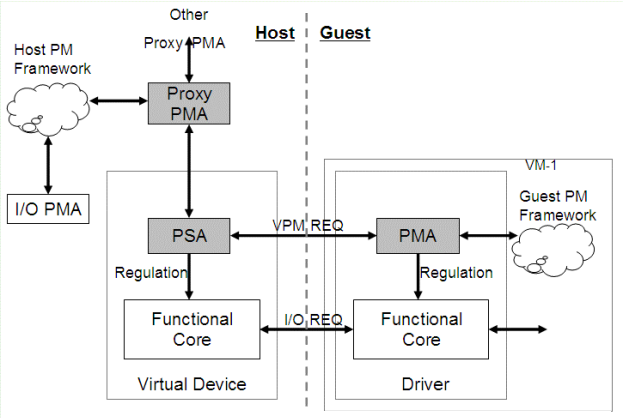

Figure 3: Architecture of Power Aware I/O Virtualization (PAIOV).

The gray boxes represent entities newly introduced or enhanced by PAIOV, including:

- A virtual PSA in the host to emulate a set of low power states for virtual devices
- A virtual I/O PMA in the guest driver, with connections to the guest PM framework
- A proxy PMA on the host side, connecting to the host PM framework

The virtual I/O PMA resides in the guest PM framework, and thus is able to leverage OS intelligence on the workload requirements. The virtual I/O PMA is connected to the virtual PSA through an I/O Virtual Power Management (VPM) channel. Then the virtual PSA behaves as the bridge to flow the guest OS decision into the host PM framework, through a new entity, the "Proxy PMA". The Proxy PMA is implemented according to the host PM specification. Before forwarding the virtual PSA info to the host PM framework, the proxy PMA translates it into a recognizable format desired by the host I/O PMA.

Coordination may also happen in the opposite direction. The proxy PMA occurs in the host PM, as a consumer on the host I/O sub-system so it can be integrated into existing coordination techniques, to tightly assist the host I/O power management, such as proposed in Currentcy [16]. That type of tight cooperation not only requires the proxy PMA to reveal guest workload requirements, but also encourages the proxy PMA to create more power saving chances by regulating virtual I/O activities. The proxy PMA may forward regulation policy to the virtual PSA, the virtual I/O PMA, or even propagate into the guest PM framework, and then reach guest workloads. Actual regulation may happen at either entity if it's allowed.

4.2 Implementation Notes

PAIOV is designed with modularization from the ground up, adapting to various I/O virtualization methods.

In the emulated I/O method, we enhance device model to expose virtual PSA, according to existing device semantics, such as the D3 state in a PCIe device [1]. An unmodified guest driver manipulates low power state through a well defined interface. Whether to have a virtual I/O PMA, however, fully depends on guest driver implementation. So, there is no assurance that virtual PSA can always acquire useful information from the guest.

Passthrough I/O has a similar situation, as the emulated I/O with an unmodified driver used in guest. Intuitively, there's no requirement to intercept power management decisions from the guest. However, it's not true since power dependencies exist in platform hierarchy, such as between the PCI bridge and the downstream device. When the downstream device is granted to a guest, PAIOV will deny guest access to the real PSA of that device. Instead, a virtual PSA intercepts the guest request and then sends it to the host PM framework, which decides whether to forward it to the real PSA.

There are some cases where a cooperative virtual I/O driver is loaded in the guest, such as in paravirtualized I/O (Xen frontend/backend and Virtio) and in self virtualized I/O (VMDq and SR-IOV). Though they differ in the way virtual or real I/O resources are accessed, they do have one merit in common. Virtual I/O PMA and I/O VPM channels are customizable to communicate rich information and also respect the distinct characteristics of the deployed I/O virtualization method. For example, new virtual I/O semantics can be observed, by noting that a virtual device is composed of a set of instructions consuming CPU, memory, and I/O resources. Virtual I/O PMA can arrange virtual I/O activity to benefit host PMAs controlling the CPU and memory, other than simply for I/O PMA.

The virtual PSA in PAIOV is implemented with plenty of power awareness. Besides forwarding guest power decisions, it also regulates emulation logic of virtual devices to consume less power, such as by relaxing the polling timer, according to a low power request it received. Various policies exist about how to map a virtual low power state to internal resource management, but we are not covering that topic here.

4.3 Proxy PMA

The proxy PMA is the most important component in PAIOV architecture, playing two main roles.

The first role is to digest information from the virtual PSA, including explicit workload requirements delivered from cooperative virtual I/O PMA, specified virtual low power states from an unmodified virtual I/O PMA, and also virtual I/O access patterns gathered from device emulation logic. The last category is useful when the guest driver is missing a virtual I/O PMA, providing the least information indirectly implicating VM workload activities. The proxy PMA then translates those hints into a recognizable format into the host PM framework to influence the host I/O PMA.

The other role is to proactively create power saving chances for the host I/O subsystem, since fine-grained cooperation between resource consumers and resource providers normally brings good energy efficient decisions. The proxy PMA must adapt to the specific coordination interface defined in the host PM framework, such as new file operation APIs defined in the Cooperative I/O [17]. In the meantime, the proxy PMA also regulates the I/O dispatch process of device emulation in a way which explicitly honors the host I/O PMA semantics:

- The proxy PMA could coordinate with other proxy PMAs serving other VMs, before dispatching I/O requests to the host I/O subsystem. Since every VM is scheduled randomly, mixed I/O access patterns of all VMs tends to be dispersed even when each proxy PMA adds burstiness to its own path. For example, Currentcy [16] allows multiple consumers, pooling their currentcy, to satisfy the entry price of disk spin-up, if a single consumer can't afford enough currentcy. Similarly, multiple proxy PMAs can pool their currentcy together and then burst their I/O requests at the same pace.
- The proxy PMA coordination can also take SLA and QoS into consideration, which is especially useful in the public cloud. A master-slave relationship can be created among multiple proxy PMAs. The proxy PMA with the strictest SLA is the master, having, at its disposal, maximum freedom to issue I/O requests. On the other hand, the proxy PMA, with relaxed SLA, simply regulates I/O dispatch in piggyback mode, such as holding its I/O request until the master does so. Amazon now allows customer to bid on unused EC2 capability, called spot instances [19]. However, spot instances can be terminated at any time, once the bid is below the latest spot prices. This is one perfect example to use slave mode, since the customer won't expect strict SLA with it.
- The proxy PMA also needs to handle urgent conditions, such as overheating circumstances. When an urgent condition occurs, the host PM framework notifies all relevant entities to reduce its activity in cooling the system down. The proxy PMA could throttle the virtual I/O dispatch aggressively, since thermal is now the first concern. When the VM has direct access to physical I/O subsystem, the proxy PMA may ask the VMM to throttle virtual interrupts injected into that VM, if the VMM provides such interface. This has the indirect effect of reducing guest I/O activities.

Above are some early ideas in this area. Future exploration is necessary to determined wider policies.

5. EXPERIMENTS

Our research is still in the early phases. We chose a typical virtualization scenario for portable devices, in which both the host and VM are idle. Virtualization on portable devices has gained increasing interest quickly and is actually necessary for them [18].

An Intel® Atom™ processor[1] based netbook is used, running in 1.8 GHz, with 1 G memory and 80 G disk. Moblin™ v.2 [13] is installed on the host, and a Windows XP VM is created with 1 vcpu, 128 M memory, and 4 G disk, using KVM [14]. Seamless RDP [11] is used to launch an XP application in host Moblin v.2, without showing the XP desktop. We measured the battery life when the system was idle, which is the typical scenario of a portable device, and then compared the battery life and power consumption, with and without virtualization, as shown in Figure 4. "Native" is the environment without virtualization. "Base" is the case after XP is launched with one IE6.0 launched in seamless RDP channel, without PAIOV. Comparing "Base" to "Native", power consumption increases by 19%, while battery life decreases by 15%.

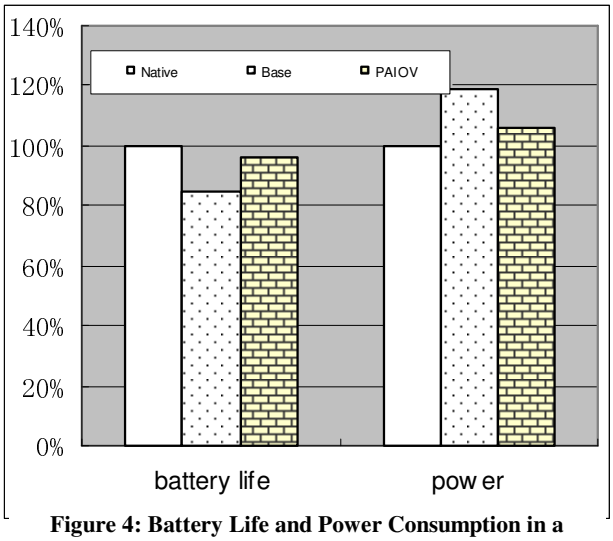

Figure 4: Battery Life and Power Consumption in a Virtualized Portable Device.

Analysis of our preliminary experiment shows that I/O virtualization is the major cause of shortened battery life, lacking virtual PSA in device emulation and effective proxy PMA. The device model (Qemu) is always placed in a busy mode, preparing to service peak requests, though the system is actually idle. In such a mode, several high frequency timers are always armed, such as a 4 ms timer in audio emulation and a 33 ms timer in VGA card emulation. Besides that, Qemu also places the underlying host stack into a similar busy mode, which adds more overhead. For example, the audio server of the host Moblin v.2 also stays in a state equipped with hot timers. Similar power unawareness exists in various I/O emulations.

We implement a simple PAIOV prototype, by introducing a virtual PSA and a proxy PMA to virtual devices with hot activities, observed in the paragraph above. Take the virtual sound card (ES1370) for example. Guest PMA in the XP sound card driver requests to disable the channel when there's no sound stream in VM. The virtual PSA intercepts the request, and then puts the virtual device into a low power mode, in which all hot timers are stopped. Then the virtual PSA sends a request to the proxy PMA, which is connected to the power management framework of Moblin v.2. Finally, the proxy PMA notifies the audio PMA of Moblin v.2 to enter low power mode. The result is promising, as shown by the "PAIOV" bar. The battery life gap reduces from 15% to 4%, and power consumption shows a similar effect, reducing from 19% to the current 6%.

6. RELATED WORK

Various coordinated power management frameworks have been proposed. In such a coordinated environment, the energy budget can be enforced in a multi-layer OS environment with energy-aware resource allocation [10]. VirtualPower [6] introduces a coordination framework in data center virtualization. VMs are allowed to manage a set of virtual frequency scaling points. The VMM intercepts these points and takes them as hints to manage the physical scaling points, which is conducted following local and global configuration policies. Though it shares a similar philosophy, it doesn't touch energy efficiency issues in the I/O virtualization area. ClientVisor [8] coordinates power management roles between VMM and a primary VM, in a specific desktop virtualization model, and thus cannot be applied to generic I/O virtualization.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we analyze I/O virtualization challenges regarding to I/O energy efficiency. Consequently, a novel power aware I/O virtualization architecture (PAIOV) is proposed, adapting to various I/O virtualization methods and diverse virtualization usages. Several techniques are proposed to efficiently integrate PAIOV with the existing host PM framework efficiently. In our experiment of a PAIOV prototype, we reduced battery life overhead in the idle scenario from the original 15% to current 4%. However, our work is still in the early stages, for example only the idle standby scenario is measured. Next, we'll extend PAIOV to various working modes (video playback, mp3 playback, web browsing, etc.) with various I/O virtualization approaches. The integration between proxy PMA and host coordination framework would be among our top interests, too.

8. REFERENCES

[1] PCI Special Interest Group, http://www.pcisig.com/home

[2] A. Menon, A. L. Cox, W. Zwaenepoel, Optimizing Network Virtualization in Xen. Proceedings of the USENIX Annual Technical Conference, Boston, MA, 2006, 15-28.

[3] G. Liao, D. Guo, L. Bhuyan, S. R King, Software techniques to improve virtualized I/O performance on multi-core systems. Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, San Jose, CA, 2008, 161-170

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, Xen and the art of virtualization, In proceedings of the 19th ACM symposium on Operating Systems Principles, Bolton Landing, NY, 2003, 164-177

[5] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir and H. Franke. DRPM: Dynamic Speed Control for Power Management in Server Class Disks. In Proceedings of the 30th annual international symposium on Computer architecture (ISCA), 2003.

[6] R. Nathuji and K. Schwan. VirtualPower: Coordinated Power Management in Virtualized Enterprise Systems. In Proceedings of International Symposium on Operating System Principles (SOSP), 2007.

[7] A. Mahesri and V. Vardhan. Power Consumption Breakdown on a Modern Laptop. In 4th International Workshop of Power-Aware Computer System (PACS), 2004.

[8] H. Chen, K. Yu, H. Jin, K. Tian, Z. Shao, and K. Hu. ClientVisor: leverage COTS OS functionalities for power management in virtualized desktop environment. In Proceedings of ACM SIGPLAN/SIGOPS international conference On Virtual Execution Environments (VEE), 2009.

[9] X. Fan, W.-D.Weber and L. A. Barroso. Power Provisioning for a Warehouse-sized Computer. In Proceedings of the 34th Intertional Symposium on Computer Architecuture (ISCA). 2007.

[10] J. Stoess, C. Lang, and F. Bellosa. Energy management for hypervisor-based virtual machines. In Proceedings of the USENIX Annual Technical Conference, June 2007.

[11] Seamless RDP. http://www.cendio.com/seamlessrdp/

[12] Virtio. http://www.linux-kvm.org/page/Virtio

[13] Moblin 2.0. http://moblin.org

[14] KVM. http://www.linux-kvm.org/

[15] Windows power management framework. http://msdn.microsoft.com/en-us/library/aa923906.aspx

[16] H. Zeng, C.S.Ellis, A.R.Lebeck, A.Vahadat. Currentcy: a unifying abstraction for expressing energy management policies. In proceedings of the USENIX Annual Technical Conference, June 2003

[17] A. Weissel, B. Beutel, F. Bellosa. Cooerative I/O: a novel I/O semantics for energy-aware applications. In proceedings of the 5yh symposium on Operating System Design and Implementation (OSDI), Dec 2002

[18] VMware Mobile Virtualization Platform (MVP). http://www.vmware.com/products/mobile/

[19] Amazon EC2 spot instances. http://aws.amazon.com/ec2/spot-instances/