Energy-Oriented Partial Desktop Virtual Machine Migration

NILTON BILA, University of Toronto and IBM Research
ERIC J. WRIGHT and EYAL DE LARA, University of Toronto
KAUSTUBH JOSHI, AT&T Labs Research
H. ANDRÉS LAGAR-CAVILLA, Google Inc.
EUNBYUNG PARK and ASHVIN GOEL, University of Toronto
MATTI HILTUNEN, AT&T Labs Research
MAHADEV SATYANARAYANAN, Carnegie Mellon University

Modern offices are crowded with personal computers. While studies have shown these to be idle most of the time, they remain powered, consuming up to 60% of their peak power. Hardware-based solutions engendered by PC vendors (e.g., low-power states, Wake-on-LAN) have proved unsuccessful because, in spite of user inactivity, these machines often need to remain network active in support of background applications that maintain network presence. Recent proposals have advocated the use of consolidation of idle desktop Virtual Machines (VMs). However, desktop VMs are often large, requiring gigabytes of memory. Consolidating such VMs creates large network transfers lasting in the order of minutes and utilizes server memory inefficiently. When multiple VMs migrate concurrently, networks become congested, and the resulting migration latencies are prohibitive. We present *partial VM migration,* an approach that transparently migrates only the working set of an idle VM. It creates a partial replica of the desktop VM on the consolidation server by copying only VM metadata, and it transfers pages to the server on-demand, as the VM accesses them. This approach places desktop PCs in low-power mode when inactive and switches them to running mode when pages are needed by the VM running on the consolidation server. To ensure that desktops save energy, we have developed sleep scheduling and prefetching algorithms, as well as the *context-aware selective resume* framework, a novel approach to reduce the latency of power mode transition operations in commodity PCs. Jettison, our software prototype of partial VM migration for off-the-shelf PCs, can deliver 44–91% energy savings during idle periods of at least 10 minutes, while providing low migration latencies of about 4 seconds and migrating minimal state that is under an order of magnitude of the VM's memory footprint.

Categories and Subject Descriptors: D.4.7 [**Operating Systems**]: Organization and Design—*Distributed systems*; D.4.4 [**Operating Systems**]: Communications Management—*Network communication*

This research was funded in part by the National Science and Engineering Research Council of Canada (NSERC) under grant number 261545-3.
Authors' addresses: N. Bila, IBM Thomas J. Watson Research Center, 1101 Kitchawan Rd, Route 134, Yorktown Heights, New York 10598, USA; email: nilton@us.ibm.com; E. J. Wright, 215 Pebblecreek Walk, London, Ontario N6G 0K3, CANADA; email: eric.wright@northernthought.net; E. de Lara, Department of Computer Science, University of Toronto, 10 King's College Road, Room 3302, Toronto, Ontario M5S 3G4, CANADA; email: delara@cs.toronto.edu; K. Joshi, AT&T Shannon Laboratory, 1 AT&T Way, Bedminster, New Jersey 07921, USA; email: kaustubh@research.att.com; H. A. Lagar-Cavilla, Google Inc., 1600 Amphitheater Parkway, Mountain View, California 94043, USA; email: andres@lagarcavilla.org; E. Park, 1600 Baity Hill Drive, Apt 119, Chapel Hill, North Carolina 27514, USA; email: eunbyung@cs.unc.edu; A. Goel, The Edward S. Rogers Sr. Department of Electrical & Computer Engineering, University of Toronto, 2001B Sandford Fleming, 10 King's College Rd, Toronto, ON M5S 3G4, CANADA; email: ashvin@eecg.toronto.edu; M. Hiltunen, AT&T Shannon Laboratory, 1 AT&T Way, Bedminster, New Jersey 07921, USA; email: hiltunen@research.att.com; M. Satyanarayanan, Computer Science Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, Pennsylvania 15213-3891, USA; email: satya@cs.cmu.edu.

DOI: http://dx.doi.org/10.1145/2699683

General Terms: Design, Experimentation, Measurement

Additional Key Words and Phrases: Desktop virtualization, virtual machine migration, power management

ACM Reference Format:
Nilton Bila, Eric J. Wright, Eyal de Lara, Kaustubh Joshi, H. Andrés Lagar-Cavilla, Eunbyung Park, Ashvin Goel, Matti Hiltunen, and Mahadev Satyanarayanan. 2015. Energy-oriented partial desktop virtual machine migration. ACM Trans. Comput. Syst. 33, 1, Article 2 (March 2015), 51 pages.
DOI: http://dx.doi.org/10.1145/2699683

1. INTRODUCTION

Personal computers (PCs) are indispensable in the modern office. Studies [Webber et al. 2006; Nedevschi et al. 2009; Agarwal et al. 2009] have found that although office PCs are idle for much of the day, they remain powered. We consider PCs to be idle in the absence of the user and when not running tasks that have been specifically scheduled to take advantage of such absence, as is often the case with compute or I/O-bound tasks, such as virus scans and system backups. Idle times have been shown to add up to close to 12 hours per day, excluding off times [Nedevschi et al. 2009]. Unfortunately, an idle PC can consume up to 60% of its peak power. Although modern computers support low-power modes of operation, defined by the Advanced Configuration and Power Interface (ACPI) specification [Intel Corporation et al. 1999], the same studies have shown that these are seldom employed mainly because of applications with always-on semantics. Applications such as instant messengers (IM), Voice over IP (VoIP) clients, and remote desktop access and administration utilities maintain network presence even when the PC is idle. To address these shortcomings, modern PCs support mechanisms for waking up from low-power mode at the request of remote hosts (e.g., Wake-on-LAN [Lieberman Software Corporation 2006]). Although these work for applications that provide remote access to the PC, they are ineffective for client-driven applications that communicate with external servers continuously. Applications such as IM and VoIP clients periodically report to protocol servers to maintain the user's online status and poll for incoming messages. Many cloud-based Web applications rely on a client-driven model to maintain a connection to external services. Asynchronous JavaScript and XML (AJAX) applications such as Facebook Chat [Facebook, Inc. 2012], Google Docs [Google, Inc. 2012b], and Gmail Chat [Google, Inc. 2012a] are just a few examples of applications for which Wake-on-LAN-like mechanisms do not work.

An attractive solution is to host the user's desktop inside a Virtual Machine (VM), migrate the VM to a consolidation server when idle, and put the PC to sleep [Das et al. 2010]. The key advantage of this approach is that it does not require changes to applications or special-purpose proxies. However, a straightforward implementation causes large network transfers during migration of memory (and optionally disk) state that can saturate shared networks in medium to large offices and allocates server memory inefficiently.

In this article, we present *partial VM migration*, an approach that consolidates only the small working set of the idle VMs. Partial VM migration is based on the observation that an idle desktop, even in spite of background activity, requires only a small fraction of its memory and disk state to function. Our experience with Windows and Linux VMs shows that, when idle, these VMs access less than 10% of their memory allocation and about 1MiB of disk state. Partial VM migration creates a partial VM on the server and transfers on demand only the limited working set accessed while the VM is idle. The PC sleeps when the consolidated partial VM needs no state from it and wakes to service on-demand requests. We call these opportune sleeps *microsleeps*. Migrating the VM back to the user's PC is fast because partial VM migration maintains VM residues on the PC and transfers only the dirty state created by the partial VM back to the

PC. PCs can be desktop machines, laptops, or other machines that host user desktop sessions locally.

Partial VM migration makes energy-oriented desktop consolidation practical. Because its network transfers are small, and partial VMs require only a small fraction of their PC mode memory footprint, both the network and server infrastructure can scale well with the number of users, and migration times are very small. High migration efficiency creates more opportunities for energy savings because shorter periods of idleness can be targeted. Fine-grain migration also lowers the penalty for poor migration decisions. If an idle user becomes active sooner than expected, the user hardly notices that his VM has migrated.

The main challenge of the approach is ensuring that desktops save energy, a goal made difficult by the increased use of power when PCs transition between low-power and full-power modes and the long latencies of the transition operations. We have developed a scheduling algorithm that enables desktops to sleep only if there is an expectation that they will save energy, explored page prefetching strategies that increase microsleep durations, and developed the context-aware selective resume framework, an architecture used to reduce the latency of power mode transition operations in commodity PCs. Context-aware selective resume provides PCs in low-power mode with a context descriptor of the task requesting the wake-up, which enables the PCs to initialize only devices and system components needed for the completion of the task.

Jettison is our software prototype of partial VM migration for off-the-shelf PCs. Our experience with a deployment of Jettison in a Linux desktop environment shows that significant energy savings are achievable without adversely impacting user experience. Within an hour of inactivity, desktops were able to save up to 78% of energy, and in 5 hours the savings increased to 91%. Experienced migration times were near 4s, and migration sizes averaged under 243MiB for Linux desktop VMs with 4GiB of nominal memory. When Jettison was combined with the context-aware selective resume framework in controlled experiments, energy savings ranged from 44% to 66% during shorter idle intervals of 10min to an hour. Our experiments also show that, in a simulated environment with 500 users, partial VM migration can deliver similar energy savings as full VM migration while using less than 10% as much network resources and providing migration latencies that are three orders of magnitude smaller. The capital investment needed to achieve these energy savings is modest. Even a small private cloud can support a large number of desktop VMs because the VMs migrated there have small network and memory footprints: they only do what is needed to sustain always-on semantics for desktop applications.

Although our current partial VM migration prototype targets VMs with local storage on the PC, the approach is equally applicable to enterprise deployments with shared network storage. Similarly, partial VM migration is complementary to solutions like Intelligent Desktop Virtualization (IDV) [Intel Corporation 2011c] that simplify desktop management by centralizing it while supporting local execution. Whereas these approaches concentrate on managing and backing up persistent VM state, partial VM migration concerns itself with the migration of runtime state.

1.1. Contributions

This article makes the following contributions: (i) it shows that office PC idle times are largely distributed throughout the day, including long periods during overnight hours and short ones during office hours; (ii) it shows that the working set of idle Linux and Windows VMs is small and consists mostly of memory state (disk is less than 1%); (iii) it shows that migrating a VM in full is unnecessary and, indeed, does not scale well for energy-oriented idle desktop consolidation; (iv) it shows that on-demand page and disk block requests are clustered enough to allow desktops to save energy by sleeping between request bursts; (v) it presents partial VM migration, an approach that consolidates only the idle working set of idle VMs; (vi) it shows that partial VM migration can save as much energy as full VM migration while sending less than 10% of the data, delivering migration latencies that are three orders of magnitude smaller, and providing consolidation ratios that are an order of magnitude higher; and (vii) it introduces context-aware selective resume, an approach that reduces latency of resume operations of PCs in low-power by selectively initializing only devices and system components relevant to the task requesting the system resume.

1.2. Organization

The remainder of the article is organized as follows. In Section 2, we begin with a discussion of the background to our work and the advancements in the PC technology that are used to reduce idle power use. In Section 3, we discuss desktop virtualization, its applications in this problem space, and we identify the challenges encountered with solutions that rely on existing virtualization technology in enterprise offices. In Section 4, we characterize the working sets of idle Linux and Windows desktop VMs with the goal of identifying opportunities to improve upon existing approaches. In Section 5, we present partial VM migration, our approach for reducing energy use by idle desktops in enterprise offices. In Section 6, we introduce Jettison, our implementation of partial VM migration for off-the-shelf desktop systems, and we present results from a deployment in a research environment. We identify two challenges arising from the use of on-demand page migration from idle PCs—the energy cost of servicing page faults and impact on reliability of desktop hardware components—and we discuss considerations when deploying partial VM migration in enterprise environments. In Section 7, we present context-aware selective resume, an approach to improve energy savings by reducing PC suspend and resume cycle times. In Section 8, we use simulations driven by user activity traces to extend results from our deployment of Jettison into large office environments and evaluate the scalability of the approach. In Section 9, we discuss related work. Finally, in Section 10, we conclude the article and discuss avenues for future work.

2. BACKGROUND

In this section, we motivate the significance of the problem with a discussion of previous studies on the power states of idle PCs, as well as our own study of user inactivity. We then explore power management features on modern PC hardware designed to tackle idle power waste. We discuss low-power operational states, dynamic scaling of CPU power, and on-demand wake-up mechanisms and explain why these have not been very successful in the field.

2.1. The Scale of the Problem

Desktop and laptop PCs pervade the modern office. A study on the after-hours status of office equipment across U.S. office buildings has found that 60% of desktop computers remain powered in overnight hours and that only 4% of them use low-power states [Webber et al. 2006]. More recently, Nedevschi et al. [2009] reported similar findings, with desktops remaining idle for more than 12 hours daily (not including off periods) and only 20% using low-power states. Ineffective management of power in idle PCs results considerable energy waste. A study commissioned by the California Energy Commission [California Energy Commission 2006] has found that, in office buildings, IT equipment is now the second largest consumer of electricity (20%), only surpassed by lighting (25%).

These studies have found that office users and administrators actively disable power management on their desktops. Desktop machines commonly ship preconfigured to

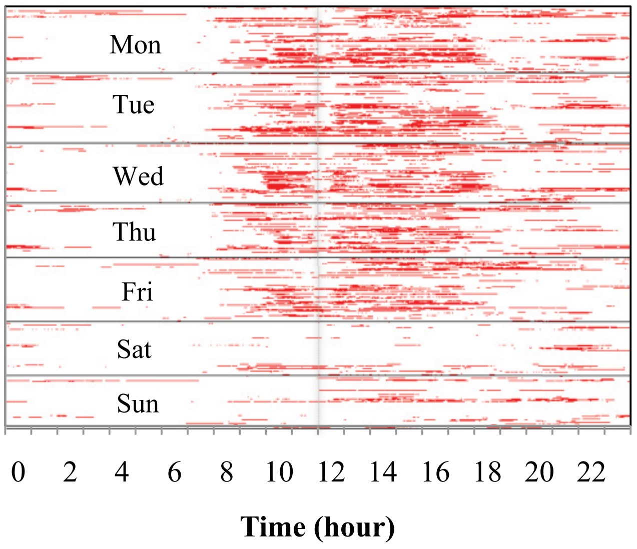


Fig. 1. Desktop and laptop user activity. Each imaginary horizontal line represents a user. Active times are represented in red and idle times in white.

enter low-power modes whenever user inactivity is detected for a significant period. For example, by default, Windows 7 systems enter low-power sleep state after 30min of user inactivity. A survey [Agarwal et al. 2009] sought to identify the reasons that people keep idle desktop PCs powered. The survey found that, among office users, the most oft-cited reasons were remote access and background applications. Of all respondents, 52% left their systems powered to support remote access to files and software on the PCs and 35% to support applications running in the background. Of these applications, IM and e-mail were the most popular, with a combined 47% share of the responses.

Idle times are not limited to after-work hours. To gain a better understanding of idle times, we collected traces of desktop and laptop user activity in an industrial research lab. We deployed an activity tracker that determines whether the user is active every 5s. Users were said to be inactive if the tracker finds no mouse or keyboard activity and no application has disabled the system's screen saver. Applications such as QuickTime video player disable desktop screen savers when playing a video to prevent interruptions. Our tracker ran on Mac OS X because this was the most popular operating system in the lab. We deployed the tracker over a period of 4 months on the primary computers of 22 researchers, including both desktops and laptops. These machines were user administered without any corporate lockdowns. We collected 2,086 person day traces from which a sample of 500 is presented in Figure 1. In the figure, each horizontal line represents one person day for one machine, with a dot indicating that the machine is in use and a white space when the machine is idle.

Our results confirm prior findings that desktops are idle for significant portions of time. More importantly, our findings show that these idle times are distributed throughout the day. Figure 1 shows that, in addition to the long overnight hours, there are significant idle times spread throughout the day, including many at lunch hour (12 p.m.), but also arbitrarily at other times. Idle times occur, for example, when users step away for meetings, coffee breaks, or even as they simply turn away from the computer to perform other tasks (e.g., reading a printout or speaking with co-workers). In our traces, idle times that are shorter than 2 hours add up to more than 25% of total idle times. This is indicative of substantial opportunities to save energy during work hours.

Table I. ACPI System Power States

State	Suspend Time (s)	Resume Time (s)	Power (W)
S0: On	N/A	N/A	60.42 ± 6.09
S3: Suspend-to-RAM	7.24 ± 0.46	8.77 ± 0.24	2.33 ± 0.01
S4: Suspend-to-disk	12.97 ± 0.06	26.29 ± 0.33	1.22 ± 0.01
S5: Soft Off	12.70 ± 0.74	40.70 ± 0.12	1.21 ± 0.01

ACPI power states and transition times for a Dell Studio XPS 7100 desktop. Standard deviations reported following ±.

The poor adoption of power management in desktop PCs is in stark contrast to its success on the monitors attached to the same PCs. For example, Webber et al. found that only 20% of computer monitors remained powered overnight (in contrast to 60% of desktop PCs), and about 75% were found to use low-power states (versus 4% of desktops). For monitors, unlike PCs, being in low-power mode has no impact on running applications, and the latency to resume to full-power operation is low. Moreover, monitor stand-by modes are highly energy efficient, making their use attractive. For example, in stand-by mode, a 19-inch Dell 1905FP LCD monitor consumes only 4% of its average active power of 32W. To achieve similar levels of success in desktop PCs, desktop power management must provide similar benefits, namely: (i) low-power operation must not interfere with application execution; (ii) resume latency must be low; and (iii) low-power mode must provide deep energy savings.

2.2. System Power States

In recognition that PCs spend a significant fraction of time idle, in 1996, the PC industry introduced the Advanced Configuration and Power Interface (ACPI) [Hewlett-Packard Corporation et al. 2009], now found in virtually all PCs. ACPI allows PCs to enter and exit low-power modes without incurring the time penalty and loss of application execution context associated with full power off and boot-ups. The ACPI standard defines system power states, as well as the power states for component devices needed to support them, that can be managed directly by the OS. System power states defined include:

(1) S0 (On): The system is in full operation and the CPU executing instructions.
(2) S1: A low wake latency sleep state. CPU stops executing instructions, but remains powered. CPU and hardware retain context.
(3) S2: Similar to S1 but CPU does not retain context. CPU context is flushed to DRAM.
(4) S3 (Suspend-to-RAM): The CPU and devices are off. DRAM remains powered only to maintain state (self-refresh mode). CPU context is flushed to DRAM.
(5) S4 (Suspend-to-Disk): The CPU and devices (including DRAM) are off. DRAM state and CPU context have been flushed to disk or persistent storage.
(6) S5 (Soft Off): CPU, DRAM, and devices are powered off. Power supply unit (PSU) maintains minimal power to support power up via keyboard, USB device, a soft power switch, or network interface.
(7) G3 (Mechanical Off): Power to the system has been fully severed by mechanical means (e.g., unplugged or PSU power switched off). No power is running through the circuitry.

Of these power states, the most widely available in modern PCs are S0, S3, S4, S5, and G3, of which only the first four can be entered via software control. Table I shows the power use and transition times for a Dell Studio XPS 7100 desktop with a 3GHz quad-core AMD CPU, 6GiB of DRAM, and configured with Debian GNU/Linux 5.0 and kernel version 2.6.32. The table points to three important facts. First, it shows that low-power states significantly reduce the power used by the desktop by more than an order of magnitude—from 60.42W to 2.33W for S3, and 1.22W for S4. Second, the table shows that entering and exiting low-power states (S3 and S4) is much faster than powering off and subsequently booting-up the desktop. Resume time—time to transition from low-power or off state to on state—is of particular importance because it is the time users must wait before the system becomes usable. While booting the system takes 40.70s, resuming from S3 and S4 only take 8.77s and 26.29s, respectively. Finally, the table shows that the low-latency low-power states provide energy savings that are comparable to full power off (within 1.1W).

These findings indicate that low-power states are better suited to reduce energy use during idle times than a full power off. They are fast, ensuring minimal user disruption, and have negligible penalty in terms of energy waste. Indeed, these results show S3 state to be the most attractive due to its low latency.

Whereas available low-power modes allow PCs to reduce their energy use when idle, they cause the CPU to halt all instruction execution, and, as a result, all applications stop running and the PC drops off the network. The user cannot remotely access the PC, and applications with always-on semantics are disconnected from network peers. The result is seen in the study by Webber et al., discussed in Section 2.1, that finds that only 4% of office desktop PCs make use of low-power states, even in overnight hours. In the next sections, we discuss solutions engendered by PC makers to support some of these background applications and identify their shortcomings.

2.3. Maintaining Network Presence

Two strategies have been pursued by PC makers to support application functionality while PCs operate in low-power. The first introduces processor level low-power modes that regulate the power used by the CPU while it executes instructions, and the second is a set of technologies that enable on-demand wake-up of PCs in low-power by remote applications.

Processor-level power management. Processor power-performance states (P-states) enable the OS to dynamically control the power used by the processor to match the demands from running workloads. This is accomplished by modulating the processor core's voltage and operating frequency (DVFS), which results in changes to the processor's dynamic power—the power dissipated directly in order to execute instructions during clock cycles [Weiser et al. 1994; Govil et al. 1995; Sueur and Heiser 2010]. P-states are designed to allow processors to lower their power use while continuing to execute instructions. However, they only manage the power used by the processor, and the processor is responsible for only a fraction of the total power used by PCs. The bulk of the power (up to 75%) is used by the remaining components, which include motherboard, DIMMs, hard disks, network cards, cooling components, and graphics cards [Fan et al. 2007], and these components continue to waste power during idle times. Moreover, because P-states only affect the dynamic power of the processor and not its static power (power dissipated as a result of current leakage in the circuit), the dynamic power range offered is limited. For example, an Intel Pentium M uses 24.5W in high-frequency mode (1.6GHz) and 6W in its lowest frequency mode (0.6GHz). More recent processors have been found to show even smaller dynamic power ranges because static power represents a larger fraction of dissipated power in smaller transistors with low threshold voltage [Sueur and Heiser 2010].

Remote wake-up. To ensure that whenever PCs enter low-power system states (which are effective in reducing total system power use) they continue to support administrative applications (e.g., compliance scanners) that require access to the desktop, the industry has developed technologies that enable these applications to remotely wake the PC on demand. These include Wake-on-LAN [Lieberman Software Corporation 2006] and Wake-on-Wireless-LAN [Intel Corporation 2006]. The former is found in most desktops and laptops with Ethernet network interfaces and the latter on laptops with 802.11 wireless interfaces. Before suspending the PC into low-power, the OS configures the Network Interface Controller (NIC) to remain powered during system low power and to wake the system upon receipt of a designated network packet. In standard Wake-on-LAN mode, known as “magic” packet mode, this is a broadcast frame containing the Media Access Control (MAC) address of the NIC. In wake-on-destined mode of operation, the NIC wakes the PC on receipt of any packet destined to itself, independent of the payload. Upon receipt of the designated packet, the NIC controller issues a Power Management Event (PME), which is a trigger for the BIOS to power the system up. Whereas on-demand wake-up solutions are suitable for centrally managed administrative applications, they are not well suited for applications that need to run on the PC itself while it is in low-power state. These are applications that either perform background tasks or issue requests to network hosts to maintain network presence. Examples of these include instant messengers, VoIP clients, and DHCP clients, as well as Web 2.0 applications that run within Web browsers and maintain presence on cloud servers. For these applications, the host continues to fall off the network when in low-power mode.

3. DESKTOP VIRTUALIZATION

In this section, we discuss machine virtualization, an established technology that decouples the logical computer (OS and applications) from the physical machine. Virtualization enables modern approaches to dealing with idle desktops. We describe two such approaches and discuss the issues involved in implementing them in production environments. The first approach, Virtual Desktop Infrastructure (VDI) has seen some adoption in enterprise office environments, and the second, consolidation of idle desktop VMs, is a more recent research proposal.

Machine virtualization or simply *virtualization* decouples the logical machine (OS, applications, etc.) from the physical machine hardware. It does this by interposing a Virtual Machine Monitor (VMM) [Seawright and MacKinnon 1979] or hypervisor layer between the operating system and the hardware. Physical resources expected by the OS are virtualized by the hypervisor, allowing them to be multiplexed among multiple VMs. The hypervisor ensures that all running VMs have fair access to the machine’s resources and provides isolation between VMs. Virtualization enables a single physical machine to run multiple desktop sessions simultaneously (including OSes and applications), each with its own (virtualized) CPU, memory, disk and network interface.

In addition to hardware multiplexing, virtualization provides another benefit: It enables migration of VMs across physical hosts [Chen and Noble 2001; Sapuntzakis et al. 2002; Kozuch and Satyanarayanan 2002]. Because the VM software is decoupled from the host hardware, a VM can begin execution at one host and subsequently migrate to a different host without significant down time. Both hardware multiplexing and VM migration can be used to reduce energy use by idle desktops, as we discuss in the next sections.

3.1. Virtual Desktop Infrastructure

Virtual Desktop Infrastructure (VDI) has emerged in the past decade as a means to simplify enterprise desktop management and deliver enterprise-class storage services [Baratto et al. 2004; VMware, Inc. 2013; Citrix Systems, Inc. 2011; Oracle Corporation 2012]. In enterprise VDI deployments, multiple desktop VMs run on shared corporate servers. Users access personal VMs over the network from a client running on their desks. As illustrated in Figure 2, user input is intercepted from the client and transmitted to the VM on the server, where processing takes place. Because the VMs run in

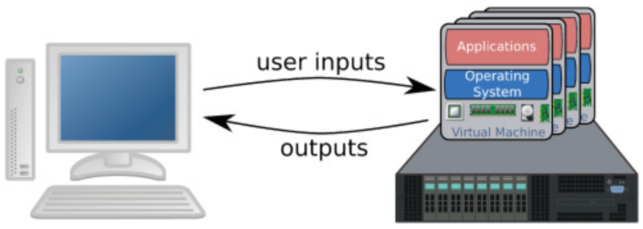


Fig. 2. Overview of virtual desktop infrastructure (VDI).

shared servers under the control of IT administrators, they can be configured to boot from a single system disk image that holds the OS and shared software. This read-only disk image is known as the “golden” image. User data and configuration are stored on private secondary disk images. This separation of system and user state ensures that administrators can perform maintenance tasks affecting many VMs (e.g., applying software patches) by updating a single disk image.

Full-time conversion to VDI can help reduce idle energy use because VDI enables the use of low-power stateless thin clients with only enough hardware to render graphics generated on the server and capture user input. When users are idle, these “thin clients” can be powered off without interrupting VM execution taking place on the shared server. However, deployment of VDI has several disadvantages:

(1) **High infrastructure cost.** First, because desktop VMs require large memory allocations (gigabytes), each server is only able to host a limited number of VMs. As a result, in large offices, a large upfront investment on servers is required, and the energy costs of running the servers are high. Second, the servers must be provisioned to accommodate the peak workloads of the desktops VMs. This means not only large memory, but also very fast CPUs, network interfaces, and the like, which have been found to be energy inefficient under low utilization [Barroso and Hölzle 2007]. Third, because of high storage performance requirements of VDI, storage cost is also high [Spruijt 2010].
(2) **Limited access to local hardware.** Because VMs run on remote servers, access to local hardware resources that deliver brisk performance or specialized functionality (e.g., 3D acceleration and dedicated media encoding hardware) is limited. Applications such as video conferencing clients and online gaming, which maintain network presence (and that could benefit from VDI), also require access to local decoding or graphics acceleration and controller devices.

The result of these shortcomings has been a slow adoption of VDI [Fograrty 2011]. Full-fledged desktops continue to outsell thin clients used in VDI, and these thick clients, with their energy inefficient idle operation, will remain in use for years.

Hardware vendors such as Intel now champion IDV [Intel Corporation 2011c], a different model of deployment of desktop virtualization. In this model, VMs run on the end-user desktops in the same client-side manner advanced in the Internet Suspend/Resume [Kozuch and Satyanarayanan 2002] and the Collective [Sapuntzakis et al. 2002] projects but, like in VDI, they rely on golden system disk images stored on enterprise-class shared file systems. Software updates remain simple to apply; however, the PCs must remain powered during idle times in order to sustain always-on applications. The approach we discuss next enables idle PCs to enter low-power modes during idle times while offering compatibility with the IDV model of desktop virtualization for enterprise deployments.

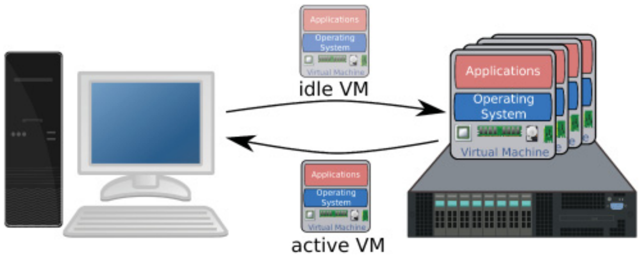


Fig. 3. Overview of idle desktop VM consolidation.

3.2. Idle Desktop VM Consolidation

The LiteGreen project [Das et al. 2010] proposed ephemeral consolidation of idle desktop VMs using existing virtualization technology to reduce energy use. When the user is active, the VM runs on desktop hardware located on the user's desk. When the user is inactive, the VM is migrated to a consolidation server (where it shares resources with other idle VMs), and the PC is placed in low-power state. When the user returns to the PC and interacts with the mouse or keyboard of the desktop, the VM is migrated back to the PC. Figure 3 illustrates the nature of idle VM consolidation. This approach uses live VM migration [Clark et al. 2005], which not only allows VMs to continue execution after migration, but also reduces down time during migration. As memory state is migrated from the source host (e.g., the PC) to the destination (e.g., consolidation server), the VM continues to execute at the source. Once a minimal memory state remains to be migrated, execution is halted at the source, the remaining state (including CPU context) is migrated, and execution resumes at the destination.

Although this approach allows background applications to continue to run and maintain network presence, and it supports access to local desktop hardware (e.g., 3D acceleration), our experiments indicate that, because desktop VMs have large memory footprints, migrating them is slow. Users are forced to interact with their VMs with degraded performance over VNC [Richardson et al. 1998] while migration completes, and this constitutes a hindrance to adoption in enterprise environments. Next, we quantify the performance of existing VM consolidation technologies on enterprise-class hardware and show that they are lacking.

Methodology. In these experiments, we migrated an idle desktop VM from a desktop computer to a consolidation server. Our desktop and server consisted of Dell PowerEdge R610 machines with 24GiB of RAM, eight 2.3 GHz Xeon® cores, Fusion-MPT SAS drives, and a Broadcom NetXtreme II™ gigabit NIC. Both systems ran Debian GNU/Linux 5.0 desktop VMs with 4GiB of memory and 12GiB of disk images on top of Xen 3.4 hypervisor [Barham et al. 2003]. The disk images were hosted on a shared storage server provided through Red Hat's network block device GNBD, which ensured disk availability upon migration (i.e., only memory and CPU state was migrated between the desktop and consolidation server). The hardware used for the desktop has CPU, memory, and storage that deliver performance exceeding that of typical desktop computers, and we will show that, even with high-performance hardware, live VM migration performance remains unsuitable for idle desktop VM consolidation.

We began the experiments with a warm-up phase, in which we ran typical desktop applications that allocate and update memory. We ran a script that opens a spreadsheet, a word document, and a presentation document on OpenOffice.org; a PDF document on Evince document viewer; as well as seven Web pages on Firefox Web browser. These Web pages included CNN.com, Slashdot.com, Maps.Google.com, Ole.com.ar, and the SunSpider JavaScript benchmark [SunSpider Benchmark 2012], as well as Acid3 Web standard compliance test [Acid3 Test 2012]. The last two pages are used to exercise various components of the Web browser. After running the script, we allowed the VM to remain idle for 1min, after which the VM was migrated to the consolidation server through a dedicated network switch. Each experiment was repeated five times, and we report the average results. Even though the footprint of the VM was 4GiB, the script would consistently lead the VM to using only 1.2GiB of its memory.

Results. Live migration took, on average, 38.59s to migrate the desktop VM and consumed 4.27GiB of network bandwidth. A user returning to her desk has to wait nearly 40s before her VM can run from the desktop. However, the migration of a single VM at a time does not give the full picture of the user experience. Specifically, a "resume storm" occurs when multiple users start work or resume work at the same time or in close proximity. Resume storm events are likely after correlated idle periods (e.g., at start of work day in the morning, at end of lunch break in the afternoon, or at the end of meetings).

We experimented with concurrent live migration of multiple VMs. As expected, live migrating multiple VMs out of a single consolidation server concurrently degrades latency linearly: four VMs take an average of 137s, whereas eight take 253s. Staggering the resume times (so that VM migrations start moments apart) helps, but even with 20s pause between resumes of eight VMs, latency still averages 115.62s. Not only is live migration unable to ensure quick resumes, it also introduces significant strain on the network (number of migrations times the average VM size).

To alleviate the latency and congestion problems caused by live VM migration, Das et al. propose using *ballooning* [Waldspurger 2002] to reduce the size of the memory image before migration. Ballooning is a technique that allows the memory footprint of a VM to shrink on demand. Although ballooning reduces the VM footprint, this happens at a considerable expense of time and I/O.

We repeated the VM migration experiments just described; however, before performing migration, we used ballooning to reduce the VM's memory footprint. In our experiments, ballooning in Xen was able to reduce the idle VM's footprint to 423MiB. To reach this footprint size, we turned on swapping on the VM to avoid killing any processes. However, ballooning took an average of 328.44 ± 69.41s to reach its saturation point, and, in the process, it evicted 275.99 ± 9.25MiB of cached disk state and swapped out 449.08 ± 51.55MiB of main memory to secondary storage using the network resources. These results demonstrate that ballooning is ineffective at reducing migration latencies. Moreover, while the ballooned VMs can be easily migrated—the VMs with 423MiB footprints were migrated, on average, in 4.86s—the memory state swapped out and the evicted cached disk state have to be reconstructed from the shared storage after resume, resulting in additional network usage and slowed desktop responsiveness. Thus, although ballooning reduces the VM's footprint on the consolidation server, it fails to significantly reduce migration latencies or network bandwidth. Even if we use local disks instead of shared storage to reduce network usage, ballooning latency is still prohibitive, and, in this case, the desktops must be awakened to serve pages being swapped in while the desktop VM executes at the consolidation server.

As a refinement to full VM migration, the consolidation server could persist pages of consolidated VMs to a local cache for use during subsequent consolidations of the same VM. When the VM is consolidated again, only the pages that have been updated on the desktop need to be migrated to the server. Although our experiments with server-side caching of pages were limited to an approach that migrates only the working set of idle VMs (which we discuss in Section 5.3), we expect the refinement to also benefit full VM migration. Results with partial migration of idle VM working sets show that the use of a server cache can reduce memory transfer by 28%. The success rate of the cache hinges on how much the memory of the VM changes while it runs on the desktop, and, since the VM is active when running on the desktop, it is likely to see large changes. For example, simple activities such as Web browsing have been shown to produce hundreds of megabytes of dirty memory state [Surie et al. 2008].

4. WORKING SETS OF IDLE DESKTOP VMS

Desktop VM consolidation is an attractive solution for reducing energy use of idle desktops. It allows desktops to enter low-power states while applications continue to run and maintain network presence; it requires no changes to applications, thus ensuring that it can be readily used with existing software; and it supports access to local hardware. However, as we showed in Section 3.2, this approach has several shortcomings that result from the large memory footprints of desktop VMs: (i) migrating desktop VMs over the network is slow; (ii) in shared networks, overlapping VM migrations can cause network congestion; and (iii) consolidation ratios on the server are low, resulting in high infrastructure costs and low energy savings. In this section, we show that idle Linux and Windows desktop VMs have small working set sizes (pages that are accessed by the idle VMs) that are up to an order of magnitude smaller than their memory footprints, a finding that we use to design our desktop consolidation approach that migrates only the idle working set.

Over two sets of experiments, we collected memory and disk access traces of idle desktop VMs. In the first set of experiments, we ran a controlled set of applications on idle Linux and Windows 7 VMs and collected memory access traces to better understand the VMs' memory access patterns and working set sizes for different applications. In the second experiment, the deployment study, we deployed an early version of our VM consolidation software (Jettison) on the desktops of three university researchers over a period of 7 weeks to understand memory and disk accesses by idle VMs. We describe the details of the prototype in Section 6.

4.1. Controlled Workloads

To record memory access traces for the Linux VMs, we used SnowFlock's [Lagar-Cavilla et al. 2009] VM fork abstraction. SnowFlock supports rapid cloning of VMs by creating and running a replica of a VM with minimal CPU and memory state. As the replica runs, it attempts to access memory pages. If these pages have not been copied from the parent VM, the replica faults and SnowFlock copies the pages on demand, allowing VM access to the pages. For our experiments, we ran our VM workloads on a desktop and, after allowing the VM to become idle for at least 5min, used SnowFlock to create a replica of the VM on the server. We then halted execution on the desktop and monitored memory pages that SnowFlock transferred from the desktop to the server as a function of the VM's execution on the server. SnowFlock also provides a migration avoidance mechanism for pages the VM's kernel intends to overwrite with new allocations. Our subsequent experiments with desktop workloads in Section 6.2 show that this mechanism reduces migration size on average by 9.06MiB.

Because SnowFlock supports only paravirtualized guest VMs (VMs whose kernels have been modified specifically to take advantage of virtualization), we used a different approach to track page accesses by the Windows guest. To that end, we instrumented the Xen hypervisor with code that invalidates all page entries in the Extended Page Table (EPT) [Neiger et al. 2006] of the memory management unit. VM accesses to pages cause faults that we trap in the hypervisor and log to a file. This approach works only with fully virtualized guests that take advantage of hardware virtualization assists, including the EPT.

We describe next the experimental platform and the idle desktop workloads studied in this experiment. For each workload, we collected traces from three runs to ensure the results reported are consistent.

Table II. Idle Session Workloads

Workload	Description
Login	The login screen of the desktop system
E-mail	Mozilla Thunderbird connected to an IMAP e-mail server; the client polls the server every 10min.
IM	The Pidgin multiprotocol IM client connected to an IRC room with more than 100 active users
Multitask	A desktop session with the e-mail client, IM client, spreadsheet (OpenOffice.org Calc), PDF reader (Evince), and the native file browser (Nautilus for Linux and Explorer for Windows)

Platforms. SnowFlock is implemented on the Xen [Barham et al. 2003] hypervisor version 3.4.0. Both the host and Linux VM use Debian Linux 5.0 with x64 version of the kernel 2.6.18.8. The VM was configured with 1GiB of RAM, a 12GiB disk image, and ran on a Dell Optiplex 745 desktop with 2.66GHz Intel Core 2 Duo CPU and 4GiB of RAM. The Windows 7 VM was configured with 4GiB of RAM and backed by a 16GiB disk image. This VM ran on Xen 4.2.0-rc4 on a Dell Studio XPS 7100 desktop with 3GHz quad-core AMD Phenom^{TM}II X4 945 processor and 6GiB of RAM.

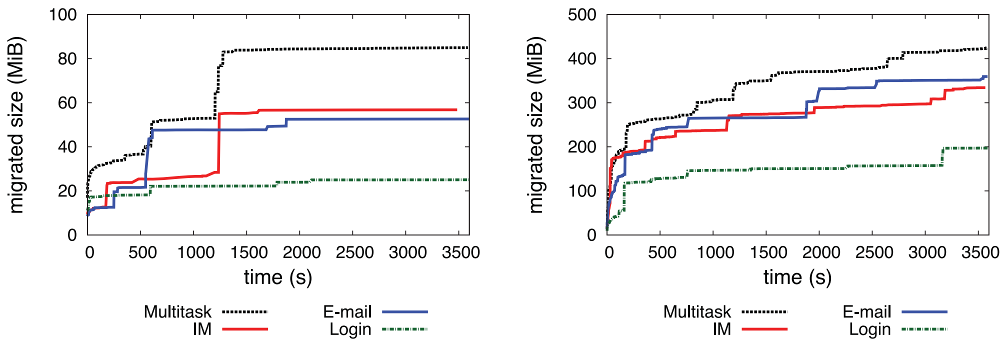
Workloads. Table II describes the workloads studied. The workloads consist of typical desktop applications. Login illustrates the nightly behavior of desktop systems whose users log out at the end of the work day. E-mail and IM are micro-workloads, consisting of a desktop session and the subject application (e-mail or IM client). For the Linux micro-workloads, we disabled the default GNOME desktop session (which provides task bars, desktop management utilities, screensavers, etc.) and ran the applications directly on the X Server. The micro-workloads are intended to give us a detailed look at the behavior of applications that maintain network presence with external hosts while idle. Multitask consists of a full-fledged desktop environment (GNOME in Linux), as well as the applications described in the table. It illustrates the behavior of the desktop of a multitasking office worker who has become idle temporarily and has not made any effort to quit any running application.

4.2. Deployment Study

Our VM consolidation software (presented in Section 6) was set up with a Linux VM on the desktops of three university researchers over a period of 7 weeks. Each user used his or her VM as the primary desktop over the duration of his or her participation. Our system monitored keyboard and mouse activity , and, when inactivity was detected for a period of at least 15s, a dialog warning of an impending migration was displayed for 5s. If no activity was detected during this warning period, the VM was migrated to the consolidation server. Otherwise, the dialog was discarded and the VM remained in the desktop.

Platform. Each user was given a desktop VM configured with 4GiB of memory and 12GiB of disk image hosted on the desktops. The VMs were configured with Debian GNU/Linux 5.0 with kernel version 2.6.18.8 for x86_64, the GNOME desktop system and desktop applications such as Mozilla Firefox Web browser, OpenOffice.org office suite, and the like. Users were free to install additional applications as needed. The VMs ran on top of the Xen 3.4.0 hypervisor.

The hardware consisted of desktop systems and a server. For desktops, we used three Dell Studio XPS 7100 systems. The server was a Sun Fire X2250 system with two quad-core 3GHz Intel Xeon CPUs and 16GiB of memory. The desktops connected to the server over a shared GigE switch.



(a) Linux VMs (b) Windows VMs

Fig. 4. Total page migration size for idle Linux and Windows 7 VMs with 4GiB of memory over 1 hour.

4.3. Working Sets of Idle VMs with Controlled Workloads

Figures 4(a) and 4(b) show the working set sizes of the idle Linux and Windows VMs, respectively, with workloads from Table II over a period of an hour. These figures show that idle Linux and Windows desktop VMs access less than 10% of their memory allocations over hour-long periods, even with different workloads. For Linux VM workloads, the Login workload accessed an average of 22.97 ± 1.94MiB of pages across all runs. E-mail, Chat, and Multitask accessed an average of 52.32 ± 0.37MiB, 55.69 ± 1.13MiB, and 91.01 ± 9.52MiB, respectively. For the idle Windows 7 VMs, the Login workload averaged 196.66 ± 0.77MiB of working set. E-mail workload averaged 372.80 ± 11.60MiB, Chat averaged 336.03 ± 12.66MiB, and Multitask averaged 381.04 ± 37.19MiB.

For both OS configurations, the traces show higher page transfer activity in earlier stages of the VM's execution on the consolidation server. More pages are migrated in the first half of each run than in the second half. This behavior is explained by the fact that, initially, when the VM begins execution on the consolidation server, none of the VM's pages is available on the server, and, as a result, any page access must be satisfied by migrating the page from the desktop. However, subsequent accesses to pages that have been migrated because of previous accesses are satisfied locally on the server. With the passage of time, a larger fraction of the VM's idle working set becomes available on the server, and the result is that fewer accesses cause page migrations.

We note that, from time to time, the VMs exhibit sudden bursts of page migration activity. These events are caused by running processes following a path of execution not previously followed during the run. For example, the Linux e-mail workload shows a large number of pages being migrated after nearly 10min. This coincides with the server poll interval of the e-mail client. IM shows a similar burst after 20min, which we believe to be caused by timer-related events of the application. Because Multitask contains both of these applications, it shows bursty migrations at both the 10th and the 20th minute of the run. The Windows VM shows more frequent bursts of migration activity because, unlike the Linux VM, its configuration does not disable built-in system tasks launched automatically on system start-up or user login, which cause their own page migrations.

4.4. Working Sets of Idle VMs During the Deployment

Results from our deployment confirm those from the controlled workloads, showing that idle desktop VMs access less than 10% of their memory allocation. Figure 5 shows the distribution of working set sizes of the three user VMs over 313 idle periods. Each time

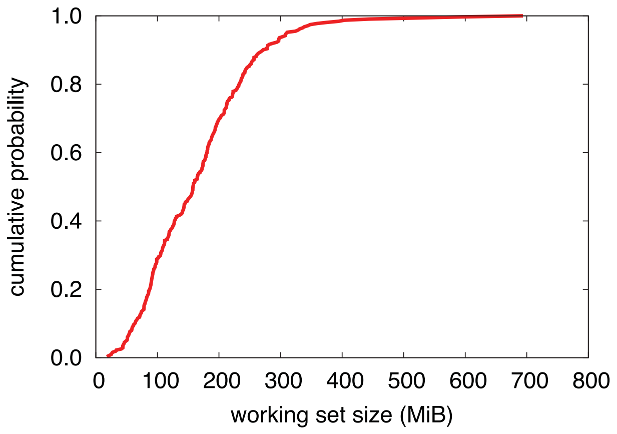


Fig. 5. Distribution of working set sizes of idle Linux VMs with 4GiB of memory.

the user was inactive, we consolidated the VM and recorded the size of pages accessed by the VM until the user returned. The mean memory working set was only 165.63 ± 91.38MiB, where ± denotes standard deviation. During these idle times, on average, 1.16 ± 5.75MiB of disk blocks were accessed. The implications of a small memory and disk footprint are that (i) little state needs to be migrated when consolidating, a benefit in terms of reduced network load; (ii) little state needs to be migrated when resuming, a network benefit but also, more importantly, an improvement of user experience by reducing reintegration latency; and (iii) limited memory needs to be committed to each running VM on the server, a benefit in reduced infrastructure costs.

These results confirm that migrating the full VM memory footprint, as described in Section 3.2, is inefficient. Instead, knowing that idle VMs only access small working sets, we can migrate only that working set for consolidation. Doing so ensures that migrations are fast, reduce network congestion, and limit the VM memory commitment on the consolidation server, increasing consolidation ratios. The difficulty, however, is in determining a priori which pages are part of the idle working set (and therefore need to be migrated). Because modern desktops consist of complex applications with nondeterministic execution [Ronsse et al. 2003], it is difficult to accurately predict which pages they will access each time the VM is migrated to the server. In Appendix A, we show that page access traces from the controlled desktop workloads on the Linux and Windows VMs present opportunities for the desktop to sleep between page requests.

In the next sections, we introduce partial VM migration, an approach that consolidates only the working set of idle VMs by migrating pages on demand, as they are needed on the server, and uses the observed distribution of page request interarrivals to make decisions about when it is energy efficient to place PCs in low power and what pages to prefetch during consolidation. We show with a deployment on desktops of four Linux users that partial VM migration is effective in reducing energy use of idle desktops.

5. PARTIAL MIGRATION OF IDLE DESKTOP VMS

Partial VM migration consolidates the working set of idle desktop VMs to allow user applications to maintain network presence while the desktop is in low-power state. Unlike full VM consolidation described in Section 3.2, partial VM migration does not migrate the VM's full memory footprint. When performing consolidation, partial VM migration first transfers the execution of the idle VM to a consolidation server by migrating CPU state, immediately halting execution on the desktop and starting it on the server. Then, as the VM executes on the server, it fetches on demand only the memory and disk state that is accessed by the VM. The VM's preconsolidation state remains as a residual on the desktop in anticipation of a reverse migration. When the user becomes active, partial VM migration transfers only the dirty state (memory pages and disk blocks that have been modified on the server) to the desktop and integrates it into the preconsolidation state. Because state is transferred from the desktop to the server on demand, the desktop only enters low-power state between transfers. We call these intervals *microsleeps*.

Partial VM migration does not require application modifications, the development of specialized protocol-specific proxies, or additional hardware. When the VM is executing on the desktop PC, the desktop has all of the VM's state, which provides full system performance to the user. When on the server, only the working set required for idle execution is available locally. By migrating only the idle working set, partial VM migration provides high consolidation ratios on the server and makes it possible to save energy by migrating often throughout the day without overwhelming the network infrastructure. Similarly, migrating back to the user's desktop is fast because only the dirty state created by the partial VM is reintegrated back into the desktop.

Partial VM migration leverages two insights. First, the working set of an idle VM is small, often more than an order of magnitude smaller than the total memory allocated to the VM. Second, rather than waiting until all state has been transferred to the server before going to sleep for long durations, the desktop can save energy by microsleeping early and often, whenever the remote partial VM has no outstanding on-demand request for state. Existing desktops can save energy by microsleeping for a few tens of seconds. Shorter intervals do not save energy because the transient power to enter and leave sleep state is higher than the idle power of the system. The challenge is to ensure that the desktop microsleeps only when it will save energy. In the next sections, we describe the policies used to inform the consolidation and reintegration decisions of idle desktop VMs. We then discuss microsleep policies used to decide when it is opportune for the desktop to sleep, and, in Section 5.3, we develop page prefetch strategies that maximize these microsleep opportunities.

5.1. Migration Policies

Partial VM migration automates migration of desktop VMs to a consolidation server when it detects opportunities to put the desktop in low-power mode. It migrates the VM back to the desktop once those opportunities are unavailable. Here, we describe the conditions for consolidation and reintegration of desktop VMs.

5.1.1. Consolidation Conditions. With partial VM migration, we consolidate a desktop VM when the following conditions are met:

(i) User inactivity. To avoid disruptions to the user, we migrate the desktop VM only when the user is not actively engaging the VM. We determine user idleness by monitoring keyboard and mouse activity. In the absence of activity, we provide an on-screen warning that allows the user to abort the consolidation.

(ii) VM idleness. The VM can execute on the server with sufficient autonomy from the desktop such that the desktop can sleep and save energy. This means that the VM must require few pages and disk blocks from the desktop. As a result, we must monitor memory and disk access rates periodically.

(iii) Server capacity. The server must have sufficient resources to accommodate the VM. Because the VM is idle during consolidation, the primary resource of concern is memory availability, and since the VM requires only enough memory for its working

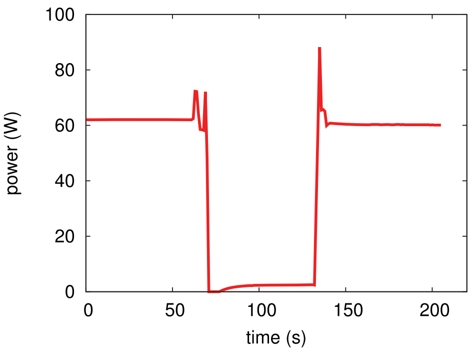


Fig. 6. Power used by a Dell Studio XPS 7100 desktop during various power states.

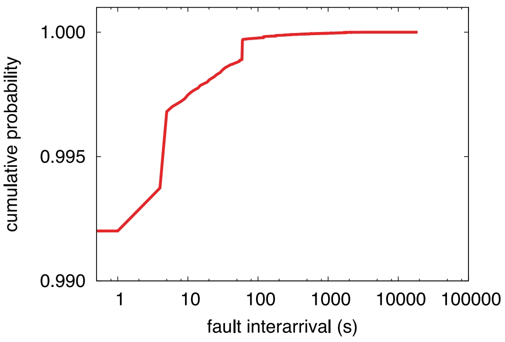


Fig. 7. Remote fault interarrival distribution of idle desktop VMs. The interarrival time for 99.23 of page requests is less than 1s.

set that is unknown at the start of consolidation, the server reserves memory based on an estimate. This estimate is made from observed sizes of VM traces and is adjusted each time the VM is consolidated. If during server-side execution the VM requires more memory than estimated and the server has enough free memory, it allocates it as needed. Otherwise, the server evicts the VM back to its desktop.

5.1.2. Reintegration Conditions. The decision to reintegrate a VM to the desktop is symmetrical to that of consolidating it to the server. It hinges on failures to meet idleness and server capacity conditions. That is, either (i) the user becomes active (e.g., the user presses a key on the keyboard), (ii) the VM becomes active and requires a large amount of state from the desktop (e.g., a scheduled virus scan walks through files on disk), or (iii) the server's capacity is exceeded (e.g., the VM requests memory when the server is fully committed). In addition, we may migrate a VM back to the desktop if (iv) the VM generates a UI event used to attract the user's attention. For example, when a VoIP client running on the consolidated VM receives a phone call and emits a ring tone through the sound device, the event can be intercepted and the VM migrated to the desktop to ensure that the user can hear the incoming call.

5.2. Microsleep Policy

Partial VM migration is designed to take advantage of lulls in page migration from desktops to consolidation servers by placing the desktops in low-power mode. However, deciding when desktops microsleep is not trivial because system power state transitions are costly. Transitions for microsleeps that end prematurely because of an incoming page request can cause the desktop to consume more energy than were it to remain powered. We design next an algorithm for determining when the desktop can microsleep. This algorithm seeks to minimize energy waste and uses the observed page request interarrival distribution of idle desktop VMs and the power profile of the desktop PC to reduce the probability of initiating a microsleep that is too short.

A desktop system experiences increased power use during transitions between low-power and full-power states. Figure 6 shows the power used by a desktop that is idle for 60s then transitions into low-power S3 mode. The desktop remains in this state for another 60s, after which it transitions again to idle state where it remains until the end of the experiment. These measurements were collected with a Watts Up? PRO power meter attached to a Dell XPS 7100 desktop system. The figure shows an increase in power use during the transition periods. The power peaks from 62W when idle up to 72W and 88W as the desktop enters and exits S3 mode, respectively.

The relatively high cost of entering and leaving low-power means that microsleeps only save energy if they last long enough to compensate for the transient rise in power used to enter sleep and wake the system to service the next remote fault.

Specifically, the energy use of an idle desktop system is given by:

$$E_i = P_i t_i, \tag{1}$$

where E_i is the energy used in watt hours, P_i is the system's idle power rate, and t_i is the idle time in hours.

The energy use of an idle system that microsleeps is:

$$E_\mu = P_i t_{i'} + P_o t_o + P_s t_s + P_r t_r, \tag{2}$$

where $t_{i'}$ is the portion of time the system remains powered, P_o and t_o the power rate and time the system spends entering sleep, P_s and t_s the power rate and time the system spends in sleep, and P_r and t_r the power rate and time the system spends exiting sleep.

Power rates t_o, and t_r depend only on the desktop's profile. In typical desktops, P_s is often an order of magnitude smaller than P_i, and P_o and P_r are larger than P_i. Then, microsleep can only save energy if t_s is long enough to compensate for increased energy use during t_o and t_r. The shortest interval for which it is energy efficient to microsleep is one in which $E_i = E_\mu$. In such intervals, the system wastes no time awake, so $t_{i'} = 0$, and the interval is given by:

$$t_b = \frac{-P_s(t_o + t_r) + P_o t_o + P_r t_r}{P_i - P_s}. \tag{3}$$

Plugging in the power profile measurements for the Dell Studio XPS 7100 desktops in Table III, we find that, for these systems, $t_b = 32.22$ s. Thus, these desktops should microsleep only when there is an expectation that no remote faults will arrive within the next 32.22s. t_b is smaller for desktops with lower power transition latencies, as we are able to achieve with context-aware selective resume in Section 7.

To determine the likelihood of a fault-free period of at least t_b length, we determine the conditional probability that the next remote fault will arrive in less than t_b as a function of the *wait time* (t_w), the time interval that has elapsed since the last remote fault arrived at the desktop. t_w tells us how long the desktop must remain awake after serving a page in order to avoid energy-inefficient microsleeps. More formally, $p(I < t_b + t_w | t_w)$ is the probability of interarrival I being energy inefficient.

Figure 8 plots the conditional probability that the next remote fault will arrive in less than 32.22s based on remote fault interarrival times shown in Figure 7 from our prototype deployment described in Section 4.2. Figure 8 shows that as the wait time increases up to 28s, the likelihood of seeing the next remote fault in less than 32.22s decreases rapidly. This decrease is because faults are highly correlated, and, as shown in Figure 7, more than 99.23% of remote faults occur within 1s of previous faults. The implication is that, for the vast majority of faults, when the desktop wakes to service one fault, it will likely be able to service faults that follow immediately, thus avoiding many inefficient microsleeps. With wait times immediately above 28s, the probability of seeing a remote fault increases significantly because 60s interarrivals are common, typically because of timer-based events.

We determine next the optimal value of wait time, t_w, that minimizes the energy waste as follows:

$$\min E_{waste}(t_w) = t_w E_i + p(I < t_b + t_w | t_w) E_\mu, \tag{4}$$

where E_{waste} is the total energy wasted. To compute E_μ, we assume the worst case, in which a fault occurs immediately after the desktop enters sleep so that $t_s = 0$ and $t_{i'} = 0$.

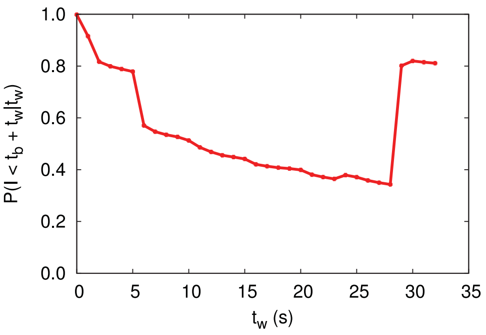


Fig. 8. Conditional probability that the next remote fault will arrive in less than 32.22s as a function of the wait time.

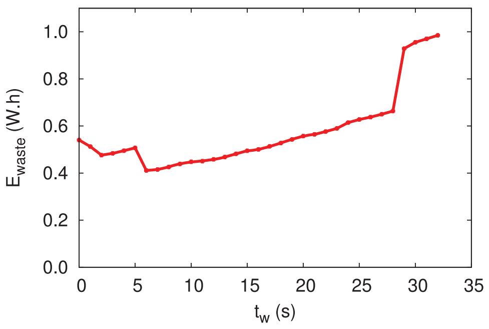


Fig. 9. Expected energy waste as a function of sleep timeout (t_w). With t_w values that are short, energy is wasted by sleep interruptions from page requests. Longer values waste energy by keeping the desktop in full-power.

With our desktops' energy profile, E_{waste} is shown in Figure 9, and it is clear that the energy waste minimizing t_w is 6 s.

5.3. Memory Prefetch

We use prefetching to increase the frequency and length of energy-efficient inter-arrivals. Prefetching proactively migrates state to the server and allows faults to be serviced locally on the server, thus not requiring the desktop to be awake.

Most state transfer (in excess of 99%)—and hence remote faults—is caused by memory accesses (see Section 4.4). As a result, we concentrate our efforts on reducing memory faults and allow disk requests to be serviced on demand, independent of whether storage is local or networked.

We explored two prefetch strategies. The first, *hoarding*, explores similarity in page frame numbers accessed between different migrations of the same VM. At the time of consolidation, this approach fetches a sequence of pages whose frame numbers were requested in previous instances in which the VM was consolidated. In the second prefetch strategy, *on-demand prefetch*, we exploit spatial locality of page accesses by using a pivot window to prefetch pages whose frame numbers are near a requested page. Both strategies fetch pages into a per VM buffer, either in disk or in a discrete memory location, and pages are only committed to the partial VM's memory when the VM attempts to access them. This approach ensures that prefetching does not grow the memory footprint of an idle VM, and, whenever the prefetch buffer is full, it can evict pages that are unlikely to be used.

Figure 10 shows how our prefetch strategies improve average microsleep durations as a function of the total memory migration size. Memory migration size is composed of pages migrated by the prefetch strategy and pages migrated due to a prefetch cache miss. To produce the figure, we developed a simulator that runs through page access traces we collected in our deployment and allows us to impose indirectly different limits to the size of memory migrated. For on-demand prefetch, we vary the prefetch pivot window, and, for hoarding, we vary the hoarding limit. The figure is based on page access traces from a user VM that was consolidated 58 times. The figure shows that both approaches reduce the number of interruptions to microsleeps, with on-demand prefetch performing better. Hoarding increases the average microsleep duration from 83.62s to 136.64s. On-demand prefetch increases it to 150.82s.

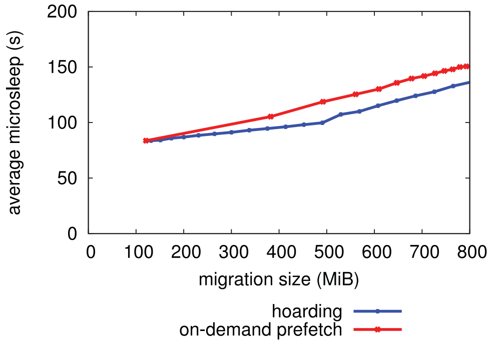


Fig. 10. Increased microsleep durations from prefetching. For a given memory migration size, prefetching increases microsleep durations.

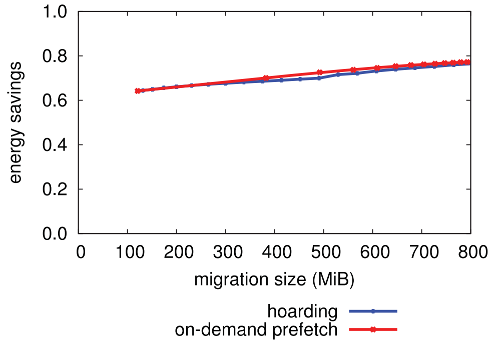


Fig. 11. Increased energy savings from prefetching. The strategies deliver similar improvements.

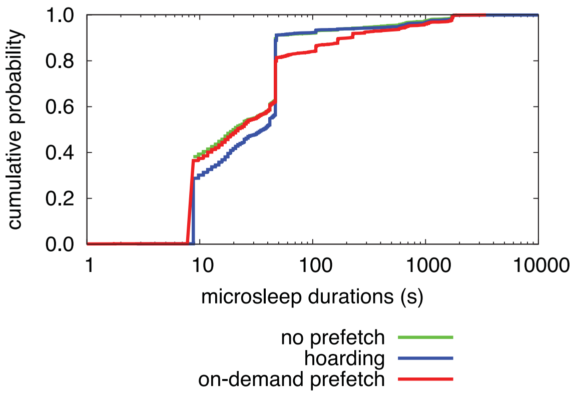


Fig. 12. CDF of microsleep durations. Hoarding increases duration of short microsleeps most, whereas on-demand prefetch increases duration of long microsleeps most.

Figure 11 shows the energy savings of the desktop as a function of the total memory migration size for the VM of the previous figures. We use the simulator to compute the energy use of the desktop by aggregating energy used over each interval the desktop would be idle, suspending, sleeping, and resuming using Equation (2). The energy savings are normalized over the energy the desktop uses when left powered during the same idle periods. The figure shows that, as a result of the improved microsleep durations, energy savings also improve. The two strategies deliver similar energy savings because, as we show next, whereas on-demand prefetch is effective at increasing the frequency of long microsleeps, hoarding increased the duration of short microsleeps, both of which lead to increased energy savings.

Figure 12 presents a CDF of microsleep durations for both strategies, with settings that migrate close to 500MiB of memory (on-demand: $pivot\ window = 20\ pages$, $migration\ size = 492.19\ MiB$; hoarding: $limit = 480MiB$, $migration\ size = 490.83MiB$). The figure shows that whereas hoarding reduced the incidence of energy-inefficient microsleeps (<32.22s) from 55.51% to 48.43%, on-demand prefetch led to more microsleeps with long durations. Without prefetching, 10.34% of microsleeps had durations of longer than 47s. Hoarding increased these to a modest 11.11%. On-demand prefetch doubled the long microsleeps to 20.13%.

We also experimented with refinements to the prefetch strategies. First, we combined the strategies to first hoard a fixed set of pages and, when cache misses occur, prefetch on-demand pages with locality. We found this to yield an improvement in energy savings for any given migration size of 1%–4% over on-demand prefetch alone. An attractive feature of on-demand prefetch is that it does not create bulk network transfers, which can quickly congest the network. Rather, it amortizes memory migration over the duration of the consolidation as pages are needed by the VM. Given similar energy savings performance among the three strategies, on-demand prefetch is preferable. Second, we explored maintaining VM residuals on the server and found that it can reduce network transfers by an additional 28%. When a VM migrates back to the desktop, the server can retain the pages it has previously received in a side cache in anticipation of a future consolidation of the same VM. Cached pages are invalidated whenever updates are made when the VM runs on the desktop. When the VM is consolidated again, page faults only need to be serviced from the desktop when the pages are not found in the cache. We considered page sharing across consolidated VMs and expect it to reduce the VMs' memory footprints by up to 28%; we anticipate a desktop VM to have more pages in common with itself over time (server residuals) than in common with other VMs.

In subsequent sections, we use on-demand prefetch for memory migration and fix the prefetch pivot window size at 20 pages, which we found to deliver the highest savings per MiB. We also fix the prefetch cache at 50MiB because prefetched pages are commonly used within a short period, and, with such buffer, we see little cache thrashing because of page eviction. Our implementation of partial VM migration does not retain VM residuals on the consolidation server, which could provide additional benefits in energy savings and reduced network traffic.

6. IDLE DESKTOP MIGRATION WITH JETTISON

In this section, we present details of Jettison, our implementation of partial VM migration. We then quantify the energy savings and network performance of the implementation with a deployment of Jettison in a research environment and use traces of user idleness we collected in a corporate research lab to estimate the performance of partial desktop VM migration in large enterprise offices. Finally, we discuss the challenges facing the adoption of partial VM migration.

6.1. Jettison Prototype

Jettison extends the functionality of Xen 3.4 hypervisor [Barham et al. 2003]. Xen is a type 1 or native [Goldberg 1973] hypervisor that runs at the highest processor privilege level and relegates guest domains to lower privileged ones. Xen supports an administrative guest domain (dom0) and multiple unprivileged guest domains (domUs). In our architecture, desktop environments are encapsulated in domUs.

Jettison is implemented as modifications to the hypervisor, daemons in dom0, and a page migration avoidance patch in the kernel of the domU. Our implementation currently supports paravirtualized guests, and we leave support of fully virtualized VMs needed for unmodified guests such as Windows for future work. In the discussion to follow, we explain where changes are needed for fully virtualized VMs. Jettison runs a number of components both on the desktops and the consolidation server, as shown in Figure 13. On the dom0 of the desktop systems, Jettison runs the following components:

(1) *memsrv.* A daemon responsible for serving VM memory pages to the consolidation server over TCP. It maps all of the VM's frames with read-only access and runs only when the VM is consolidated.

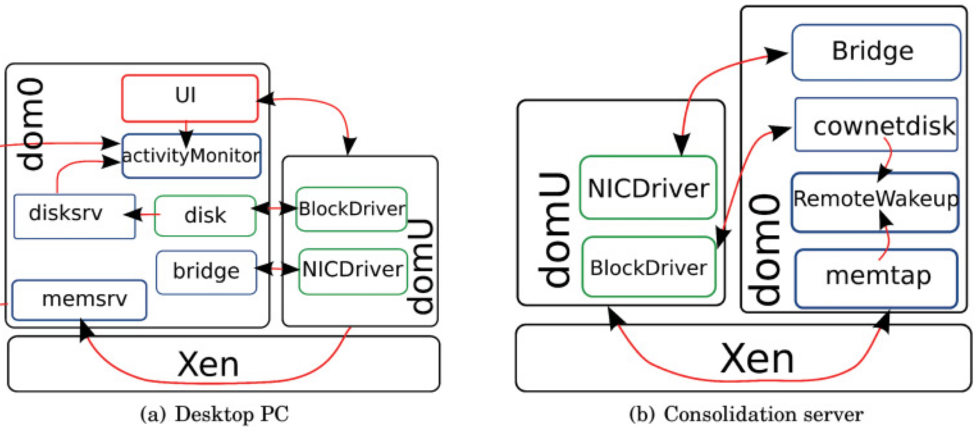


Fig. 13. Architecture of Jettison. The desktop runs an activityMonitor daemon that coordinates VM migration and PC sleep in response to detected UI events and memory and disk requests. On the consolidation server, the memtap daemon services faults by fetching from the PC.

(2) *disksrv*. A daemon responsible for serving disk blocks to the server over TCP. disksrv opens the VM's disk image in read-only mode.
(3) *activityMonitor*. A daemon responsible for detecting user activity, initiating both consolidation and user-triggered reintegrations, maintaining the microsleep wait timer t_w introduced in Section 5.2, and suspending the desktop system to low-power state S3 (suspend-to-memory). At present, activityMonitor monitors keyboard and mouse and, after a user-configured period of inactivity, provides an on-screen warning for another preconfigured period before consolidating the VM. activityMonitor also receives notifications from memory and disk servers whenever a page or disk block request is received and resets t_w accordingly. Although we are able to extract CPU and I/O utilization statistics from the hypervisor, we did not implement a mechanism for detection of memory, I/O-, or CPU-bound tasks, which would cause VMs to remain on the desktop. We leave its implementation to future work.
(4) *memrestore and diskmerge*. Utilities that integrate memory and disk state received from the consolidation server. memrestore maps the VM's memory frames and updates them with the pages received.

On the consolidation server's dom0, Jettison runs the following components:

(1) *memtap*. A process that monitors VM page faults, notifies remoteWakeup (described later), issues page requests to memsrv running on the desktop, and, upon receiving the page, updates the VM's page frame. One memtap process is instantiated for each consolidated VM. memtap maps its VM's frames with write access so it can perform direct updates. To support memtap, the hypervisor page fault handler code maintains a bitmap of migrated pages, and, on a fault, it forwards notifications to memtap using Xen event channels (Xen's own IPC mechanism), suspends the faulting vCPU execution, and, on receipt of the page, reschedules the vCPU. The migration avoidance code in the domU ensures that overwriting a page (page allocation) does not cause a notification to memtap.
(2) *cownetdisk*. A block device driver we built to migrate blocks on request and track block updates by the consolidated VM. Xen employs a split-device model in which a device front-end interface runs in the kernel of domU and a back end, implementing the functionality of the device, runs in dom0. cownetdisk is our instantiation of the block device back end for consolidated VMs with desktop local storage. It is based

on the blocktap interface [Warfield et al. 2005] and implements a copy-on-write networked disk device. While a VM runs on the server, cownetdisk maintains two sparse virtual disk slices as files in dom0. The bottom slice, the read-only slice, keeps blocks that have only been read by the VM, and the top slice, the dirty slice, maintains all that have been written to, which will be migrated to the desktop during reintegration. A bitmap is used to identify the valid slice for a block. Read requests for blocks not in the server are fetched from the desktop and placed in the read-only slice. First writes to a block cause the promotion of the block to the dirty slice. Once a block is promoted, all future accesses occur in the top slice. Writes to blocks not present in the server cause a fetch from the desktop first and an immediate promotion. The exception is whole block writes, which cause only a promotion. On a remote fetch, cownetdisk also notifies remoteWakeup first to ensure that the desktop is awake.

(3) *remoteWakeup*. A daemon responsible for waking a sleeping desktop via Wake-on-LAN [Lieberman Software Corporation 2006] whenever remote state is required. This daemon abstracts the management of the power state of desktops consolidated from the memory and disk clients. Before the memory client (memtap) or the disk client (cownetdisk) request a page or disk block from the desktop, they notify the local remoteWakeup daemon on the consolidation server, providing the IP address of the desktop they wish to access. The remoteWakeup daemon maintains an IP-to-MAC address translation table and the power state of all desktops whose VMs are consolidated. The translation table is used to construct the Wake-on-LAN packet, and the power state of the desktop is used to decide whether to send a Wake-on-LAN packet without first sending a network probe. The state of a desktop is determined with a timestamp recorded when the last request was sent to the desktop. When a new request is made, remoteWakeup assumes that the desktop is asleep whenever the timestamp is elapsed by a period of at least t_w seconds, the period for which the desktop is configured to wait before going to sleep. Both memtap and cownetdisk are configured to resend their requests to remoteWakeup whenever the desktop fails to respond.

To support our migration policies, we also monitor the VM's I/O and CPU usage. Xen already maintains these statistics, which we can access to determine that the VM is idle. We can determine memory usage periodically via the hypervisor's *shadow page tables*. Using shadow page tables, Xen can initially mark all pages of a VM read-only, and an attempt by the VM to make a write is trapped by the hypervisor, which sets a dirty bit for the page.

What happens during consolidation. On the desktop, the execution of the VM is halted, and our dom0 tools generate a VM descriptor and all memory state of the VM remains in core. The descriptor contains VM configuration metadata, such as device configuration, vCPU register state, page table pages, and configuration pages shared between the domain and hypervisor. The largest component of the descriptor is the page table pages. The descriptor is migrated to the server, which creates a new domain and begins its execution. A disksrv process and a memsrv process are instantiated, and device back ends are disconnected from the halted VM. Whenever these state servers receive a request, they notify the activityMonitor so it knows not to schedule an immediate sleep of the desktop.

As the VM begins execution on the server, it faults on page accesses. These faults generate an interrupt handled by the hypervisor. In turn, using an event channel, Xen's interdomain communication interface, the hypervisor notifies the memtap process of the fault and suspends the faulting vCPU. When memtap has received the page and updated the VM's frame, it notifies the hypervisor via the same event channel. The hypervisor then reschedules the faulting vCPU for execution.

What happens during reintegration. When the VM resumes execution on the desktop—for example, because the user has returned—any state that was modified while it ran on the server needs to be integrated into the desktop state. Because the desktop contains all of the VM's state and only a fraction of it has become stale, only the new state is migrated back. For this, we use the disk and memory dirty state tracking mechanisms described earlier. When the VM is reintegrated, only pages and disk blocks marked as dirty are migrated to the desktop.

On the consolidation server, the VM is halted, and our dom0 tools map in dirty memory frames and vCPU register state and send their contents to the desktop. The dirty disk slice, if any, is also sent. On the desktop, the VM's memory frames are mapped with write permission by our dom0 tools, which update them with received dirty state. In parallel, the dirty disk slice is merged with the local disk. Once all state has been updated, device back ends are started, and the VM is allowed to begin execution.

Network migration. Network migration is supported within LAN environments where both the desktop and the server are in the same Layer 2 broadcast domain. In these environments, because Jettison VMs rely on host network bridging and maintain the same MAC address across hosts, they continue to receive network packets that are destined to them after migration. When adding the VM's virtual NIC to its Ethernet bridge, the host sends an ARP probe that notifies network switches to direct frames with matching MAC addresses to itself. This allows existing connections to remain active with minimal latencies during migration. When the desktop and server connect via a Layer 3 or higher network device, the device must ensure that both are in the same broadcast domain. For example, a router must include both the desktop's and the server's subnets within the same Virtual LAN.

Dynamic memory allocation. Dynamic memory allocation is achieved by allocating on demand the underlying pages of memory of a consolidated VM. For paravirtualized guests, we realize on-demand allocations through the concept of a "ghost MFN." Xen uses two complementary concepts to address a page frame. Machine Frame Number (MFN) refers to the machine address of the frame, as viewed by the MMU. Physical Frame Numbers (PFNs) are indirect addresses given to the paravirtualized VM kernel to refer to the real MFNs. PFNs give the VM the illusion of having access to a contiguous address space. A ghost MFN has the property of serving as a placeholder that encodes the PFN that it backs and as a flag indicating absence of actual allocation. The ghost MFN is placed in lieu of an allocated MFN in the page tables and the PFN-to-MFN translation table that each Xen paravirtual guest maintains. The first guest access to the PFN triggers a shadow page fault in the hypervisor, which is trapped and handled by allocating the real MFN to replace the ghost. We limit fragmentation of the host's free page heap by increasing the granularity of requested memory chunks to 2MiB at a time while still replacing ghost MFNs one at a time.

Although we have not implemented dynamic memory allocation and remote fault handling for fully virtualized guests, known as *Hardware Assisted VMs* (HVMs), we plan to use Xen's built-in populate-on-demand (PoD) mechanism to do both. PoD maps PFNs to MFNs on demand for HVMs by faulting on first access to each page. This mechanism is used to boot HVMs with lower memory commitments than their maximum reservation. PoD allocates a preset target memory to a per-guest cache and maps pages to the guest's memory on demand. When the cache runs out of pages, PoD scans the memory of the guest for zero pages, then unmaps and returns them to the cache. For our purposes, we modify the PoD cache so it starts with a small chunk of memory, and, when it runs out of pages, instead of scanning for zero pages, it gets additional allocation chunks from the hypervisor, as we do for our ghost MFN implementation.

We note that, in both approaches, the faults used to allocate memory on demand are the same as those we use to fetch missing state on demand. That is, when a fault occurs, first, we commit the backing frame to the VM, and we then notify memtap to fetch the content from the desktop.

Security. Migration of desktop memory and disk state over the network introduces new risks of exposure of private user data. First, network hosts can passively listen to and capture memory and disk state that is transmitted over the network between desktop PCs and the consolidation server. Second, unauthorized hosts may actively communicate with memory and disk servers on the desktops or the consolidation server either by directly establishing a connection to the server or by redirecting requests of a legitimate client in a *man-in-the-middle attack*. Preventing unauthorized network access to memory and disk state requires a robust implementation of end-to-end authentication and encryption of network data. We envision the use of a secure network protocol for state migration, such as the Transport Layer Security protocol (TLS) [Dierks and Rescorla 2008]. TLS is particularly suitable because it is an application layer protocol widely used to authenticate the endpoints and encrypt the streams for the transport layer. We leave the implementation of network security on Jettison for future work.

6.2. Experimental Evaluation

We evaluated the performance of Jettison with a deployment that involved four users and lasted 6 days. We use the results of this deployment to answer the following questions:

(1) How much energy is saved by partial VM migration?
(2) Does microsleeping save energy?
(3) How much state needs to be migrated to the consolidation server to run an idle VM?
(4) How much data need to be migrated back to the desktop when reintegrating the VM?
(5) How long does it take to migrate a consolidated VM back to the desktop?

6.2.1. Experimental Setup. Our deployments employed desktop systems and a consolidation server. When the users were active, VMs ran on the desktops. When inactive, the VMs migrated to the server. Jettison was configured to consolidate VMs when no keyboard or mouse activity was detected for a period of 15s. Once a VM was found to be inactive, an on-screen warning was displayed for 5s before migrating the VM to the consolidation server, thus allowing a nearby user to cancel an impending consolidation if needed.

Each VM was configured with 4GiB of memory and 12GiB of disk. The VMs ran Debian GNU/Linux 5.0 with kernel version 2.6.18.8 for a 64-bit x86 platform, the GNOME desktop configured with Mozilla Firefox Web browser, Mozilla Thunderbird e-mail client, OpenOffice.org office suite, Pidgin IM client, and OpenSSH daemon, among others. Users were free to install additional applications as needed. Background IM and e-mail traffic was often present, including occasional delivery of messages. Some of our users used the IM client to connect to Google Talk and used Thunderbird for e-mail, whereas others used web-based Gmail and chat. Our VMs were also accessible via SSH, and some users reported downloading documents from their consolidated VMs from home. Such activities did not require VM reintegration because they did not cause high I/O activity or significant growth of VM memory.

Table III. Power Profile of Dell Studio XPS 7100 PC

State	Time (s)	Power (W)
Suspend	8.38 ± 0.22	107.90 ± 1.77
Resume	8.58 ± 0.85	121.72 ± 24.52
Idle	N/A	61.40 ± 0.03
Sleep (S3)	N/A	1.95 ± 0.02
Network	N/A	136.63 ± 2.81

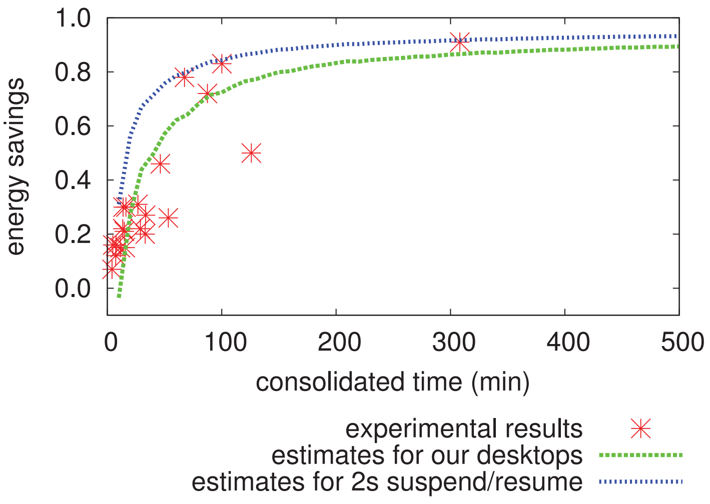


Fig. 14. Recorded desktop energy savings along with estimates showing with fine granularity the savings a desktop can expect as a function of the duration of the idle period.

The desktops were Dell Studio XPS 7100 systems, with a 3GHz quad-core AMD Phenom II X4 945 processor and 6GiB of RAM. Table III presents the desktop's power profile obtained with a GW Instek GPM-8212 power meter. These numbers are comparable to those of other published systems [Agarwal et al. 2009; Das et al. 2010]. The server was a Sun Fire X2250 system with two quad-core 3GHz Intel Xeon CPUs and 16GiB of memory. Its idle power averaged 150.70W. The desktops connected to the server over a GigE switch shared with approximately 100 other hosts. We measured the effective throughput between the desktops and servers to be 813.44Mbps. Power use of the desktops during the deployment was measured with Watts Up? PRO power meters.

6.2.2. Energy Savings. Figure 14 shows energy savings experienced by desktop users during the deployment. Energy savings are normalized over the energy these desktops spend if left powered during those idle periods. The figure also shows two estimates. The first is a finer grained estimate of expected energy savings for the desktops used in the deployment over varying lengths of consolidation time. The second estimate computes the energy savings with a desktop with suspend and resume times of just 2s (faster than the 8.5s of our desktops.) These estimates were computed from memory access traces, as described in Section 5.3. The estimate for the existing desktops match our experimental data well.

The experimental results show that our users were able to see reductions on their desktop energy use from idle periods as short as 4min. While short idle times of under 10min show savings of 7%–16%, in longer idle times the savings were significant. In

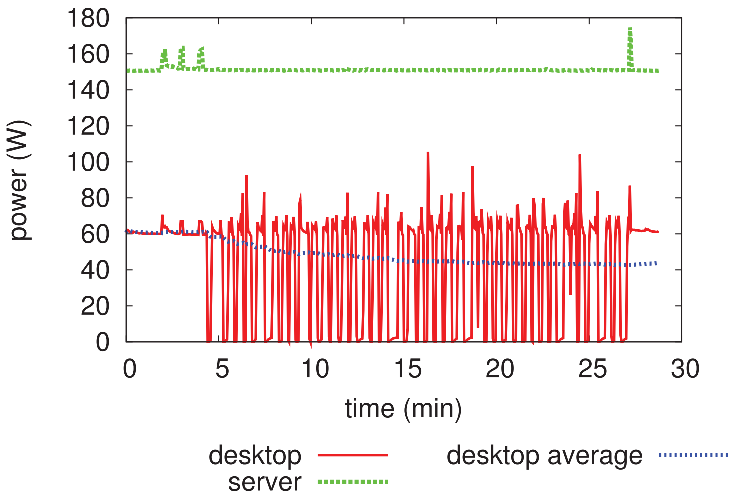


Fig. 15. Power usage of a desktop and server during partial migration. The figure shows a reduction in the average energy use over time.

idle times of 67min, we see 78% savings, and in idle times of 308min, we see even higher savings of 91%.

In too-short idle times, the energy expended by the desktop going to sleep and waking up is not offset by the energy savings of the short microsleeps available because the desktops we use have very slow suspend and resume times (nearly 8.5s). Still, we can see that, in sufficiently long idle times, where faults are extremely rare (not recorded), the energy savings converge toward $(P_i - P_s)/P_i = 96.82\%$.

The estimate for a desktop with 2s suspend and resume times simulates the performance on next-generation PCs with faster system power state transitions. This estimate shows improved energy savings, especially in short idle times. Intervals under 10min show savings that are closer to 30%. Several computers already provide resume times close to 2s, including the MacBook Air [Apple, Inc. 2012] and Acer Aspire S3 [Acer, Inc. 2012], and we show in Section 7 that such latencies are achievable even with existing desktop PCs.

Figure 15 shows detailed power usage of one desktop and the consolidation server over a 30min period in which the VM is consolidated to the server for 25min. The figure shows the power usage patterns as the desktop performs microsleeps. At 1min 57s, the VM begins migration to the server. This event is represented by the first spike in power use in both the desktop and server. As the server runs the VM, we note two additional spikes as batches of pages are fetched for the VM. From 4min 21s, the desktop performs a series of microsleeps until VM resume time. While initially the energy use of the desktop nominally exceeds its idle use, as soon as the first microsleeps take place, the average energy use of the desktop drops below idle, and it continues to drop over the course of the idle period. As a result, the average power use of the desktop over the idle period drops from 61.4W to 43.8W, a reduction of 28.8%. The energy savings of the desktop and the length of each microsleep increase over time.

Although the server adds to energy use over an environment in which no consolidation is performed, it is worth noting that with our VM's working set sizes, each server is capable of hosting at least 98 VMs, so its power use per VM amounts to less than 2W. This power can be driven down further by increasing only the memory capacity of the server.

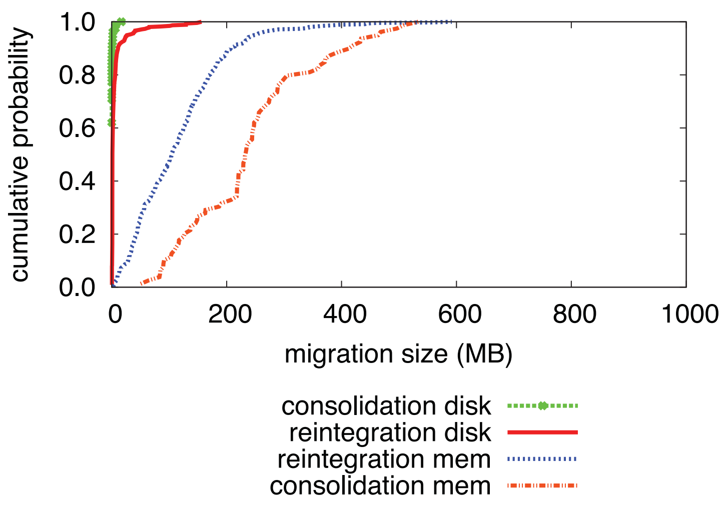


Fig. 16. Distribution of migration sizes for VMs with 4GiB of memory.

6.2.3. Network Load. Figure 16 shows the distribution of disk and memory state migrated during consolidation and resume stages. The results show that partial VM migration makes frugal use of the network. Overall, the average size of memory migrated (including data migrated by on-demand prefetching) to the consolidation server was 242.23MiB, a mere 6% of the VMs' nominal memory. The migration avoidance optimization, which ensures that pages being overwritten in the partial VM are not migrated, avoided fetching an average of 9.06 ± 25.94MiB.

The average size of migrated disk state to the consolidation server was much smaller at 0.50MiB. Similarly, on average, each VM migrates 114.68MiB of memory and 6.81MiB of disk state back to the desktop, thus confirming that VMs do not generate much dirty state while idle. This dirty state is generated by all processes that run in the VM independent of user activity, including always-on applications, but also tasks that run periodically, such as OS daemons and user tasks (e.g., browser JavaScript).

6.2.4. Migration Latencies. User-perceived latency is important because it directly affects the user's experience and his willingness to accept any approach that relies on migration. Of particular importance is *reintegration latency*, the time it takes for a consolidated VM to migrate to the desktop and resume execution on user request. Users of VM migration solutions will wait first for the desktop hardware to resume from low power and then for the VM to migrate back to the desktop. Our experiments show that partial VM migration delivers small migration times. On average, VMs migrate to the desktops in 4.11s. Similarly, the average time to consolidate a VM is 3.78s. The bulk of waiting is needed for the desktop to resume from low-power. For our desktops, resume times equal 8.58s. Best-in-class machines, such as those discussed in Section 6.2.2, can deliver resume times of about 2s. Hardware resume times affect all energy-saving approaches that use low-power states.

6.3. Challenges with Partial VM Migration

Although the results presented show that partial migration of desktop VMs achieves substantial energy savings with high consolidation ratios and low migration latencies, several challenges emerge from its use of microsleeps, which cause frequent transitions between power states on the PC.

(i) Limited energy savings during short idle periods. Power state transitions reduce total sleep time and energy savings of the PC. As discussed in Sections 5.2 and 5.3, the PC tries to reduce the number of power state transitions by servicing consecutive page requests at a time and prefetching likely page candidates for future requests before returning to low power. Even so, results from our deployment of Jettison, presented in Section 6.2, found that, during short idle times when faults are frequent, the energy savings are low. For example, although Jettison achieved savings of up 78% in idle periods lasting an hour, in idle periods of about 30 min, savings ranged from 20% to 31%. In idle periods of under 10min, energy savings ranged from 7% to 16%.

Over what idle periods to consolidate? Short idle periods present considerable opportunities to conserve energy. For example, the traces of desktop user activity presented in Section 2.1 indicate that idle periods of under an hour correspond to 17% of the total idle times (including overnight hours). In taking advantage of short idle periods, a balance must be reached between energy conservation and ensuring that the impact on user experience is minimal. We looked at best practices from industry for guidance on the duration of idle periods that are attractive for energy conservation. The Energy Star program mandates that compliant PCs be configured to switch the display off within 15min of user inactivity and transition into low-power mode within 30min [U.S. Environmental Protection Agency 2013]. The U.S. Department of Energy recommends that users actively turn off their monitors whenever they will be away from the computer for more than 20min and turn off the computer when away for at least 2 hours [U.S Department of Energy 2013]. The default power management plan for Windows 7 PCs (the balanced plan) turns the display off after 10min of inactivity and enters low-power mode after 30min. For the Mac OS X 10.6, these defaults are both of 10min. These policies make a distinction between power management modes that allow applications to run (turning off the monitor) and those that do not (PC off or sleep mode). Because partial VM migration enables applications to run during sleep modes, we apply the policies designed for modes in which applications can run. While the various policies and guidelines dictate different thresholds of inactivity for engagement of power management, the apparent consensus is between 10 and 20min. We must, therefore, ensure that partial VM migration can deliver savings, even during idle periods of 10min. As we have shown so far, the savings have been modest during such periods.

(ii) Reduced hardware reliability. The second challenge is that frequent power state transitions may lead to reduced life span of hardware components not designed for frequent power cycles. The potential impact on the life span of system components may arise particularly because, on system wake-up, current desktops power up all devices, independent of whether they are required. For most instances of system wake-up in partial VM migration, the majority of devices are not needed. Indeed, most desktop wake-ups require only access to CPU, memory, and network card.

(iii) Long response times. The third challenge is that slow PC wake-up times reduce responsiveness of applications running in the consolidation server during faults, and, for networked applications, this may cause unintended side effects on connections. In terms of software reliability, the desktop applications we used were able to contend gracefully with the increased latency required to fulfill a remote fault when the desktop is microsleeping. In our experience, applications that rely on TCP and do not expect real-time network performance can function reliably even during short absences of an end point. For example, the default Linux configuration for TCP allows for retries for up to 13–30min which, in our usage of the system, has proved far more than sufficient to deal with the 8–17s remote fault latency in the worst case.

In Section 7, we present context-aware selective resume, a solution that addresses many of the problems caused by on-demand memory migration. It bypasses initialization of hardware and software components that are not relevant to handling page faults during PC wake-up. This approach allows the application causing the wake-up to specify the context for the wake-up request anytime it wakes the PC. It has the dual benefits that wake-ups are fast, thus increasing energy savings, and most hardware components are not reinitialized for on-demand page migration.

6.4. Summary

In this section, we presented Jettison, our implementation of partial migration of idle desktop VMs. Jettison extends the functionality of the Xen hypervisor and enables idle desktops to save energy while the consolidated VM runs in a shared server. It migrates state on demand and allows the desktop to microsleep when not serving requests. We presented results from our deployment that show that Jettison provides significant energy savings with the dual benefits that network and server infrastructure can scale well with the number of users, and migration latencies are very small. We showed that a desktop can achieve energy savings of 78% in an hour of consolidation and up to 91% in longer periods while maintaining migration latencies of about 4s. Finally, we discussed the challenges of the approach, which we address in the next section.

7. EFFICIENT ON-DEMAND MIGRATION OF PAGES WITH CAESAR

On-demand migration of pages enables partial VM migration to limit network traffic and deliver high consolidation ratios on the server. At consolidation, it transfers only VM configuration and vCPU context necessary to initiate execution of the VM on the server. VM execution on the server leads to accesses to memory pages (and disk blocks) that are not present—remote page faults. The hypervisor traps these faults and fetches the pages from the PC. To do so, the hypervisor must wake the PC from low-power mode, retrieve the page, and instruct the PC to return to low-power mode. The result is PCs that cycle from low-power to full-power and back to low-power mode dozens of times per hour.

Our experience with Jettison, our partial VM migration prototype, has shown that energy saving ACPI power states have not been designed for frequent transitions. Although the ACPI standard makes provisions for low-latency states such as S1 and S2, these have little effect in reducing energy use and are not widely available. Transition times between energy saving states (S3, S4, and S5) are slow, lasting up to tens of seconds (Section 2.2) and result in high rate of power use (Section 5.2).

In a previous study [Wright et al. 2011], we found that 87% of resume-suspend cycle times on commodity PCs is taken by OS and BIOS activity, and only 13% comprises hardware suspend and reinitialization. *Resume-suspend cycle time* is the time taken by PCs to resume from low-power state, send a network packet, and return to low-power state, as is done to service page faults of VMs that have been partially consolidated. When suspending into S3 low-power state, the OS iterates through devices and saves their state in DRAM. When resuming, first, the BIOS identifies system devices and performs initializations of low-level devices, such as system chipsets and CPUs. Subsequently, the OS iterates through peripheral devices, restoring their presuspend state. Our observation is that applications providing intermittent services require functionality from only a few devices and exercise only a small fraction of the OS code. Therefore, suspend and resume times can be reduced by saving and restoring state of only those OS subsystems and devices required to provide the service. We present *context-aware selective resume*, an approach that selectively suspends and resumes devices and OS subsystems based on the wake-up context. The wake-up context informs the PC of the service causing the wake-up. The context for partial VM migration informs the PC on wake-up that only the memory server functionality is needed, and, as a result, the PC needs only to reinitialize DRAM, CPU, and the network interface.

In this section, we introduce context-aware selective resume and show that it reduces the cost of on-demand page migration (the energy cost of taking the PC out of low-power mode.) High cost of on-demand page migration makes saving energy from consolidations over short intervals (when faults are frequent) only a modest success. We introduce CAESAR, our implementation of the context-aware selective resume framework, and experiment with a memory server built on CAESAR. CAESAR reads the wake-up context from the Wake-on-LAN packet payload sent by the consolidation server; initializes the CPU, DRAM controller, and NIC interface; and invokes an appropriate context handler. For page migration, this handler is the memory server, which sends the requested page and returns control to the framework. CAESAR then returns the PC to low-power state.

Our experiments show that context-aware selective resume effectively reduces resume-suspend cycles by 65%, from 9.0s to 3.2s, and increases desktop sleep time over an hour-long consolidation by 26%–103%. As a result, context-aware selective resume increases energy savings over the full resume approach throughout the hour-long interval. For consolidations lasting 5 min, it increased energy savings by 13%–66%. In 10min, the increase was by 43%–54%. In 20min, it increased savings by 38%–49%, and finally, over an hour, the approach increased energy savings by 14%–65%.

The remainder of this section is organized as follows. In Section 7.1, we discuss the architecture of the context-aware selective resume framework and present our implementation of a fast memory server. In Section 7.2, we present experimental results demonstrating the benefits of context-aware selective resume when migrating memory on demand. Finally, in Section 7.3, we conclude the section.

7.1. The Context-Aware Selective Resume Framework

Context-aware selective resume expedites resume-suspend cycle times of desktop PCs. When an event causes the PC to wake from a low-power state, the firmware is notified of the context or cause of the wake-up, and, based on this context, the firmware re-initializes only the necessary devices and invokes code relevant to the context. Applications that require context-aware functionality when the PC enters low-power mode install context vectors with the context-aware selective resume framework. Each context vector maps a numerical context ID to the memory address containing the context handling code.

While we envision an implementation of context-aware selective resume within the firmware of PCs, due to the proprietary nature of PC firmware, CAESAR, our context-aware selective resume prototype, was built on the Xen hypervisor. During desktop resume, CAESAR is invoked after BIOS initialization but before OS invocation. Based on context ID received, CAESAR decides whether to invoke a context handler or the full-fledged OS. Our hypervisor-level implementation does not apply the selective resume approach to BIOS initialization. However, we have reported previously [Wright et al. 2011] that BIOS initialization takes under 1s—limiting, as a result, the additional gains from a firmware-level implementation.

CAESAR is capable of inspecting context IDs embedded in Wake-on-LAN packet payloads. Absence of a context ID in the Wake-on-LAN packet or system wake-up from peripherals other than the NIC cause a full OS resume. The Wake-on-LAN packet is an Ethernet broadcast frame (with any network and transport layer protocol) containing six set bytes (0xFFFFFFFFFFFF) and the 48-bit MAC address of the destination network interface repeated 16 times. CAESAR packets are UDP packets with at least 118 bytes of payload. In addition to the Wake-on-LAN packet payload of 102 bytes, CAESAR packets carry 16 bytes of context data. The first eight bytes are the context-aware magic number (0x53524d50), which is used to validate that this is a context-aware selective resume packet. The remaining eight bytes provide the context ID, specifying the relevant application. Beyond these, the application may use the remainder of the packet payload to pass arguments to the context handler, such as the Physical Frame Number for the memory server application.

To intercept suspend and resume tasks, CAESAR modifies Xen's *enter_state* function, responsible for invoking the low-level ACPI call that puts the PC in suspend mode and where execution returns on resume. It is in this function that CAESAR inspects wake-up context and invokes the appropriate context vector. CAESAR's context vector interface supports four application-defined functions: (i) *presuspend*, which provides functionality that is invoked after the OS suspends but before the hypervisor invokes low-level ACPI suspend calls; (ii) *resume*, invoked when the received context matches the application; (iii) *resuspend*, invoked after the context handler's resume function exits; and (iv) *postsuspend*, invoked before full OS resume.

CAESAR implements basic NIC driver functionality in order to access Wake-on-LAN packets without OS support. Our implementation supports the Intel 82574L controller based Gigabit CT Desktop NIC, which was achieved by porting the Linux-based Intel e1000e driver into the hypervisor. When the desktop resumes from low-power mode, the BIOS performs initialization of low-level devices, which include the CPU and chipsets, and powers up the memory controller from self-refresh mode. Subsequently, it transfers execution to the hypervisor, which invokes the e1000e driver code to read the NIC's Wake-Up Packet Memory (WUPM), a set of registers that store the Wake-on-LAN packet. CAESAR then invokes the *resume* function of the context handler, passing to it any remaining data in the Wake-on-LAN packet.

To ensure rapid reinitialization of the NIC, CAESAR locks the NIC's link speed at suspend time, which avoids renegotiation during resume. Normally, when the desktop enters low-power mode, the Intel NIC reduces its link speed from 1Gbps to 10Mbps, which uses marginally less energy. At resume time, the NIC renegotiates the link speed with the switch at the opposite end of the wire in a process known as *Ethernet autonegotiation*. We found this renegotiation to cause a delay in NIC initialization of up to 4–6s. CAESAR prevents link speed renegotiation by setting NIC registers that disable resetting of link speed at suspend time.

Fast Memory Server. We have implemented a memory server that serves VM pages upon receipt of context-aware Wake-on-LAN request packets. The Wake-on-LAN packet payload contains an 8-byte PFN, which CAESAR passes on to the memory server's *resume* function. During the initial suspend of the desktop, CAESAR invokes the memory server's *presuspend* function to map in the VM's PFN-to-MFN translation table. When the memory server's *resume* function is invoked with a PFN, it looks up its memory address in the PFN-to-MFN translation table so that it can read the requested page and send it over the network. Before resuming the OS, CAESAR invokes the memory server's *postsuspend* function, which unmaps the PFN-to-MFN translation table.

7.2. Experimental Evaluation

In this section, we show with an experimental evaluation that context-aware selective resume increases energy savings in desktops using partial VM migration. We specifically answer the following questions:

(1) How effective is context-aware selective resume in reducing resume-suspend cycle times?
(2) Are desktops able to sleep longer with context-aware selective resume?
(3) What is the improvement in energy savings, particularly during short idle times?

Table IV. Power Profile of the Custom PC

State	Time (s)	Power (W)
Suspend	3.0 ± 0	61.07 ± 1.15
Resume	6.33 ± 0.58	65.39 ± 0.82
Idle	N/A	55.50 ± 0.04
Sleep (S3)	N/A	5.06 ± 0.05

Table V. Resume-Suspend Cycle and Page Response Times

	Full Resume	CAESAR
Cycle time (s)	9.0 ± 0.0	3.17 ± 0.41
Page time (s)	6.53 ± 0.0	1.49 ± 0.0

7.2.1. Methodology. To evaluate the benefits of CAESAR, we collected traces of page requests during the deployment of Jettison described in Section 4.2. These traces contain time-stamped sequence of page numbers (PFNs) requested each time VMs were consolidated. Recall that in our initial deployment of Jettison, three university researchers were given desktops running Jettison and employed these as their primary desktops. Each desktop ran a VM configured with 4GiB of memory and 12GiB of disk image. The VMs were configured with Debian Linux 5.0 and ran the GNOME desktop system, along with desktop applications such as Mozilla Firefox, Mozilla Thunderbird, and OpenOffice.org office suite.

For these experiments, we developed an emulator that runs in the consolidation server and replays page requests in the traces while being faithful to the page request interarrivals. In replaying the requests, the emulator wakes up the desktop if it is in low-power mode and retrieves the page before the desktop returns to low power. The emulator enables a direct comparison of the energy savings with and without context-aware selective resume for the same consolidation.

Recall from the discussion in Section 5.2 that we use a wait timer (t_w) that keeps the desktop awake for a predetermined number of seconds after servicing a page request before returning to sleep mode. This is done to anticipate consecutive page requests that have interarrival times that are too low to permit the desktop to microsleep long enough to offset the energy used in the transition to and from low power. In the experiments that follow, the full resume configuration used a wait timer of 6s. Because context-aware selective resume lowers suspend and resume cycle times, and therefore the energy used for transitions, we found that a wait timer of 2s was sufficient to ensure that minimal energy is wasted in transitions for microsleeps that are too short to save energy.

For these experiments, we used a custom desktop PC with a 2.70GHz Dual-Core Intel Celeron CPU and the power profile described in Table IV. The PC was equipped with an Intel Gigabit CT Desktop NIC adapter. Power use of the desktop was measured with a Watts Up? PRO power meter. We ran our experiments on sample traces from all three users over a period of 1 hour.

7.2.2. Resume-Suspend Cycle Times. A basic measure of the energy cost of serving pages on demand is the duration of cycle times, the time the PC takes to resume from low-power mode, serve a page, and return to low-power mode. To measure cycle times, we suspended the desktop and allowed it to idle for 1min. The consolidation server then issued a page request, forcing the PC to resume, serve the page, and immediately return to low-power state. We repeated this experiment five times.

Table V shows the cycle times of the desktop using either full resume or context-aware selective resume. With full resume, the desktop spends, on average, 9.0s out of low-power to serve a single page. CAESAR reduces cycle times by 64.78% to 3.17s. This result demonstrates that CAESAR is effective in reducing the penalty for serving pages on demand.

The page response times, measured in the consolidation server as the time between issuing a page request and receiving the page, dropped accordingly from 6.53s to 1.49s. Table V also shows the response times measured on the consolidation server. Short

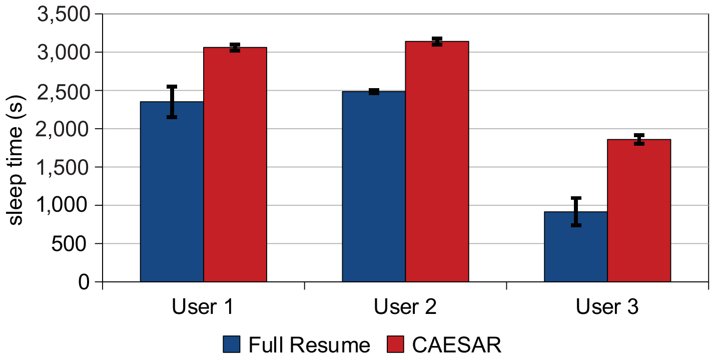


Fig. 17. Desktop sleep times over hour-long consolidations.

response times lower performance degradation of applications running in the consolidation server.

7.2.3. Sleep Times. Shorter cycle times increase the total sleep the desktop can perform during consolidation. Sleep time refers to the time in which the desktop is in low-power mode while its VM is consolidated. Figure 17 shows increases in desktop sleep times with the use of CAESAR over an hour-long consolidation of the VMs for the three users. Sleep times aggregate all intervals of at least 1s in which the desktop used less than 8W of power. The figure shows that, for User 1, CAESAR increases total sleep time from 39.19 to 51.01min. For User 2, it increases total sleep time from 41.41 to 52.32min. And finally, for User 3, CAESAR increases total sleep time from 15.24 to 30.98min. Across all users, CAESAR increases sleep times by 26%–103% during the hour-long period of consolidation.

7.2.4. Energy Savings. We now show that context-aware selective resume increases energy savings of idle desktop systems, which is of particular importance during short periods of consolidation. Figures 18(a), 18(b), and 18(c) compare energy savings when desktops resume fully to serve pages, with savings obtained when CAESAR invokes only the memory server. The figures show the energy savings of the desktop when its VM is consolidated over a period of 1 hour.

Figure 18(a) shows that for User 1, CAESAR increased energy savings from 17% to 28% in the first 5min. At 10min, the savings increased from 28% to 44%. At 20min, CAESAR increased the savings from 44% to 61%, and, finally, at the end of the hour, the savings increased from 61% to 74%. Initially, during the first minute, CAESAR leads to increased energy usage as our implementation first transitions the PC into low-power, which invokes the memory server presuspend function to map the VM's memory and incurs the transition cost before it can start serving pages. This initial penalty can be avoided by serving pages from the administrative domain (as done by Jettison) at the start and transitioning to the CAESAR-based memory server only after the first microsleep.

Figure 18(b), shows that for User 2, CAESAR increased energy savings from 18% to 28% in the first 5min. At 10min, the savings increased from 29% to 42%. At 20min, CAESAR increased the savings from 39% to 57%, and, finally, at the end of the hour, CAESAR increased from 66% to 75%.

Figure 18(c), shows that for User 3, CAESAR increased energy savings from 14% to 16% in 5min. In 10min, it increased savings from 17% to 24%. In 20min, it increased energy savings from 21% to 31%. And, finally, over the course of an hour, savings increased from 26% to 42%. The figure shows that although savings for User 3 are lower across the consolidation, CAESAR is still able to improve savings significantly.

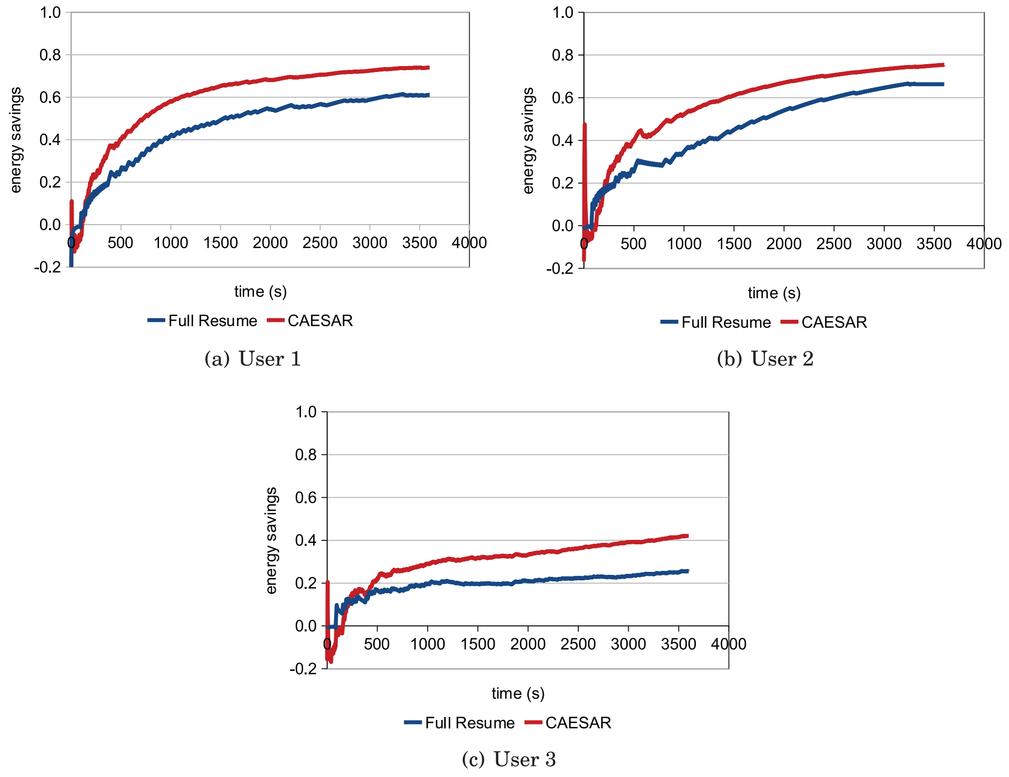


Fig. 18. Desktop energy savings for page access traces from three users of our initial deployment of Jettison.

Across users, CAESAR delivers improvements of 13%–66% over 5min, 43%–54% in 10min, 38%–49% in 20min, and 14%–65% in an hour.

7.3. Summary

In this section, we presented context-aware selective resume, an approach that reduces desktop resume suspend cycles. Current desktops have slow power state transition times with a high rate of power use. We observed that the high cost of on-demand page migration is a limiting factor to achieving considerable savings in short idle times, where faults are frequent. Context-aware selective resume speeds up resume and suspend cycles by initializing only devices and code that is necessary for the wake-up task. The approach allows applications to supply a context ID to the waking PC that informs it of the task causing the wake-up. We presented CAESAR, a framework that implements context-aware selective resume and uses the Wake-on-LAN packet payload to define the wake-up context. We evaluated the performance of a CAESAR-based memory server with traces of VM page requests from VMs of three users. Our results show that CAESAR reduces resume-suspend cycle times by 67% from 9s to 3.17s. It increases total desktop sleep time by 26%–103% over an hour-long consolidation. Most importantly, our experiments show that CAESAR increases energy savings over short idle times between 5min and an hour by up to 66%.

8. SCALABILITY AND COMPARISON WITH FULL VM MIGRATION

In this section, we extrapolate the benefits of partial migration for settings with hundreds of desktops using user-idleness traces collected from real users in an office environment together with the migration and energy-saving performance of Jettison that we measured in the deployment of Section 6.2. We address the following questions:

(1) How does partial migration compare against full migration in terms of network usage, overall energy savings, and the desktop reintegration latency experienced by users?
(2) Do the techniques scale with the number of desktops?
(3) Can they weather "resume storms" present in actual usage patterns?
(4) Most importantly, do the energy savings exceed the capital costs required to deploy each technique?

These experiments are conservative in that they exclude the additional energy savings offered by context-aware selective resume.

Simulation Environment. Our evaluation uses simulation driven by real user traces collected using a Mac OS X-based tracker that runs on a desktop and tracks whether the user is active every 5s. Users are said to be inactive if they are not using the keyboard or mouse and no program (e.g., a video player) has disabled the OS screen saver timer. We deployed the tracker for 4 months at an industrial research lab on 22 researchers' primary work Macs, including both desktops and laptops. The machines had user-controlled software environments—there were no corporate lockdowns in place. We collected 2,086 person day traces from which a sample of 500 are shown in Figure 1. Of the full traces, 1,542 days were weekdays and 544 were weekends. Because a number of traces were from laptops that users take home, usage patterns in the evenings and nights were heavier than would be expected of office desktops. Furthermore, since the lab has flexible work hours, the data do not show tightly synchronized resume storms at the beginning of the workday; the most highly correlated period of inactivity was the lunch hour. Therefore, we expect this dataset to provide fewer sleep opportunities but a somewhat friendlier environment for migration than a traditional office environment.

The traces were fed into a simulator that simulates consolidation and reintegration activity over the course of a single day for a given number of users (traces) and a given value of the *idle timeout*, the time of user inactivity the system waits before consolidating a VM. Because of qualitatively different user behavior on the weekends, we ran simulations using weekday and weekend data separately. We report only on weekday results unless otherwise stated. The simulations assume a shared GigE network, desktop VMs with 4GiB of RAM, and the same energy profile as the desktops used in our experiments (Table III). The simulator takes into account network contention due to concurrent VM migrations when computing consolidation and reintegration latencies. It also takes into account energy use during migrations and desktop sleep periods when computing energy savings. We bias the results in favor of full migration by ignoring iterative precopy rounds or disk accesses and by assuming exactly a 4GiB network transfer per migration for both consolidation and reintegration. Finally, we assume that full migration saves 100% of the desktop's idle power when the VM executes on the server.

For partial migration, we used the distributions of VM memory and disk migration sizes for consolidations and reintegrations shown in Figure 16. Even though partial migration consolidations create network traffic on demand, we assumed bulk transfers on consolidations for ease of simulation. This creates more network congestion and biases results against partial migration. To estimate energy savings for partial migration while accounting for the energy costs of consolidation, reintegration, and servicing of faults, we scale the time the VM remains on the server by a factor obtained from Figure 14 that estimates the savings as a function of consolidation time for our desktops.

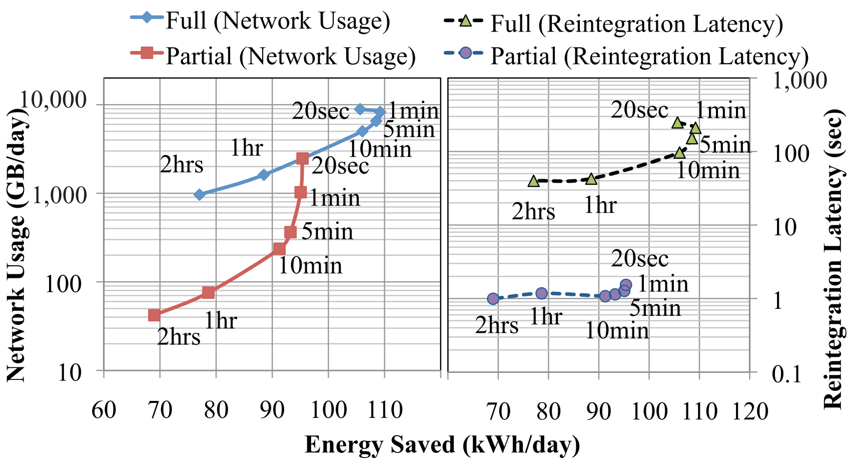


Fig. 19. Energy savings in kW-h per day vs. reintegration latency and network utilization for 100 desktops and varying idle timeout values.

Is Partial VM Migration a Real Improvement? Section 6.2 suggests that when compared to full migration, partial migration significantly improves the network load and user-perceived reintegration latency at the expense of reduced energy savings. The question then arises whether full migration can be made competitive simply by increasing the idle timeout to migrate less aggressively, thus reducing network load and improving reintegration latencies, but reducing sleep opportunities and energy savings. Figure 19 shows that the answer to this question is an emphatic no. It shows a scatterplot of energy savings per day against network load (left graph), and energy savings per day against reintegration latency (right graph) for different values of idle timeout in an office with 100 desktops. While partial migration does not match the highest energy savings possible using full migration in this setting (although it gets to within 85%), for an equal amount of energy saved, it has over an order of magnitude lower network load and reintegration latency.

The graphs also show that for both full and partial migration, there is a sweet-spot between 5 and 10min for the idle timeout. Higher values significantly reduce energy savings, whereas lower values dramatically increase network load and reintegration latency without increasing energy savings much. For full migration, energy savings actually decline for small idle timeouts because the aggressive migrations led to a lot of energy wasted in aborted migrations and oscillations between the desktop and consolidation server. Similar graphs for 10–500 desktops show that an idle timeout between 5 and 10min provided the best balance of energy savings and resource usage across the board.

Scaling with Number of Desktops. Next, we show how the benefits of partial migration scale with the number of users. We use an idle timeout of 5min for these experiments.

Figure 20 shows a greater than two orders of magnitude reintegration latency advantage for partial migration at 100 users that grows to three orders of magnitude at 500 users. Increased congestion and resume storms cause the performance of full migration to degrade with scale. In contrast, the latency of partial migration remains very stable. We contend that even at 100 users, the 151s reintegration latency of full migration could be intolerable for users. Das et al. [2010] propose using a remote desktop solution to provide immediate reintegration access to users to mask long reintegration latencies of full migration. However, remote desktop access has many limitations, such as the

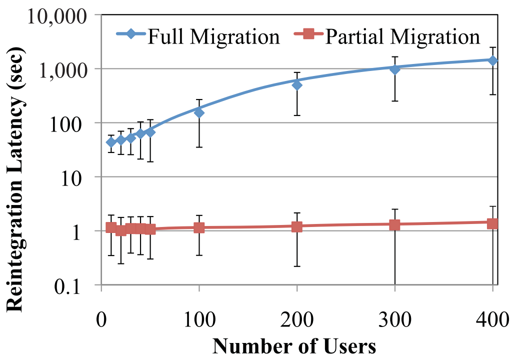


Fig. 20. Reintegration latency of partial migration and full migration as desktops increase. The error bars show the standard deviations.

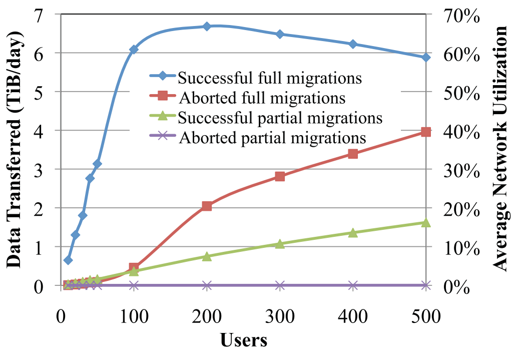


Fig. 21. Network load of partial migration and full migration. Aborted migrations occur when a new migrations overrides an ongoing one.

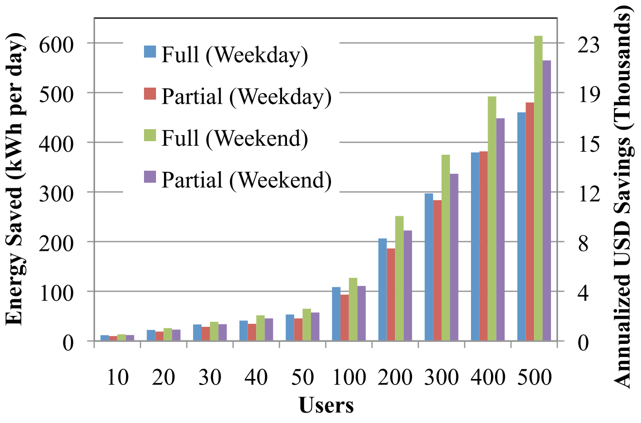


Fig. 22. Total energy savings in kWh and USD by partial migration and full migration.

inability to seamlessly access local devices such as graphics cards and the reliance on the performance of an overburdened network that is the cause of the long reintegration latencies in the first place. We show that partial migration offers a superior alternative.

Figure 21 shows that network utilization of partial migration is an order of magnitude lower than full migration and remains low even as the number of users grows. Due to the fast consolidation and reintegration times, there are few aborted migrations. Aborted full migrations result from long migration times that increase with network congestion, and these reduce successful attempts and, ultimately, energy savings. The y2 axis of the figure shows the average daily network utilization in terms of total network capacity. Full migration quickly dominates the network (65% utilization at 100 users) and, as a result, often requires dedicated network infrastructure to prevent interfering with other applications.

Cost Effectiveness. Figure 22 shows the overall energy savings in kWh per day for partial and full migration for both the weekday and weekend datasets. The y2 axis shows the corresponding annualized energy savings using the average July 2011 US price of electricity of USD 0.1058 per kWh.[1] As the number of desktops increase,

[1]US Energy Information Administration: http://www.eia.gov/electricity/.

partial migration becomes more efficient than full migration (85% of full migration at 10 users to 104% at 500 users) because the large consolidation times for full migration on an increasingly congested network significantly reduce sleep time opportunities. Weekends are better, but weekdays have significant savings as well: With an idle timeout of 5min, VMs spend an average of 76% of a weekday on the server. With at least 100 desktops, energy savings increase almost linearly with the number of desktops, at a rate of USD 37.35 and 33.95 per desktop per year for partial and full migration, respectively.

We can compare these savings to the yearly depreciation costs for the consolidation servers to determine whether the schemes can pay for themselves. The question we ask is: Assuming a 3-year depreciation window, can each migration scheme justify the purchase of a server with energy savings alone? We assume a server with 16GiB of memory, similar to our testbed system. Since fully migrated idle VMs are memory constrained on the server side, we assume four 4GiB VMs on a single server, giving us a breakeven server budget of USD 33.95×4 VMs $\times 3$ years, or USD 407.40. In comparison, the results from Section 6.2 show that we can fit 98 partial VMs on a 16GiB server when using partial migration, giving partial migration a budget of 37.35×98 VMs $\times 3$ years, or USD 10,980.90. To put these numbers in context, we priced the SunFire X2250 server used in our testbed at USD 6,099.[2] In conclusion, given existing server and electricity prices, a large consolidation ratio is required to make consolidation of idle desktop VMs cost effective, and partial VM migration is able to provide this high consolidation.

8.1. Considerations for Deployment of Partial VM Migration

There are three additional considerations that implementers must make when deploying partial migration of idle desktop VMs in enterprise environments: (i) where to place VM storage, (ii) how to handle device accesses, and (iii) what happens in case of failures.

Storage Placement. As described in Section 1, partial VM migration is data-placement agnostic. Disk state can be placed either in network servers or locally on the desktops. The benefit of partial VM migration is in reducing migration of run state, and, as we have shown in Section 5, more than 99% of state accessed by idle VMs is memory (165.63MiB), whereas disk represents less than 1% (1.16MiB). An environment with shared network storage reduces the number of faults that must be serviced from the desktop and potentially increases the energy savings with partial VM migration, albeit minimally. In our deployment, we used desktop local storage, which supports legacy environments where shared storage is not always available.

Device Support. Thick-client desktops can provide access to dedicated hardware devices to applications and users. The use of devices such as webcams, microphones, sound cards, GPUs, fingerprint readers, and even network interfaces can complicate the migration of VMs for two reasons. First, some devices found on the desktops may not be available on the consolidation server. Second, for high-performance applications, or in order to protect hypervisors from unstable or untrusted device drivers, VMs may be given direct access to devices via passthrough [LeVasseur et al. 2004; Xia et al. 2008; Advanced Micro Devices, Inc. 2009; Intel Corporation 2011a]. In these instances, the hypervisor is unable to capture and migrate the state of devices, and the in-VM device driver is exposed to any mismatch between device models on the server and the desktop. Solutions have been proposed to support migration of passthrough devices

[2]https://shop.oracle.com.

[Kadav and Swift 2008; Zhai et al. 2008], and some newer devices provide support for Single-Root I/O Virtualization and Sharing (SR-IOV) [Intel Corporation 2011b; Dong et al. 2008], which virtualizes the device itself, allowing context switching between VMs, and, more importantly, allows the hypervisor to capture and restore device state.

Even if devices can be migrated, it is well worth considering when it is appropriate for a consolidated VM to: (i) access the device on the server, (ii) forward device accesses to the desktop, or (iii) migrate back to the desktop whenever it requires access to a device. For example, although it is sensible for the consolidated VM to access the network interface of the server, accesses to the sound card may be more useful when forwarded to the desktop. This could enable a VoIP or IM client to notify a nearby user of an incoming call or message. Similarly, access to a 3D acceleration interface may require the VM to be migrated back to the desktop. An emulated device backend is needed on the consolidation server to capture access events and forward them to the physical device on the desktop. This approach may also be used to address device heterogeneity by forcing the VM to only use the device on the desktop, when an incompatibility is detected. Finally, a given device type may support different access modalities, and a policy needs to be in place to determine how best to access the device. For example, disk accesses may be forwarded to the desktop when infrequent. However, frequent accesses caused, for example, by a running virus scanner, could force the VM to migrate to the desktop because they do not allow the desktop to sleep. Our prototype implementation supports paravirtualized devices that make migration simple and provide a consistent interface across hosts. We have also implemented an emulated back end on the consolidation server that forwards block accesses on the disk device to the desktop, and we found this configuration to work well in our deployment. More work is needed to support VM migration in environments with heterogeneous devices where VMs are configured with device passthrough.

Failure Semantics. The semantics of failure of a consolidated VM on storage consistency is similar under local or network storage configurations and is based on checkpointing. When a VM is consolidated, memory and disk state are checkpointed on the desktop. Disk changes made by the partial VM are stored in the per-disk dirty slice held as a file in the server. If a failure occurs on the server, Jettison resumes the VM from checkpointed state on the desktop. The benefit of this approach is that, in case of server failure, the desktop resumes from consistent disk and memory state. The disadvantage is that state generated on the server and that may otherwise be useful may be lost. This loss can be limited by periodically propagating resume-consistent disk and memory checkpoints to the desktop or replicating this state across multiple servers. In cases of server-side failures that do not corrupt the host's file system and from which the host can recover (e.g., by rebooting), dirty disk state may still be recovered. We have implemented server-side disk writes buffering, which enables desktop recovery from checkpoint only for the local disk driver. Adding a similar functionality to shared storage drivers is left for future work.

In cases of network failures, the hypervisor suspends each faulting vCPU until it is able to fetch the missing pages from the desktop. Memtap is configured to retry to fetch pages for a period of up to 3 hours. Guests experience vCPU suspend events as they do with VM checkpointing and resumes, which are widely used in virtualized environments. Faulting vCPUs are not rescheduled until outstanding page faults can be serviced, and, when network failures persist, the partial VM remains in suspend state. It is possible to reintegrate the partial VM state back to the desktop if an out-of-band channel is available to transfer the state, or the partial VM can be discarded and the desktop VM resumed from its own checkpoint state.

8.2. Summary

With traces of PC user activity, we showed that in small to medium-sized offices, partial migration provides energy savings that are competitive with full VM migration (85% of full migration at 10 users to 104% at 500 users) while providing migration latencies that are two to three orders of magnitude smaller and network utilization that is an order of magnitude lower. We also identified considerations that implementers must make when deploying partial migration of idle desktop VMs in enterprise environments.

9. RELATED WORK

This article introduces partial desktop VM migration, an approach that consolidates idle desktop VMs and places desktop PCs in low-power state to reduce their energy use. This approach migrates VM memory pages to the consolidation server on demand while maintaining state residues on the desktop. Migrations are fast and do not congest enterprise networks, and consolidation ratios on servers are high. When migrating the VM back to the desktop PC, the approach transfers only deltas consisting of state that has been updated while the VM ran on the server. We introduced context-aware selective resume, a framework that reduces desktop resume and suspend times by reinitializing only devices and code necessary for a given wake-up context.

In this section, we discuss prior art in two areas to which our contributions apply: energy conservation in desktop systems and migration of VMs and processes. We build on the discussions of Section 2 and Section 3, discuss how our approach relates to previous work, and identify the challenges with existing solutions.

9.1. Energy Conservation

Early approaches to energy conservation have been motivated by the limited battery life of mobile devices [Noble et al. 1997; Flinn and Satyanarayanan 1999; Zeng et al. 2002]. Recently, energy usage and heat generation have become a concern in data centers [Barroso and Hölzle 2007; Meisner et al. 2009; Fan et al. 2007; Tolia et al. 2008]. In more recent years, energy use of desktop computers has also garnered the interest of researchers, motivated by rising energy costs and the environmental impact of electricity generation [Webber et al. 2006; Gunaratne et al. 2005; Agarwal et al. 2009; Nedevschi et al. 2009; Agarwal et al. 2010; Das et al. 2010; Sen et al. 2012].

9.1.1. Thin Clients. Thin clients, such as VNC [Richardson et al. 1998] and SLIM [Schmidt et al. 1999], place low-power stateless clients on the user's desk and run their applications remotely on shared servers. The thin client is only responsible for displaying the output of the user session running on the remote server and forwarding user input to the server. Because thin clients are low-power devices, the energy wasted when idle is relatively small. However, thin clients remain unpopular due to reduced interactive performance, lacking crispness in response to user interaction, and lack of access to acceleration in local hardware. In addition, whereas thin clients reduce client energy use, they do little to improve server energy efficiency because these run user sessions with full state. Servers must also be provisioned to accommodate the peak workloads of each user, thus limiting the number of concurrent sessions a single server can run. In contrast, our approach delivers the performance of local hardware whenever the user is active and, when the user is inactive, runs the session remotely with minimal state and transitions the desktop into low-power state. Only idle workloads run on the shared server, and, as a result, hardware resources can be shared among many more users.

9.1.2. Energy Proportionality in Data Centers. Studies of data centers have found that idle servers draw about 60% of their peak power and that the average server utilization is only 20%–30% [Barroso and Hölzle 2007; Meisner et al. 2009; Fan et al. 2007]. A recent study commissioned by the *New York Times* found that between 88% and 94% of the power used by IT data centers is wasted in idle servers [Glanz 2012]. Although frequent, idle periods in data centers can be short, often lasting a few seconds or less. These findings have prompted calls for a fundamental redesign of system components so they consume energy in proportion to their utilization [Barroso and Hölzle 2007]. Rather than requiring fine-grain power performance tuning from all components, PowerNap [Meisner et al. 2009] calls for systems that transition between high-performance active state and low-performance nap state rapidly in response to instantaneous load changes and for systems that support energy-efficient sleep states. Experiments conducted in the PoweNap project show that, for this approach to work, state transitions must take no more than 10ms; obviously, a challenge with existing hardware. In contrast, our approach works well with existing hardware for which transition times last up to tens of seconds. Dynamic voltage and frequency scaling combined with opportune consolidation of idle VMs (which enables server shut off) have been used to approximate energy proportionality at the data center scale [Tolia et al. 2008]. This approach relies on full migration of VMs, which we have shown to be slow, cause large network traffic, and not lead to high consolidation ratios.

We expect improvements in hardware performance, either with energy-proportional components or with fast power state transitions, to benefit our approach considerably. Energy-proportional components can mean that serving pages on demand uses only a fraction of power needed by the components used to transmit the pages (CPU, memory, and NIC). Similarly, fast transition times ensure that PCs spend much of their time in low-power and not in transition, again, lowering the cost of serving pages on demand.

9.1.3. Opportunistic Sleep in Desktop PCs. Early approaches based on opportunistic sleep sought to provide support for LAN-based remote access applications such as file access or system management. Wake-on-LAN [Lieberman Software Corporation 2006] was developed by the Intel and IBM Manageability Alliance to allow system administrators to wake sleeping computers and perform maintenance tasks. At a high level, when a computer goes to sleep, it maintains its network interface powered on and constantly scanning for packets on the network. When the NIC receives a specially crafted "magic packet" containing its own MAC address, it generates a power management event that wakes the host. Wake-on-Wireless-LAN [Intel Corporation 2006] extends this functionality to 802.11 wireless networks. Wake-on-Wireless [Shih et al. 2002] is targeted at mobile computers such as laptops. It supplements the mobile device with a low-power radio that is used to wake the host in lieu of keeping the 802.11 NIC powered during sleep time. An external proxy is used to signal the device over the low-power radio interface.

CellNotify [Agarwal et al. 2007] substitutes the low-power radio interface for a cellular radio, enabling it to work over a wide area. Although acceptable to mobile communication devices with a cellular interface, widespread adoption of the approach is infeasible in desktop environments.

Although Wake-on-LAN and, to some degree, Wake-on-Wireless LAN are widely available on modern PCs, these techniques are seldom used. Approaches that wake the PC on demand do not work well with applications that must run while the PC is in low-power state. Fundamentally, all opportunistic sleep-based techniques described thus far do not support applications with always-on semantics that need to maintain Wide Area Network (WAN) presence from within the desktop.

Exploiting short opportunities for sleep while a host is waiting for work is also explored in Catnap [Dogar et al. 2010]. Catnap exploits the bandwidth difference between WLAN interface of end hosts and the WAN link to allow idle end hosts to sleep during network downloads while content is buffered in network proxies. That approach is focused on energy reduction in ongoing network transfers and not in providing continued execution of desktop applications during sleep.

9.1.4. Protocol Proxies. Low-power protocol proxies have been used to maintain network presence of always-on applications whenever the PC goes to sleep, only waking the PC when its resources or attention are required [Gunaratne et al. 2005; Jimeno et al. 2008; Nedevschi et al. 2009; Reich et al. 2010]. The GreenUp decentralized wake-up service [Sen et al. 2012] develops a distributed architecture that enables participating PCs to act as proxies for sleeping hosts in the same subnet. Somniloquy [Agarwal et al. 2009] delegates basic application functionality to application stubs that run on a smart NIC of the desktop whenever the desktop is idle. The NIC is capable of operating when the main processor is asleep and can wake the desktop when necessary. Turducken [Sorber et al. 2005] is a more general instance of the proxy-based approach that relies on a hierarchical power management architecture in which each tier is incrementally more powerful and more energy taxing. For example, the top tier may consist of a full-size desktop capable of running rich graphical applications. The next tier may consist of an embedded processor with flash storage and networking capabilities that runs application stubs. A third tier may consist of a sensor whose job is to detect network availability and wake tier two accordingly.

Proxy-based approaches require that always-on applications be reengineered to support bimodal operation between the host and proxy; thus, even for solutions whose proxies only handle simple protocols and wake the desktop on more sophisticated requests [Nedevschi et al. 2009; Reich et al. 2010], they are unable to support the plethora of cloud applications with client-driven architectures. For example, Reich et al. found that for cloud storage applications Microsoft LiveMesh and Microsoft LiveSync, the client needed to periodically retrieve file updates from the cloud, which was not supported with the simple proxy employed. Similarly, asynchronous JavaScript and XML (AJAX) applications such as Facebook Chat [Facebook, Inc. 2012], Google Docs [Google, Inc. 2012b], and Gmail Chat [Google, Inc. 2012a], are just few examples of applications that are growing in popularity and for which proxy-triggered wake-on-demand does not work.

SleepServer [Agarwal et al. 2010] bears some similarity to our approach in that application presence of the sleeping desktop is maintained in VMs running remotely. However, this approach differs from ours because the VM runs instances of purpose-engineered application stubs separate from the main instance of the application running in the desktop. Our approach migrates the VM instance running on the desktop into the cloud. The SleepServer approach presents the same drawbacks as the proxy-based approaches described previously. Either applications are re-engineered, or client-driven cloud applications with network presence do not work.

9.1.5. VM Consolidation. Consolidation of VMs has been used in the data center to increase server efficiency [Tucker and Comay 2004; Badaloo 2006; Marty and Hill 2007]. Early approaches made permanent assignments of VMs to hosts and only rarely revised those assignments. More recently, however, with the development of live migration of VMs [Clark et al. 2005], consolidation has been performed dynamically, allowing server footprint to expand and shrink as a function of workload. Dynamic VM consolidation has been used to consolidate workloads and power off underutilized servers to deliver data center energy proportionality [Tolia et al. 2008].

VDI, discussed in Section 3, is an approach that permanently runs user VMs from shared server infrastructure. Users connect to their VMs from either thin or thick clients. VDI has been designed to simplify enterprise desktop management and deliver enterprise-class storage characteristics such as high reliability. By running hundreds of desktop VMs from a single on-disk system image, the “golden image,” VDI enables IT administrators to propagate system updates by patching only the golden image. Because servers are provisioned to run the peak desktop workloads of all users, and VDI imposes high storage performance requirements [Spruijt 2010], it delivers low consolidation ratios, has high infrastructure cost, and its servers use power inefficiently. VDI also limits access to dedicated local hardware on the user’s client, such as 3D acceleration and dedicated media encoding hardware. These challenges have led to a slow adoption of VDI [Fograrty 2011].

A recent proposal by Intel [Intel Corporation 2011c] places VMs on PCs while maintaining VM provenance from a golden image stored on a server. The IDV provides the ease of management benefits of VDI but makes no provision for idle energy conservation. Partial migration of idle desktop VMs can deliver the energy savings for idle desktops in these environments.

LiteGreen [Das et al. 2010], which we discuss in detail in Section 3, uses consolidation of idle desktop VMs to reduce energy use. Active VMs run on desktop PCs, and idle VMs run on consolidation servers. This approach uses full migration of VMs, which we show to be slow, congest enterprise networks, and to not inherently lead to high consolidation ratios. We showed in Section 8 that migration patterns in offices with 300 users lead to migration latencies that exceed 17min when full VM migration is utilized. Users must wait that long before they can access their VMs. Partial VM migration delivers the energy savings of this approach while maintaining low migration latencies of less than 5s, migrating less than 10% of the VM’s memory state, and delivering high consolidations ratios.

9.2. Migration of Virtual Machines and Processes

Migration is an inherent benefit of machine virtualization. Virtualization enables the serialization of full machine execution state, including CPU registers, memory, and device state, via checkpointing. Once serialized to persistent storage, this state can be used to resume execution of the VM at a later time. Execution may be resumed on the same or a different physical host. Previous work has taken advantage of this ability to enable VM mobility [Sapuntzakis et al. 2002; Kozuch and Satyanarayanan 2002; Whitaker et al. 2004].

In an effort to reduce migration sizes, record and replay of UI interactions was proposed to synchronize two VMs over a wide area [Surie et al. 2008]. A log of user-generated UI events is recorded on the source VM and replayed on an outdated copy of the VM at destination. Although this approach can reduce synchronization time, especially in low-bandwidth environments, as the authors found, replay of interactions is insufficient to eliminate divergent VM state. There are many events that lead to diverging state, including timer events, interrupts, and other external stimuli such as network activity. Also, its performance is bounded by the speed of replay, which in turn depends on the speed with which applications can perform the task caused by each interaction.

9.2.1. Live Migration of VMs. Precopy live VM migration [Clark et al. 2005] improved the state of the art by migrating most state while the OS continues to run at the source to ensure minimal down times (as low as 60ms). Precopy live VM migration is performed iteratively in three phases. In the first, it copies all memory to the destination and tracks any pages dirtied by the running VM at the source in the interim. In the second phase, it copies the dirty pages to the destination, tracking additional pages that get dirtied. This step is repeated until only a small set of dirty pages remains. The third phase consists of suspending the VM’s execution at the source, copying all remaining dirty pages and CPU context to the destination, and beginning execution there. To reduce migration sizes during the second phase of live migration (which can be large if the running workload is write heavy), per-page deltas can be computed and compressed [Svärd et al. 2011]. CloudNet [Wood et al. 2011] reduces the number of copy iterations by detecting when consecutive iterations no longer reduce the number of dirty pages copied and switching to the final phase of migration as soon as a local dirty page minimum is reached. It also uses content-based redundancy elimination and subpage delta transmission to reduce migration sizes.

Remus [Cully et al. 2008] and SecondSite [Rajagopalan et al. 2012] take advantage of precopy live VM migration to perform rapid replication of VMs and deliver fault tolerance and high availability for critical services. They aggressively checkpoint and replicate the state of an executing VM by migrating only the updated state to the destination. VMsync [Bickford and Cáceres 2013] considers replication of a live VM between mobile devices by distributing a base VM image across the devices and incrementally propagating deltas of the running VM with subpage granularity.

Postcopy live VM migration [Hines and Gopalan 2009] inverts the migration order of precopy live VM migration. First, it halts execution of the VM at the source and transfers the CPU state to the destination, where it initiates VM execution immediately. Then, in the background, it migrates all memory pages. When the running VM accesses a missing page, postcopy live VM migration, pages in the missing page from the source. To do so, postcopy live VM migration implements a network device to which pages are swapped out at the source and swapped in at the destination.

The live migration approaches discussed are designed to migrate the VM to the destination in full. Our work has demonstrated that migrating VMs in full is unnecessary, and indeed does not scale well for energy-oriented idle desktop consolidation in enterprise networks.

9.2.2. Fine Grain Migration of VMs. SnowFlock [Lagar-Cavilla et al. 2009] has demonstrated the benefits fine-grain migration of VM state in the data center. SnowFlock implements the VM fork abstraction, which enables server VMs to create incomplete stateful replicas of themselves to take advantage of additional processors in dealing with increased workloads. Each replica, referred to as a *clone* in SnowFlock parlance, is created with minimal amount of state, consisting of vCPU state, page tables, and device configuration metadata. The clone migrates pages on demand as it accesses them. Because the source of the migration, known as the *master VM*, continues to run, it uses copy-on-write on its memory and disk to preserve its migration time state for use by its clones. Upon completion of their tasks, clones must use an out-of-band mechanism to communicate their results because SnowFlock does not persist their state in the master VM. The challenge with SnowFlock has been that on-demand state propagation causes an extended warm-up period over which network page faults are frequent and the performance of the clone is significantly degraded.

Kaleidoscope [Bryant et al. 2011] uses SnowFlock to enable *cloud microelasticity*. Clouds grow and shrink at will in support of the changing workloads. An enabler of cloud microelasticity is *state coloring*. State coloring classifies memory into sets of semantically related regions and optimizes propagation and deduplication of the state among the clones. Coloring enables prefetching of related pages that may be placed in disjoint physical memory locations.

Partial VM migration uses the same mechanisms as SnowFlock to migrate pages on demand from the desktop to the consolidation server. However, partial VM migration provides the mechanisms to reintegrate replica state back to the desktop by preserving VM residues on the desktop and tracking and migrating dirty state back from the server. Our work shows that on-demand migration is suitable for reducing energy use of idle desktops.

9.2.3. Process Migration. Work on process migration predates live VM migration. The goal is similarly to migrate running processes from one machine to another without significant disruption to the processes' execution. Process migration differs from live VM migration in migration granularity: Instead of migrating VMs (OS and all), it migrates individual processes. Many systems have been developed that provide process migration capabilities, including DEMOS [Powell and Miller 1983], V [Theimer et al. 1985], Mach [Accetta et al. 1986], Condor [Michael J. Litzkow and Mutka 1988], Sprite [Douglis and Ousterhout 1991], Accent [Zayas 1987], MOSIX [Barak et al. 1993], and the Load Sharing Facility [Zhou et al. 1993]. Several studies have demonstrated the need to migrate processes for delivering good interactive performance, especially in compute clusters with mixed parallel and interactive workloads [Arpaci et al. 1995; Anderson et al. 1995]. Despite the large body of work, process migration has not achieved widespread use [Milojicic et al. 2000]. Reasons for failure to gain widespread acceptance include function-level residual dependencies on the source of the migration and lack of compatibility between kernel versions, libraries, and devices across hosts. Migrated processes have residual dependencies to in-kernel context that may not be migrated, such as open sockets and file descriptors. Systems such as Condor and Sprite try to address residual dependencies by forwarding some system calls of processes running on foreign hosts back to the hosts where they started, which at best leads to degraded I/O performance.

Partial VM migration combines the benefits of process migration (namely, small migration footprints) with the reliability of VM migration. It deals with residual dependencies at the page granularity, which limits access to the source to, at most, once per page and does not introduce remote function call dependencies. Partial VM migration migrates all dependencies accessed by migrated processes (including OS objects) transparently, thus avoiding the problems of heterogeneity in OS, libraries, and (virtualized) devices across hosts. For large processes, much like the lazy page copying mechanisms in Accent and Sprite, partial VM migration reduces migration payload to only those pages accessed during idle execution.

10. CONCLUSION AND FUTURE WORK

This article introduced partial migration of VMs. An important use of this capability is for energy savings through consolidation of idle desktops in the private cloud of an enterprise to support applications with always-on network semantics. When the user is inactive, partial VM migration transfers only the working set of the idle VM for execution on the consolidation server and puts the desktop to sleep. When the user becomes active, it migrates only the changed state back to the desktop. It is based on the observation that idle Windows and Linux desktops, in spite of background activity, access only a small fraction of their memory and disk state: typically less than 10% for memory and about 1MiB for disk. It migrates state on demand and allows the desktop to microsleep when not serving requests. Partial VM migration reduces idle time energy use with the dual benefits that network and server infrastructure can scale well with the number of users and migration latencies are small. Our results show that migration latencies of partial VM migration are low, averaging 4s.

Results with our prototype show that partial VM migration is effective in reducing energy use even in short idle periods. In contrast, commercial systems limit energy savings in short periods of inactivity (10–20min) to display management only because of their inability to maintain application network presence when the PC is in low-power mode (e.g., default behaviors of Windows 7, Mac OS X 10.6, and guidelines of the EPA [U.S. Environmental Protection Agency 2013] and the Department of Energy [U.S Department of Energy 2013]). Partial VM migration not only turns the monitor off, but also transitions the PC into low-power mode because it is able to keep applications running and can transition back to local execution quickly.

Our initial deployment results with Jettison show that, excluding monitor savings, our desktops saved 78% of the energy used in an hour, and 91% in 5 hours. These results also showed that, in small to medium-sized offices, partial VM migration provides energy savings that are competitive with full VM migration (85% of full migration at 10 users to 104% at 500 users), with migration latencies that are two to three orders of magnitude smaller and network utilization that is an order of magnitude lower.

We identified the cost of servicing page faults on demand as a challenge to achieving substantial energy savings in short idle periods, as well as a potential challenge to the reliability of hardware and timing-sensitive software systems. During our initial deployment, energy savings in idle periods up to 10min remained less than 16%. To address these challenges, we developed context-aware selective resume, a software-only solution for legacy desktop PCs that improves power state transition times by providing a wake-up context and initializing only devices and code needed for the task of the context. Our experiments with a memory server based on CAESAR, the context-aware selective resume framework, show that this approach increases energy savings in idle intervals of under an hour by up to 66%. In idle intervals of 10min, it delivers energy savings of 24%–44%, and, in intervals of 20min, it produces savings of 31%–61%. Context-aware selective resume makes saving energy during short idle intervals attractive.

Additional improvements are needed to make partial VM migration practical in production environments. Improvements to hardware power state transition and reintegration times are needed to reduce worst-case resume times; additional studies are needed to better understand the tradeoffs between the goals of energy conservation and usability, as well as any long-term effects of the use of our approach on user behavior; finally, additional studies of page-level OS behavior may improve page prefetch performance, thus resulting in improved energy savings.

REFERENCES

Mike Accetta, Robert Baron, William Bolosky, David Golub, Richard Rashid, Avadis Tevanian, and Michael Young. 1986. Mach: A new kernel foundation for UNIX development. In *Summer USENIX Conference*.

Acer, Inc. 2012. Acer Aspire S3. Retrieved from http://us.acer.com/ac/en/US/content/series/aspiresseries.

Acid3 Test. 2012. The Acid3 Test. Retrieved from http://acid3.acidtests.org/.

Advanced Micro Devices, Inc. 2009. AMD I/O Virtualization Technology (IOMMU) Specification. Retrieved from http://support.amd.com/us/Embedded_TechDocs/34434-IOMMU-Rev_1.26_2-11-09.pdf.

Yuvraj Agarwal, Ranveer Chandra, Alec Wolman, Paramvir Bahl, Kevin Chin, and Rajesh Gupta. 2007. Wireless wakeups revisited: Energy management for VOIP over Wi-Fi smartphones. In *5th International Conference on Mobile Systems, Applications and Services (MobiSys'07)*.

Yuvraj Agarwal, Steve Hodges, James Scott, Ranveer Chandra, Paramvir Bahl, and Rajesh Gupta. 2009. Somniloquy: Augmenting network interfaces to reduce PC energy usage. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI'09)*.

Yuvraj Agarwal, Stefan Savage, and Rajesh Gupta. 2010. SleepServer: A software-only approach for reducing the energy consumption of PCs within enterprise environments. In *USENIX Annual Technical Conference (USENIX ATC'10)*.

Thomas E. Anderson, David E. Culler, and David A. Patterson. 1995. A case for NOW (networks of workstations). *IEEE Micro* 15, 1 (Feb. 1995), 54–64.

Apple, Inc. 2012. MacBook Air. Retrieved from http://www.apple.com/macbookair/performance.html.

Remzi H. Arpaci, Andrea C. Dusseau, Amin M. Vahdat, Lok T. Liu, Thomas E. Anderson, and David A. Patterson. 1995. The interaction of parallel and sequential workloads on a network of workstations. In *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'95)*.

Moonish Badaloo. 2006. An Examination of Server Consolidation: Trends that Can Drive Efficiencies and Help Businesses Gain a Competitive Edge. Retrieved from http://www.ibm.com/systems/optimizeit/pdf/server_consolidation_whitepaper.pdf.

Ammon Barak, Shai Guday, and Richard G. Wheeler. 1993. *The MOSIX Distributed Operating System—Load Balancing for UNIX*. Vol. 672. Springer-Verlag.

Ricardo A. Baratto, Shaya Potter, Gong Su, and Jason Nieh. 2004. MobiDesk: Mobile virtual desktop computing. In *International Conference on Mobile Computing and Networking (MobiCom'04)*. Philadelphia, PA.

Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. 2003. Xen and the art of virtualization. In *19th Symposium on Operating Systems Principles (SOSP'03)*.

Luiz André Barroso and Urs Hölzle. 2007. The case for energy-proportional computing. *IEEE Computer* 40, 12 (Dec 2007).

Jeffrey Bickford and Ramón Cáceres. 2013. Towards synchronization of live virtual machines among mobile devices. In *14th Workshop on Mobile Computing Systems and Applications (HotMobile'13)*.

Roy Bryant, Alexey Tumanov, Olga Irzak, Adin Scannell, Kaustubh Joshi, Matti Hiltunen, Horacio Andrés Lagar-Cavilla, and Eyal de Lara. 2011. Kaleidoscope: Cloud micro-elasticity via VM state coloring. In *ACM European Conference on Computer Systems (EuroSys'11)*.

California Energy Commission. 2006. California Commercial End-Use, CEC-400-2006-005. Retrieved from http://www.energy.ca.gov/2006publications/CEC-400-2006-005/CEC-400-2006-005.PDF.

Peter M. Chen and Brian D. Noble. 2001. When virtual is better than real. In *8th IEEE Workshop on Hot Topics in Operating Systems (HotOS'01)*.

Citrix Systems, Inc. 2011. Citrix VDI-in-a-Box. Retrieved from http://www.citrix.com/site/resources/dynamic/salesdocs/VDIinaBox_Datasheet.pdf.

Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. 2005. Live migration of virtual machines. In *2nd Conference on Symposium on Networked Systems Design and Implementation (NSDI'05)*.

Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and Andrew Warfield. 2008. Remus: High availability via asynchronous virtual machine replication. In *5th USENIX Symposium on Networked Systems Design and Implementation (NSDI'08)*.

Tathagata Das, Pradeep Padala, Venkat Padmanabhan, Ramachandran Ramjee, and Kang G. Shin. 2010. LiteGreen: Saving energy in networked desktops using virtualization. In *USENIX Annual Technical Conference (USENIX ATC'10)*.

Tim Dierks and Eric Rescorla. 2008. The TLS Protocol: Version 1.2. Retrieved from https://tools.ietf.org/html/rfc5246.

Fahad R. Dogar, Peter Steenkiste, and Konstantina Papagiannaki. 2010. Catnap: Exploiting high bandwidth wireless interfaces to save energy for mobile devices. In *International Conference on Mobile Systems, Applications and Services (MobiSys'10)*.

Yaozu Dong, Zhao Yu, and Greg Rose. 2008. SR-IOV networking in xen: Architecture, design and implementation. In *First Workshop on I/O Virtualization (WIOV'08)*.

Fred Douglis and John Ousterhout. 1991. Transparent process migration: Design alternatives and the sprite implementation. *Software Practical Experience* 21, 8 (Aug. 1991), 757–785.

Facebook, Inc. 2012. Facebook Chat. Retrieved from http://www.facebook.com/sitetour/chat.php.

Xiaobo Fan, Wolf-Dietrich Weber, and Luiz André Barroso. 2007. Power provisioning for a warehouse-sized computer. In *ACM International Symposium on Computer Architecture (ISCA'07)*.

Jason Flinn and M. Satyanarayanan. 1999. Energy-Aware adaptation for mobile applications. In *17th ACM Symposium on Operating System Principles (SOSP'99)*.

Kevin Fograrty. 2011. The Year of the Virtual Desktop Fails to Materialize–Again. Retrieved from http://www.cio.com/article/691303/The_Year_of_the_Virtual_Desktop_Fails _to_Materialize_Again.

James Glanz. 2012. Power Pollution and the Internet: Data Centers Waste Vast Amounts of Energy, Belying Industry Image. Retrieved from http://nyti.ms/1sW5bMM.

Robert P. Goldberg. 1973. *Architectural Principles for Virtual Computer Systems*. Technical Report. Harvard University.

Google, Inc. 2012a. Gmail. Retrieved from http://gmail.com.

Google, Inc. 2012b. Google Docs. Retrieved from http://docs.google.com.

K. Govil, E. Chan, and H. Wasserman. 1995. Comparing algorithms for dynamic speed-setting of a low power CPU. In *International Conference on Mobile Computing and Networking (MobiCom'95)*.

Chamara Gunaratne, Ken Christensen, and Bruce Nordman. 2005. Managing energy consumption costs in desktop PCs and LAN switches with proxying, split TCP connections, and scaling of link speed. *International Journal of Network Management* 15, 5 (Sep. 2005), 297–310.

Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd, and Toshiba Corporation. 2009. Advanced Configuration and Power Interface Specification. Retrieved from http://www.acpi.info/DOWNLOADS/ACPIspec40.pdf.

Michael R. Hines and Kartik Gopalan. 2009. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'09)*.

Intel Corporation. 2006. Intel® Centrino® Mobile Technology Wake on Wireless LAN Feature: Technical Brief. Retrieved from http://www.intel.com/network/connectivity/resources/doc_library/tech_brief/wowlan_tech_brief.pdf.

Intel Corporation. 2011a. Intel® Virtualization Technology for Directed I/O: Architecture Specification. Retrieved from http://goo.gl/VscXC.

Intel Corporation. 2011b. PCI-SIG SR-IOV Primer: An Introduction to SR-IOV Technology. Retrieved from http://www.intel.com/content/dam/doc/application-note/pci-sig-sr-iov-primer-sr-iov-technology-paper.pdf. (Jan 2011).

Intel Corporation. 2011c. Vision Paper: Intelligent Desktop Virtualization. Retrieved from http://www.intel.com/content/dam/doc/white-paper/intelligent-desktop-virtualization-overview-paper.pdf.

Intel Corporation, Microsoft Corporation, and Toshiba Corporation. 1999. Advanced Configuration and Power Interface Specification. Retrieved from http://www.acpi.info/DOWNLOADS/ACPIspec10b.pdf.

Miguel Jimeno, Ken Christensen, and Bruce Nordman. 2008. A network connection proxy to enable hosts to sleep and save energy. In *27th IEEE International Performance Computing and Communications Conference*.

Asim Kadav and Michael M. Swift. 2008. Live migration of direct-access devices. In *1st Workshop on I/O Virtualization (WIOV'08)*.

Michael Kozuch and M. Satyanarayanan. 2002. Internet suspend/resume. In *4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'02)*.

Horacio Andrés Lagar-Cavilla, Joseph Andrew Whitney, Adin Matthew Scannell, Philip Patchin, Stephen M. Rumble, Eyal de Lara, Michael Brudno, and Mahadev Satyanarayanan. 2009. SnowFlock: Rapid virtual machine cloning for cloud computing. In *4th ACM European Conference on Computer Systems (EuroSys'09)*.

Joshua LeVasseur, Volkmar Uhlig, Jan Stoess, and Stefan Götz. 2004. Unmodified device driver reuse and improved system dependability via virtual machines. In *6th Symposium on Operating Systems Design and Implementation (OSDI'04)*.

Lieberman Software Corporation. 2006. White Paper: Wake on LAN Technology. Retrieved from http://www.liebsoft.com/pdfs/Wake_On_LAN.pdf.

Michael R. Marty and Mark D. Hill. 2007. Virtual hierarchies to support server consolidation. In *34th International Symposium on Computer Architecture (ISCA'07)*.

David Meisner, Brian T. Gold, and Thomas F. Wenisch. 2009. PowerNap: Eliminating server idle power. In *14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'09)*.

Miron Livny, Michael J. Litzkow, and Matt W. Mutka. 1988. Condor—A hunter of idle workstations. In *8th International Conference of Distributed Computing Systems (ICDCS'88)*.

Dejan S. Milojicic, Fred Douglis, Yves Paindaveine, Richard Wheeler, and Songnian Zhou. 2000. Process migration. *Computer Surveys* 32, 3 (Sep 2000), 241–299.

Sergiu Nedevschi, Jaideep Chandrashekar, Junda Liu, Bruce Nordman, Sylvia Ratnasamy, and Nina Taf. 2009. Skilled in the art of being idle: Reducing energy waste in networked systems. In *6th USENIX Symposium on Networked Systems Design and Implementation (NSDI'09)*. Boston, MA.

Gil Neiger, Amy Santoni, Felix Leung, Dion Rodgers, and Rich Uhlig. 2006. Intel® Virtualization Technology: Hardware support for efficient processor virtualization. *Intel® Technology Journal* 10, 3 (Aug 2006).

Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker. 1997. Agile application-aware adaptation for mobility. In *16th ACM Symposium on Operating System Principles (SOSP'97)*.

Oracle Corporation. 2012. Oracle Virtual Desktop Infrastructure. Retrieved from http://www.oracle.com/us/virtual-desktop-infrastructure-ds-067844.pdf.

M. L. Powell and B. P. Miller. 1983. Process migration in DEMOS/MP. In *9th Symposium on Operating Systems Principles (SOSP'83)*.

Shriram Rajagopalan, Brendan Cully, Ryan O'Connor, and Andrew Warfield. 2012. SecondSite: Disaster tolerance as a service. In *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'12)*.

Joshua Reich, Michel Goraczko, Aman Kansal, and Jitendra Padhye. 2010. Sleepless in Seattle no longer. In *2010 USENIX Annual Technical Conference*.

T. Richardson, Q. Stafford-Fraser, K.R. Wood, and A. Hopper. 1998. Virtual network computing. *IEEE Internet Computing* 2, 1 (Jan/Feb. 1998).

Michiel Ronsse, Koen De Bosschere, Mark Christiaens, Jacques Chassin de Kergommeaux, and Dieter Kranzlmüller. 2003. Record/Replay for nondeterministic program executions. *Communications of the ACM* 46, 9 (Sep. 2003), 62–67.

Constantine P. Sapuntzakis, Ramesh Chandra, Ben Pfaff, Jim Chow, Monica S. Lam, and Mendel Rosenblum. 2002. Optimizing the migration of virtual computers. In *5th Symposium on Operating Systems Design and Implementation (OSDI'02)*.

Brian K. Schmidt, Monica S. Lam, and J. Duane Northcutt. 1999. The interactive performance of SLIM: A stateless, thin-client architecture. In *17th ACM Symposium on Operating Systems Principles (SOSP'99)*.

L. H. Seawright and R. A. MacKinnon. 1979. VM/370—A study of multiplicity and usefulness. *IBM Systems Journal* 18, 1 (Mar. 1979), 4–17.

Siddhartha Sen, Jacob R. Lorch, Richard Hughes, Carlos Garcia Jurado Suarez, Brian Zill, Weverton Cordeiro, and Jitendra Padhye. 2012. Don't lose sleep over availability: The GreenUp decentralized wakeup service. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)*.

Eugene Shih, Paramvir Bahl, and Michael J. Sinclair. 2002. Wake on wireless: An event driven energy saving strategy for battery operated devices. In *8th Annual International Conference on Mobile Computing and Networking (MOBICOM'02)*.

Jacob Sorber, Nilanjan Banerjee, Mark D. Corner, and Sami Rollins. 2005. Turducken: Hierarchical power management for mobile devices. In *3rd International Conference on Mobile Systems, Applications and Services (Mobisys'05)*.

Ruben Spruijt. 2010. Local Storage for VDI Done Right–Part 1. Retrieved from http://www.brianmadden.com/blogs/rubenspruijt/archive/2010/11/27/vdi-and-storage-deep-impact.aspx.

Etienne Le Sueur and Gernot Heiser. 2010. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Workshop on Power Aware Computing and Systems (HotPower'10)*.

SunSpider Benchmark. 2012. SunSpider 0.9.1 JavaScript Benchmark. Retrieved from http://www.webkit.org/perf/sunspider-0.9.1/sunspider-0.9.1/driver.html.

Ajay Surie, H. Andrés Lagar-Cavilla, Eyal de Lara, and M. Satyanarayanan. 2008. Low-bandwidth VM migration via opportunistic replay. In *9th Workshop on Mobile Computing Systems and Applications (HotMobile'08)*.

Petter Svärd, Benoit Hudzia, and Johan Tordsson. 2011. Evaluation of delta compression techniques for efficient live migration of large virtual machines. In *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'11)*.

M. Theimer, K. Lantz, and D. Cheriton. 1985. Preemptable remote execution facilities for the v-system. In *10th Symposium on Operating System Principles (SOSP'85)*.

Niraj Tolia, Zhikui Wang, Manish Marwah, Cullen Bash, Parthasarathy Ranganathan, and Xiaoyun Zhu. 2008. Delivering energy proportionality with non energy-proportional systems—optimizing the ensemble. In *Workshop on Power Aware Computing and Systems (HotPower'08)*.

Andrew Tucker and David Comay. 2004. Solaris zones: Operating system support for server consolidation. In *3rd Virtual Machine Research and Technology Symposium (VM'04), Works-In-Progress*.

U.S. Department of Energy. 2013. Enegy-Efficient Computer Use. http://energy.gov/energysaver/articles/energy-efficient-computer-use. (Jan 2013).

U.S. Environmental Protection Agency. 2013. ENERGY STAR® Program Requirements for Computers. http://www.energystar.gov/ia/partners/product_specs/program_reqs/Computers_Program_Requirements.pdf. (Jan 2013).

VMware, Inc. 2013. Virtual Desktop Infrastructure. Retrieved from http://www.vmware.com/pdf/virtual_desktop_infrastructure_wp.pdf.

C. A. Waldspurger. 2002. Memory resource management in VMWare ESX Server. In *5th Symposium on Operating Systems Design and Implementation (OSDI'02)*. Boston, MA.

Andrew Warfield, Steven Hand, Keir Fraser, and Tim Deegan. 2005. Facilitating the development of soft devices. In *USENIX Annual Technical Conference (USENIX ATC'05)*. Anaheim, CA, USA.

Carrie A. Webber, Judy A. Robertson, Marla C. McWhinney, Richard E. Brown, Margaret J. Pinckard, and John F. Busch. 2006. After-hours power status of office equipment in the USA. *Energy* 31, 14 (Nov 2006), 2487–2502.

Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. 1994. Scheduling for reduced CPU energy. In *Symposium on Operating Systems Design and Implementation (OSDI'94)*.

Andrew Whitaker, Richard S. Cox, Marianne Shaw, and Steven D. Gribble. 2004. Constructing services with interposable virtual hardware. In *1st Symposium on Networked Systems Design and Implementation (NSDI'04)*.

Timothy Wood, Prashant Shenoy, K. K. Ramakrishnan, and Jacobus Van der Merwe. 2011. CloudNet: Dynamic pooling of cloud resources by live WAN migration of virtual machines. In *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'11)*.

Eric J. Wright, Eyal de Lara, and Ashvin Goel. 2011. Vision: The case for context-aware selective resume. In *International Workshop on Mobile Cloud Computing and Services (MCS'11)*.

Lei Xia, Jack Lange, and Peter Dinda. 2008. Towards virtual passthrough I/O on commodity devices. In *1st Workshop on I/O Virtualization (WIOV'08)*.

Edward R. Zayas. 1987. Attacking the process migration bottleneck. In *11th ACM Symposium on Operating System Principles (SOSP'87)*.

Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat. 2002. ECOSystem: Managing energy as a first class operating system resource. In *10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'02)*.

Edwin Zhai, Gregory D. Cummings, and Yaozu Dong. 2008. Live migration with pass-through device for linux VM. In *1st Workshop on I/O Virtualization (WIOV'08)*.

Songnian Zhou, Xiaohu Zheng, Jingwen Wang, and Pierre Delisle. 1993. Utopia: A load sharing facility for large, heterogeneous distributed computer systems. *Software—Practice and Experience* 23, 12 (Dec. 1993), 1305–1336.

Received February 2014; revised September 2014; accepted November 2014