Energy-Oriented Partial Desktop Virtual Machine Migration

by

Nilton T. Bila

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Energy-Oriented Partial Desktop Virtual Machine Migration

Nilton T. Bila

Doctor of Philosophy

Graduate Department of Computer Science

University of Toronto

2013

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 proven 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 solutions have been proposed that perform consolidation of idle desktop virtual machines. 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 simultaneously, each VM's experienced migration latency grows, and this limits the use of VM consolidation to environments in which only a few daily migrations are expected per VM. This thesis introduces *partial VM migration,* an approach that transparently migrates only the working set of an idle VM, by migrating memory pages on-demand. It creates a partial replica of the desktop VM on the consolidation server by copying only VM metadata, and transferring pages to the server, as the VM accesses them. This approach places desktop PCs in low power state when inactive and resumes them to running state when pages are needed by the VM running on the consolidation server.

Jettison, our software prototype of partial VM migration for off-the-shelf PCs, can

deliver 78% to 91% energy savings during idle periods lasting more than an hour, 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. In shorter idle periods of up to thirty minutes, Jettison delivers savings of 7% to 31%.

We present two approaches that increase energy savings attained with partial VM migration, especially in short idle periods. The first, Context-Aware Selective Resume, expedites PC resume and suspend cycle times by supplying a context identifier at desktop resume, and initializing only devices and code that are relevant to the context. CAESAR, the Context-Aware Selective Resume framework, enables applications to register context vectors that are invoked when the desktop is resumed with matching context. CAESAR increases energy savings in short periods of five minutes to an hour by up to 66%.

The second approach, the low power page cache, embeds network accessible low power hardware in the PC, to enable serving of pages to the consolidation server, while the PC is in low power state. We show that Oasis, our prototype page cache, addresses the shortcomings of energy-oriented on-demand page migration by increasing energy savings, especially during short idle periods. In periods of up to an hour, Oasis increases savings by up to twenty times.

Dedication

In memory of Jorge Francisco Bila.

Acknowledgements

I am indebted to many people for their contributions this work. First and foremost is Eyal de Lara, my advisor and long-suffering editor. This work has been shaped largely from frequent discussions with him, and from his steering of my research in one direction or another. I am very grateful for all the guidance that helped make this journey navigable. Satya (M. Satyanarayanan) has been my unofficial co-advisor whenever I needed one, and has provided additional guidance and insightful perspectives on this research.

This work has also been molded by my many collaborators, to whom I am greatly indebted. H. Andrés Lagar-Cavilla, Kaustubh Joshi, Matti Hiltunen, Ashvin Goel, Eric Wright, Eunbyung Park, Karan Dhiman, Maxim Mai, and Vishal Shrivastav, have all contributed the work being presented here. This has truly been a community effort.

My supervisory committee members, Bianca Schroeder and Khai N. Truong, as well as Angela Demke Brown and Orran Krieger have helped immensely to improve this work.

I am also grateful to the Systems and Networks Lab community, for so bravely use the software developed in this work on their desktops, and for the never-ending discussions that helped make graduate school fun. I will miss the quiet Sundays in the lab.

This is also an opportune moment to acknowledge the contributions of anonymous educators who along the way have helped shape my career. Of particular importance, are those at Li Po Chun United World College of Hong Kong, my alma mater, who have opened the doors to a new world.

Finally, I am very thankful to two wonderful women. Wanjiru, for her patience and never-ending support. This academic journey has been a long-term commitment to both of us. And my mother, for her support and counsel that have led me here.

Contents

Chapter 1

Introduction

The crowding of modern offices with personal computers (PCs) is now a fact of life. Previous studies by Webber et al. [104], Nedevschi et al. [77] and Agarwal et al. [29] have found that office PCs are idle for much of the day and yet 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 [77]. Unfortunately, an idle PC can consume up to 60% of its peak power. While modern computers support low power modes of operation, defined by the Advanced Configuration and Power Interface (ACPI) specification [1], the same studies have shown that the main reason these are not used is because of applications that require 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 are endowed with built-in mechanisms (e.g. Wake-on-LAN [71]) that allow remote hosts to bring the PCs out of low power modes on-demand. However, while these work for applications that provide remote access to the PC, they are ineffective for applications running within the PC that use a client-driven model

to 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. Further, the popularity of cloud computing has led to the proliferation of web-based applications that are client-driven. Asynchronous JavaScript and XML (AJAX) applications such as Facebook Chat [8], Google Docs [10], Gmail [9], are just a few examples of applications for which Wake-on-LAN-like mechanisms do not work.

1.1 Objectives

The goal of this thesis is to provide an energy conservation approach for idle PCs that is practical, scalable, and supports modern applications with always-on network presence. Specifically, this approach must meet the following requirements:

(i) Application transparency. Changes to applications in order to support energy conservation is not always possible. In particular, legacy and third party applications are often the hardest to update. Energy conservation approaches that are not compatible with these applications are unlikely to see widespread adoption.

(ii) Always-on application semantics. Users and administrators have been found to disable existing power management solutions in order to support applications that maintain network presence [29]. It is therefore imperative that our approach enables this class of applications to remain online, even during periods of energy conservation.

(iii) Low resume latency. For users, it is important that the desktop be immediately available for use on-demand, and without performance degradation. Solutions that impose high latency to resume the desktop or operate with degraded performance are unattractive to users, and are often dismissed [29].

(iv) Scalability. To work in modern offices, where hundreds to thousands of PCs are the norm, energy conservation solutions must scale well with the size of the enterprise. They must continue to deliver high energy savings and low resume latencies with modest investments in the infrastructure. The approach may only use network resources modestly and require minimal computing resources.

1.2 Partial Migration of Idle Desktop VMs

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 desktop to sleep [47]. The key advantage of this approach is that it does not require changes to applications or special purpose proxies. However, a straightforward implementation requires large network transfers, to migrate memory (and optionally disk) state, which can saturate shared networks in medium to large offices, and utilizes server memory inefficiently.

This thesis introduces *partial VM migration*, an approach that addresses these challenges. 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 1 MiB of disk state. Partial VM migration creates a partial VM on the server, and transfers on-demand only the limited working set that is accessed while the VM is idle. The desktop sleeps when the consolidated partial VM needs no state from it, and wakes up to service on-demand requests. We call these opportune sleeps *microsleeps*. Migrating the VM back to the user's desktop is fast because partial VM migration maintains VM residues on the desktop and transfers only the dirty state created by the partial VM back to the desktop.

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 desktop mode memory footprint, partial VM migration has the benefit that both, the network and server infrastructure, can scale well with the number of users, while providing migration times that 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 much sooner than expected, he hardly notices that his VM has migrated. This approach is suitable for personal computers, such as desktops and laptops, which have local execution state, and we refer to these simply as desktops.

Jettison is our software prototype of partial VM migration for off-the-shelf PCs. Our experience with a Jettison deployment 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 five hours the savings increased to 91%. Experienced migration times were near 4 seconds and migration sizes averaged under 243 MiB for Linux desktop VMs with 4 GiB of nominal memory. 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.

While our experience with Jettison demonstrates that it is able to save considerable amounts of energy, especially in idle periods of at least one hour, the savings have been smaller in shorter periods. In idle periods of ten to thirty minutes, Jettison delivered energy savings of 15% to 31%. We present two approaches that increase energy savings of partial VM migration during short idle periods. The first, Context-Aware Selective Resume, is a software solution that provides context awareness to PC wake-up function-

ality, and ensures that only devices and code paths required for the task of the wake-up context are initialized, providing fast wake-ups from low power modes. It ensures that during on-demand page transfer for partial VM migration, the PC spends minimal time powered up. Our experiments with a memory server based on CAESAR, the Context-Aware Selective Resume framework, show that this approach increases energy savings in idle periods of under an hour by up to 66%. In idle periods of ten minutes, it delivers energy savings of up to 44%. And in periods of twenty minutes, savings of up to 61%.

Our second approach, deploys a network accessible low power page cache in the hardware of desktop PCs, to ensure that memory state is accessible to network hosts, even when the PC is in low power. A low power page cache is useful for on-demand page migration of partial VM migration as it enables PCs to continue to save energy during page migration. We present Oasis, a prototype implementation of the low power cache architecture, that uses the Gumstix [17] computer-on-module platform to instantiate a cache. In idle periods of up to an hour, Oasis increases the energy savings of partial VM migration over the baseline, which wakes the PC up whenever a page is needed, two to twenty fold. In idle periods of ten minutes, the savings increase to 50%. In longer idle periods of twenty minutes and one hour, Oasis delivers savings up to 73% and 88%, respectively. Finally, in long periods of up to fourteen hours, Oasis delivers savings of up to 91%. Context-Aware Selective Resume and the low power page cache make energy savings in short idle periods practical.

While our current partial VM migration prototypes target 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) [65] 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.

1.3 Contributions

This thesis makes seven contributions: (i), it shows that the working set of idle Windows and Linux VMs is small and consists mostly of memory state (disk is less than 1%); (ii), it shows that migrating a VM in full is unnecessary, and indeed does not scale well for energy-oriented idle desktop consolidation; (iii), it shows that on-demand state requests are clustered enough to allow desktops to save energy by sleeping between request bursts; (iv), it shows that partial VM migration can save as much energy as full VM migration while sending less than 10% of the data, with migration latencies that are three orders of magnitude smaller; (v), it presents a complete architecture used to consolidate idle partial VMs and reintegrate them back to their desktop, when active; (vi), it shows that it is possible to optimize PC suspend and resume cycle times to lower the time and energy required to serve state, and that this approach results in additional energy savings; and (vii), it shows that by using a low power processor coupled with limited inexpensive storage, it is possible to serve desktop pages over the network while the PC is in low power, making on-demand page migration more energy efficient. The use of a page table data structure, enables fast page lookups, per-page compression, bandwidth pooling, and page persistence in the cache. All of these are strategies we use to address bandwidth and capacity limitations of inexpensive storage. Left untreated, low storage bandwidth can lengthen the duration of memory uploads from the PC to the cache, which limits the usefulness of the cache in short idle periods.

1.4 Organization

The remainder of the thesis is organized as follows. In Chapter 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 Chapter 3, we discuss desktop virtualization and its applications in this problem space and identify the challenges encountered with

solutions that rely on existing virtualization technology in enterprise offices. In Chapter 4, we characterize the behaviour of idle Windows and Linux desktop VMs with the goal of identifying opportunities to improve upon existing approaches. In Chapter 5, we introduce partial VM migration, our approach that is suitable for saving the energy used by idle desktops in enterprise offices. In Chapter 6, we introduce Jettison [39], our implementation of partial VM migration for off-the-shelf desktop systems, and present results from a deployment in a research environment and, using simulations driven by user activity traces, extend these results to large office environments and evaluate the scalability of the approach. 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 discuss considerations when deploying partial VM migration in enterprise environments. In Chapters 7 and 8, we present two alternative solutions. The first, Context-Aware Selective Resume [109], a software-only solution readily deployable on existing PCs, and the second, the low power page cache, a solution that encompasses software and hardware improvements to PCs. In Chapter 9, we discuss related work. Finally, in Chapter 10, we conclude the thesis and discuss avenues for future work.

Chapter 2

Background

In this chapter, 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 by Webber et al. [104] on the after-hours status of office equipment across U.S. office buildings finds that 60% of desktop computers remain powered in overnight hours, and only 4% of them use low power states. In contrast, only 20% of computer monitors remained powered, and about 75% were found to use low power states. In a more recent smaller study, Nedevschi et al. [77] report similar findings, with desktops remaining idle for more than 12 hours daily (not including off periods), and only 20% using low power states. These studies found that office users and administrators actively disable power management of their desktops. Desktop machines commonly ship pre-configured to enter low power states

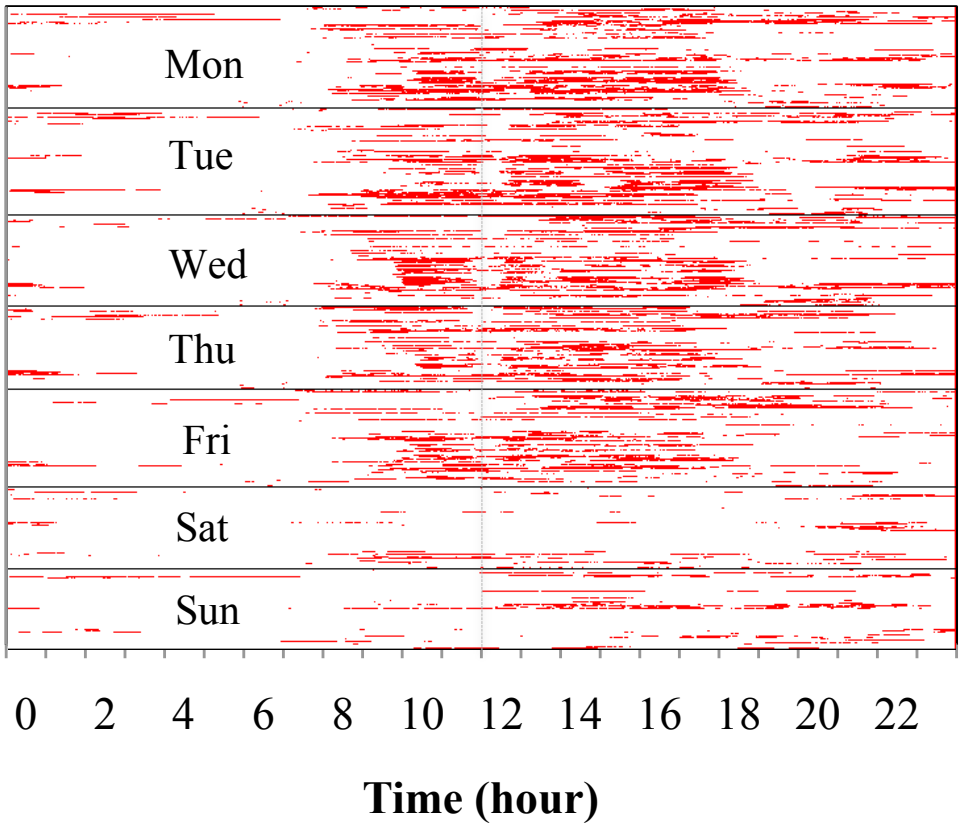


Figure 2.1: Desktop and laptop user activity. Each imaginary horizontal line represents a user. Active times are represented in red, and idle times in white.

whenever user inactivity is detected for a significant period. For example, by default, Windows 7 systems enter low power sleep state after 30 minutes of user inactivity.

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 5 seconds. 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 from the screen savers. Our tracker ran on Mac OS X, as 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 2086 person day traces from which a sample of 500 is presented in Figure 2.1. 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 that 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 2.1 shows that, in addition to the long overnight hours, there are significant idle times spread throughout the day, including many at lunch hour (12 pm), 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 indicates that there are substantial opportunities to save energy during work hours.

A survey by Agarwal et al. [29] sought to identify the reasons why people keep idle desktop PCs powered. The survey found that, among office users, the most oft-cited reasons were remote access and background applications. 52% of respondents left their systems powered to support remote access to files and software on the PC, 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.

The result of inefficient management of power use by idle PCs is considerable energy waste. A study commissioned by the California Energy Commission [41] finds that, in office buildings, IT equipment is the second largest consumer of electricity, at 20%, only surpassed by lighting, at 25%.

It is clear that idle monitors were much less a problem than PCs in Webber's study. Unlike PCs, monitor transitions to low power modes have no impact on running applications, and transitions to full power are low latency operations. 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 32 W. To obtain similar levels of success with PCs, they must emulate the properties of monitors, by allowing applications to run, even when in low power mode, provide low latency transition to full power mode, and deliver deep energy savings, even when the idle periods are short.

2.2 ACPI 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) [59], 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 state associated with full power off and boot ups. ACPI enables management of PC power consumption by the operating system (OS). The ACPI standard defines system power states as well as the power states for component devices needed to support the former. 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 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.

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

Table 2.1: ACPI power states and transition times for a Dell Studio XPS 7100 desktop. Standard deviations reported following ±.

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 the power states above, 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 2.1 shows the power use and transition times for a Dell Studio XPS 7100 desktop with a 3 GHz quad-core AMD CPU, 6 GiB 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 60.42 watts (W) to 2.33 W for S3, and 1.22 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 up takes 40.70 s, resuming from S3 and S4 only take 8.77 s and 26.29 s, respectively. Finally, the table shows that the faster low power states provide energy savings that are

comparable to full power off (differing by at most 1.1 W).

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.

While 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 Wake Up On-demand

To support remote administration applications, the PC industry developed technologies that enable PCs in low power states to be woken up by remote hosts on-demand. These technologies include Wake-on-LAN [71] and Wake-on-Wireless-LAN [62]. The former is found in most desktops and laptops with ethernet network interfaces, and the latter on laptops with 802.11 wireless interfaces.

In PCs with Wake-on-LAN, before suspending the PC into low power, the OS configures the network interface controller to remain powered during system low power state, and to wake the system up upon receipt of a designated network packet. In standard Wake-on-LAN mode, known as "magic" packet mode, this is an ethernet broadcast packet defined by the Wake-on-LAN protocol to contain the ethernet layer address of the PC's network interface, known as the Media Access Control (MAC) address. In Wake-on-

destined mode of operation, the NIC wakes the PC up 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 used by the BIOS to power up the system. Wake-on-Wireless-LAN functions in the same manner as Wake-on-LAN, except on the wireless 802.11 network interface.

Both these approaches were designed to allow system administrators to manage office desktops remotely. Administrators can remotely wake up enterprise desktops and apply software patches without physically visiting each desk. Similarly, applications for which requests are triggered by outside hosts (e.g., remote desktop access) can be served by wake up on-demand.

However, Wake-on-LAN and Wake-on-Wireless-LAN do not support applications that need to run on the PC while 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 IM and VoIP clients, but also DHCP clients that must renew the host's IP leases. For these applications, the host continues to fall off the network when in low power mode.

2.4 Dynamic Voltage and Frequency Scaling

To support background applications during idle times while reducing power use, CPU manufacturers developed techniques to dynamically modulate the power use of the processor, as a function of its operating frequency and voltage. Known as ACPI performance states or P-states, these states are accessed by the OS via the ACPI interface. The power drawn by computer chips can be broken down into static power and dynamic power. Static power results from current leakage in the circuit, while the dynamic power is attributable to the clock cycles performed by the processor. Dynamic voltage and frequency scaling (DVFS) [105, 57, 96] seek to reduce the dynamic power draw of the

chip. The power draw of a CMOS integrated circuit is governed by the equation [96]:

$$P = C \times V^2 \times F + P_{static}$$

Where C is the capacitance of the circuit, V is the voltage, and F, the clock frequency. While frequency scaling may not always lead to reduced energy use, because a slow chip takes longer to process workloads, low frequencies have lower voltage thresholds, allowing the processor to enter lower voltage operating points. Conversely, low voltage increases the settling time across the processor's gates, and, for correct operation the processor must lower its clock frequency. CPUs that support dynamic voltage scaling often can operate under various frequency/voltage points. For example, Table 2.2 lists the operating points for a mobile Pentium M processor, which supports Intel's SpeedStep technology [61].

Frequency (GHz)	Voltage (V)
1.6	1.484
1.4	1.420
1.2	1.276
1.0	1.164
0.8	1.036
0.6	0.956

Table 2.2: Supported performance states for the 1.6 GHz Intel Pentium M Processor.

For the Pentium M, Intel reports a 24.5 W draw in high frequency mode (1.6 GHz) and 6 W in the lowest frequency mode (0.6 GHz). More recent processors have been found to have smaller dynamic power ranges [96], as static power represents a larger fraction of dissipated power in smaller transistors with low threshold voltage. Even so, DVFS savings only apply to the CPU, and the CPU consumes only a fraction of the PC's power.

Component	Power (W)	Share (%)
CPU	80–140	25.6–31.3
Motherboard	50–150	19.5–27.5
Video (PCIe)	50–150	19.5–27.5
RAM (2 DIMM)	30	5.5–11.7
DVD	20–30	5.5–7.8
HDD	15–30	5.5–5.9
NIC (PCI)	5–10	1.8–2.0
Fans (2)	6	1.1–2.3
Total	256–546	100

Table 2.3: Estimates of per component power use for desktop PCs.

In Table 2.3, we estimate contribution of each PC component to the PC's peak power. These estimates come from PC Power & Cooling's recommended power supply unit (PSU) ratings, as a function of the components in a desktop PC [82]. The range given in the table, estimates power use for two types of desktop PCs, one with low end components and the other with high end ones. These desktops are both assumed to have a CPU, motherboard, PCI Express video card, two RAM DIMMs, a DVD player, a hard disk drive, a NIC and two fans. The table shows that while the CPU ranks amongst the largest contributors to a desktop's peak power, it consumes only 30% of the power. The motherboard and video card are also large contributors, and while the CPU has a dynamic power range given by DVFS, other components are found to be non-power proportional [36], and as a result their share of power use increases at idle times. Fan et al. [51] report similar findings in their measurements of power use on X86 servers. They find that the CPU accounts for 25% to 40% of the energy used.

With non-CPU components contributing to a large fraction of the PC's power use, the result is that DVFS has only a limited effect in reducing the energy use of idle PCs. The

60 to 70% of the power that is wasted by non-CPU components remains unaddressed. Moreover, even with DVFS a completely idle CPU wastes power. As shown above, for the Pentium M processor, DVFS can reduce power use by 75%. The remaining 25% of the peak power are wasted servicing both current leakage in the chip and its dynamic power.

2.5 Summary

In this chapter, we have shown that idle PCs are a major source of energy waste in office buildings. Office PCs are idle for vast amounts of time, averaging 12 hours daily. While idle, PCs remain powered to support background applications that maintain network presence. The result is that IT infrastructure is now the second largest consumer of electricity in office buildings.

Low power ACPI system states have been developed to enable PCs to lower their power use when idle. Because widely available low power system states disable CPU execution, a side effect of entering low power is that the system becomes network inaccessible. Wake-on-LAN was developed to enable on-demand wake up of PCs in low power for remote access. Low power states and Wake-on-LAN have proven insufficient to address idle PCs in the workplace. Low power states prevent applications with network presence to continuously communicate with network peers during idle periods. Low power CPU performance states, provided by DVFS, enable CPU instruction execution during idle times, but they yield savings that are only modest. DVFS has no impact on 60% to 75% of the idle power drawn by the PC's many components.

In the next chapters, we will discuss approaches that enable full power off of the CPU and the many components of the PC, by taking advantage of ACPI low power states, while enabling applications to continue to run.

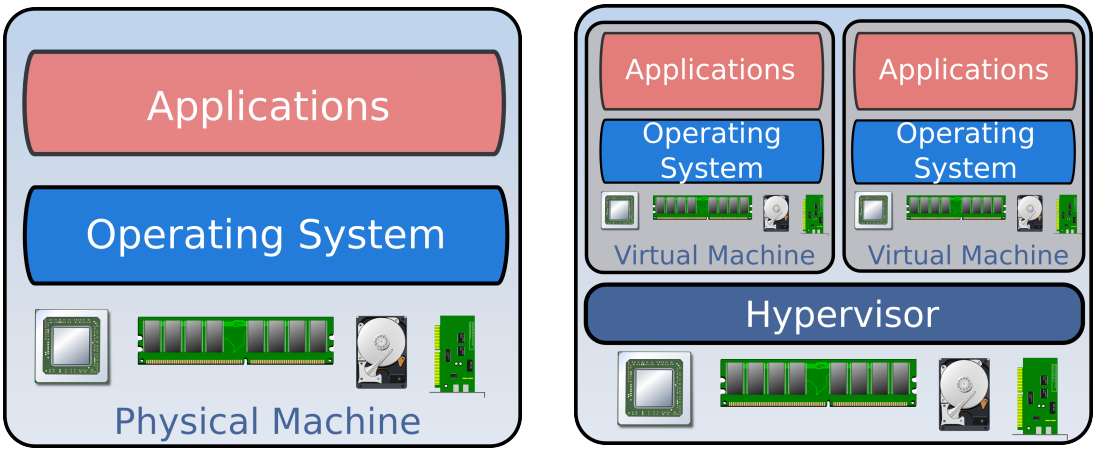
Chapter 3

Desktop Virtualization

In this chapter, 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.

3.1 Machine Virtualization

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) [89] 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 virtual machines (VMs). CPU is virtualized inside a virtual CPU (vCPU), which can be preempted by the hypervisor without the VM's awareness. Memory is isolated, and devices are handled by virtualized drivers (e.g., virtual disk for storage, virtual NIC for network, and virtual frame buffers for graphics).



(a) Legacy Machine (b) Virtualized Machine

Figure 3.1: Comparison of the architectures of a machine without virtualization (3.1(a)), and a machine with virtualization (3.1(b)).

Figure 3.1 shows the architecture of a legacy machine without virtualization (3.1(a)), and that of a machine with virtualization (3.1(b)). Virtualization enables the physical machine to run simultaneously multiple OS instances (e.g., a Windows VM with office applications, and a Linux VM with a software development environment), each with its own (virtualized) CPU, memory, disk and network interface. The hypervisor becomes the arbitrator of hardware access, to ensure all executing VMs have fair access, and provides isolation between VMs.

Besides hardware multiplexing, virtualization provides another benefit: it enables migration of VMs across physical hosts [42, 87, 68]. 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.

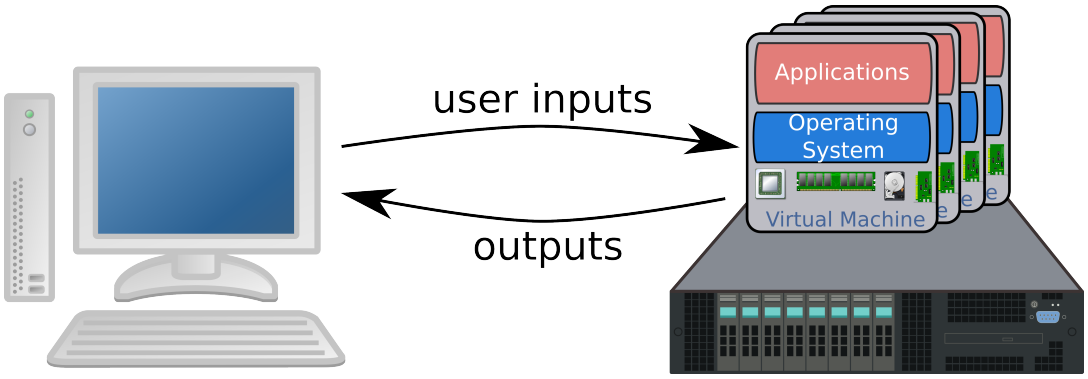


Figure 3.2: Overview of VDI.

3.2 Virtual Desktop Infrastructure (VDI)

VDI has emerged in the past decade as a means to simplify enterprise desktop management and deliver enterprise class storage services [34, 101, 43, 81]. 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. Because VMs run in shared servers under the control of IT administrators, the VMs can be configured to run from a single system disk image. Thus, many VMs can use the same OS and system wide software installation that resides in a read-only "golden" disk image. User data and configuration is stored on VM private disk images. This separation of system and user state ensures that administrators can perform maintenance tasks affecting many VMs by updating a single disk image. To update the software on desktop VMs, administrators only need to update the golden image.

Full time conversion to VDI can help reducing idle energy use because VDI enables the use of low power stateless thin clients with only enough hardware to display graphics generated on the server, and to 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, etc, which have been found to be energy inefficient under low loads [36].

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, that maintain network presence (which 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 [53]. 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 to come. VDI has limited effect in reducing energy use in these environments.

Hardware vendors such as Intel, now champion Intelligent Desktop Virtualization (IDV) [65], a different model of deployment of desktop virtualization. With this model, VMs run on the end user desktops in the same client-side manner advanced by Kozuch et al. [68] and Sapuntzakis et al. [87], but continue to rely on golden system disk images stored on enterprise class shared file systems. Software updates remain simple to apply, however, desktops must remain powered during idle times, in order to sustain always-on applications. The approach we discuss next enables idle desktops to enter low power modes during idle times while remaining compatible with the IDV model.

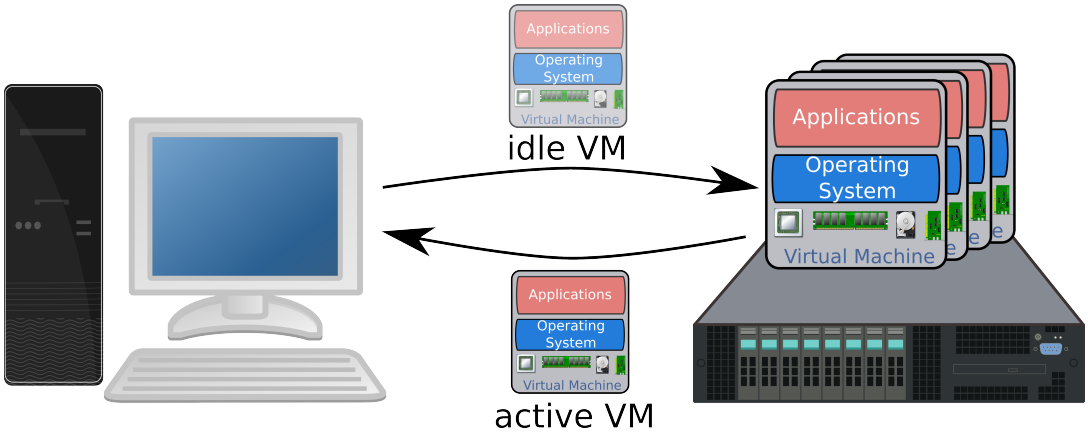


Figure 3.3: Overview of idle desktop VM consolidation.

3.3 Idle Desktop VM Consolidation

Recent research by Das et al. [47] propose 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 desktop hardware is placed in low power state. When the user returns to the desktop and interacts with the mouse or keyboard of the desktop, the VM is migrated back to the desktop. Figure 3.3 illustrates the nature of idle VM consolidation. This approach uses live VM migration [44], 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., desktop) to the destination (e.g., consolidation server), the VM continues to execute at the source. Only once a minimal amount of memory state remains to be migrated, execution is halted at the source, the remaining state (including CPU register context) is migrated, and execution resumes at the destination.

While this approach allows background applications to continue to run and maintain network presence, and 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 per-

formance over VNC [85] while migration completes, and this constitutes a hindrance to adoption in enterprise environments. In the next sections, we quantify the performance of existing VM consolidation technologies on enterprise class hardware, and show that they are lacking.

3.3.1 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 24 GiB of RAM, 8 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 4 GiB of memory and 12 GiB of disk images on top of Xen 3.4 hypervisor [35]. 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, which 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, the SunSpider JavaScript benchmark [12], as well as Acid3 Web standard compliance test [7]. 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 one minute, after which the VM was migrated to the consolidation server through a dedicated network switch. Each

experiment was repeated 5 times and we report the average results. Even though the footprint of the VM was 4 GiB, the script would consistently lead the VM to using only 1.2 GiB of its memory.

3.3.2 Results

Live migration took, on average, 38.59 seconds to migrate the desktop VM and consumed 4.27 GiB of network bandwidth. A user returning to her desk has to wait nearly 40 seconds 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: 4 VMs takes an average of 137 seconds, while 8 take 253 seconds. Staggering the resume times (so that VM migrations start moments apart) helps, but even with 20 second pause between resumes of 8 VMs, latency still averages 115.62 seconds. 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 [102] 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. While ballooning reduces the VM footprint, this happens at a considerable expense of time and I/O.

We repeated the VM migration experiments above, however, before performing migration, we used ballooning to reduce VM's memory footprint. In our experiments,

ballooning in Xen was able to reduce the idle VM's footprint to 423 MiB. To be able 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 ± 69.41 seconds to reach its saturation point, and in the process evicted 275.99 ± 9.25 MiBs of disk cached state and swapped out 449.08 ± 51.55 MiBs 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 423 MiB footprints were migrated, on average, in 4.86 seconds—the memory state swapped out and the cached disk state have to be reconstructed from the shared storage after resume resulting in additional network usage and slowed desktop responsiveness. Thus, while 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 in order to reduce network usage, the ballooning latency is still prohibitive and, in this case, the desktops must to be woken up to serve pages being swapped in while the desktop VM executes at the consolidation server.

Chapter 4

Understanding 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, ensuring that it can be readily used with existing software; and it supports access to local hardware. However, as we have shown in Section 3.3, 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 causes network congestion; (iii) consolidation ratios on the server is low, resulting in high infrastructure costs and low energy savings. In this chapter, we study the memory access behaviour of idle desktop VMs in order to understand the design space for effective VM consolidation.

We collected traces of memory page accesses of idle Windows 7 and Debian Linux desktop VMs under different desktop workloads [38]. With these traces we identify the working set of the idle VMs (how much of its full memory allocation these VMs access during idle time); and how frequent those memory accesses are. We use these traces to show that idle desktops require only a small fraction of their memory footprint, and that after an initial period of time, first time accesses to this memory can be infrequent. We argue that with these access patterns, we can migrate memory from the desktop to the

server on-demand, and still allow the desktop to spend a large fraction of the idle time in low power mode.

4.1 Methodology

To record memory access traces for the Linux VMs, we used SnowFlock's [69] 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 master 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 five minutes, used SnowFlock to create a replica of the VM on the server. We then halted execution on 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 9.06 MiB.

Because SnowFlock supports only paravirtual 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) [78] 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 which take advantage of hardware virtualization assists, including the EPT.

This study focuses on memory migration, which we found in Section 3.3 to be a challenge with existing technology. While disk is assumed to reside in network storage,

in the manner implemented with existing consolidation solutions, we will later show in Chapter 5, that the approach we develop to migrate memory can also be used to migrate disk state in environments where disk images reside on desktops. This is because idle VMs access very little disk state.

The next sections describe our experimental platform and the idle desktop workloads we studied. For each workload, we collected traces from three runs to ensure the results reported are consistent.

4.1.1 Platforms

SnowFlock is implemented on the Xen [35] 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 image was configured with 1 GiB of RAM and a 12 GiB disk. We collected our traces on a Dell Optiplex 745 with 2.66GHz Intel Core 2 Duo CPU and 4 GiB of RAM.

The Windows 7 VM was configured with 4 GiB of RAM and backed by 16 GiB of disk. This VM ran on Xen 4.2.0-rc4.

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 10 minutes.
IM	The Pidgin multi-protocol 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).

Table 4.1: Idle session workloads.

4.1.2 Workloads

Table 4.1 describes the workloads we studied. The workloads consist of typical desktop applications. Login illustrates the nightly behaviour 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 workloads, we were able to disable the default GNOME desktop session (which provides task bars, desktop management, etc). The micro-workloads are intended to give us a detailed look at the behaviour 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 behaviour of the desktop of a multitasking office worker who has become idle temporarily, and has not made any effort to exit any of her applications.

4.2 Working Set of Idle VMs

We begin by investigating the working set of idle desktop VMs. We define the working set of an idle VM as the memory pages that the VM accesses while it is idle. Figures 4.1, and 4.2 show the amount of memory that is accessed by each idle VM over the course of one hour. The figures show that, as each VMs begins idle time execution, it accesses pages causing their migration from the desktop. For Linux VMs, the Login workload accessed an average of 22.97 ± 1.94 MiB of pages across all runs. E-mail, Chat and Multitask, accessed an average of 52.32 ± 0.37 MiB, 55.69 ± 1.13 MiB, and 91.01 ± 9.52 MiB, respectively, where ±s denote the standard deviations. For all idle workloads, the working sets observed amount to less than 10% of the VM's nominal memory.

Similarly, idle Windows 7 VMs access an order of magnitude less pages than their total allocation. Across all runs, the Login workload averaged 196.66 ± 0.77 MiB of working set. E-mail workload averaged 372.80 ± 11.60 MiB, Chat averaged 336.03 ±

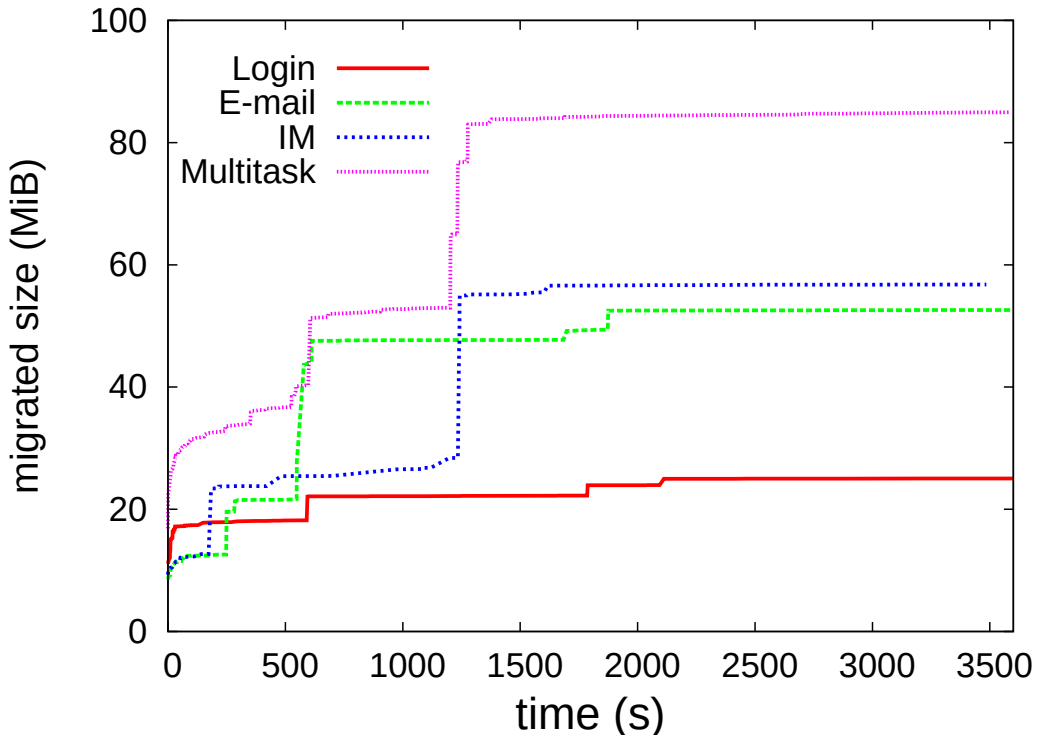


Figure 4.1: Cumulative size of migrated pages for Linux VMs.

12.66 MiB, and Multitask averaged 381.04 ± 37.19 MiB.

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 behaviour is explained by the fact that initially, when the VM begins execution on the consolidation server, none of the VM's pages are 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

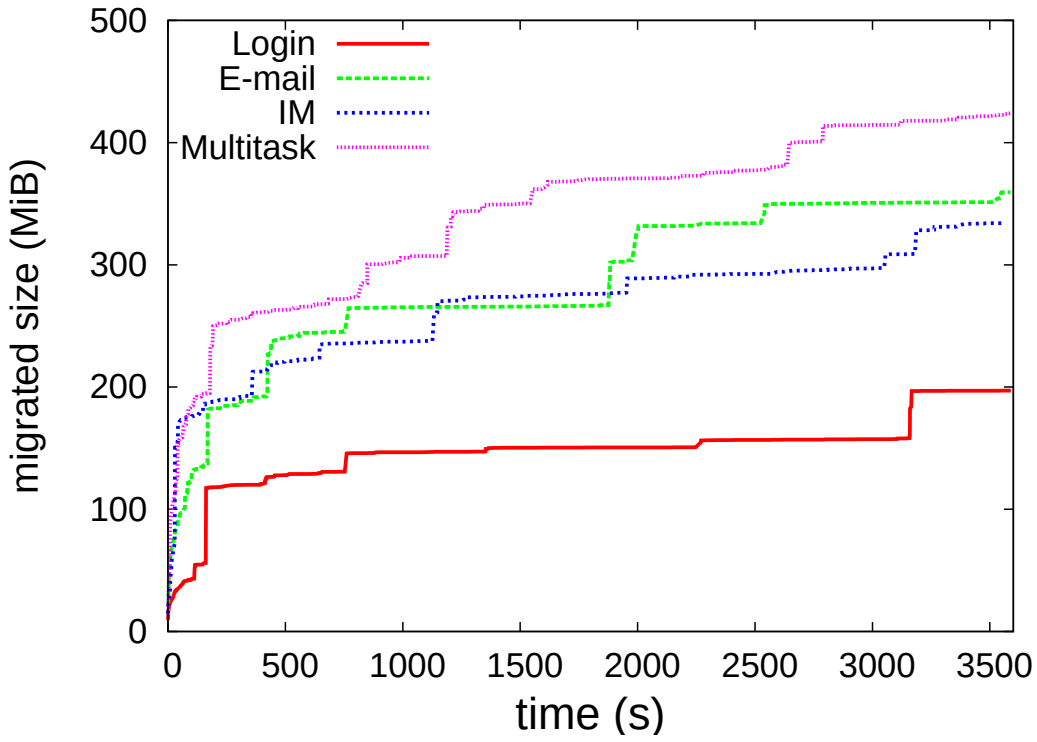


Figure 4.2: Cumulative size of migrated pages for Windows 7 VMs.

a large number of pages being migrated after nearly 10 minutes. This coincides with the server poll interval of the e-mail client. IM shows a similar burst after 20 minutes, which we believe to be caused by timer related events of the application. As 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.

Across the workloads, the Linux VM requires less than 92 MiB of memory pages out of its 1 GiB allocation, in order to run on the consolidation server for an hour. The Windows VM requires at most 391 MiB out of its 4 GiB allocation. With both OSes the VMs access an order of magnitude less pages when idle than their allocations. These results show that migrating the full VM memory footprint at consolidation time, as

performed by the idle VM consolidation approach described in Section 3.3, is inefficient. Instead, the insight we gain is that a better migration approach must migrate only the pages that are accessed by the idle VM. Doing so can ensure that migrations are fast, and can allow us to limit the VM memory commitment on the consolidation server, increasing consolidation ratios. The difficulty, however, is in determining *a priori* which pages must be migrated. Because modern desktops consist of complex applications with nondeterministic execution [86], it is difficult to accurately predict which pages they will access each time the VM is migrated to the server. We propose instead to migrate these pages on-demand, as the VM requires them. In order to migrate pages on-demand, the desktop must be accessible to service page requests. However, keeping the desktop PC running while the VM runs on the consolidation server is counterproductive, as it cannot save energy. Instead, we must explore the means to keep the desktop on only when pages are needed and off or in low power otherwise. This approach can only work if the desktop is able to stay in low power or off for significant periods. In the next section, we show that there are significant gaps between page requests that can allow the desktop to sleep and save power.

4.3 Desktop Sleep Opportunities

We begin this section with an analysis of the page migration patterns exhibited by the desktop VMs. Migration patterns show when pages are migrated from the desktop to the consolidation server. We use these patterns to identify *sleep opportunities*, the intervals in which no page is migrated and the desktop is able to transition into a low power state.

Figures 4.3 and 4.4 show the page migration patterns for the Chat and Multitask workloads in the Linux VM. These figures show locality in page migrations. Pages are migrated in bursts that include, application code, data, and kernel data, shown in the higher address spaces of the figures, and kernel code, in the low address space. The

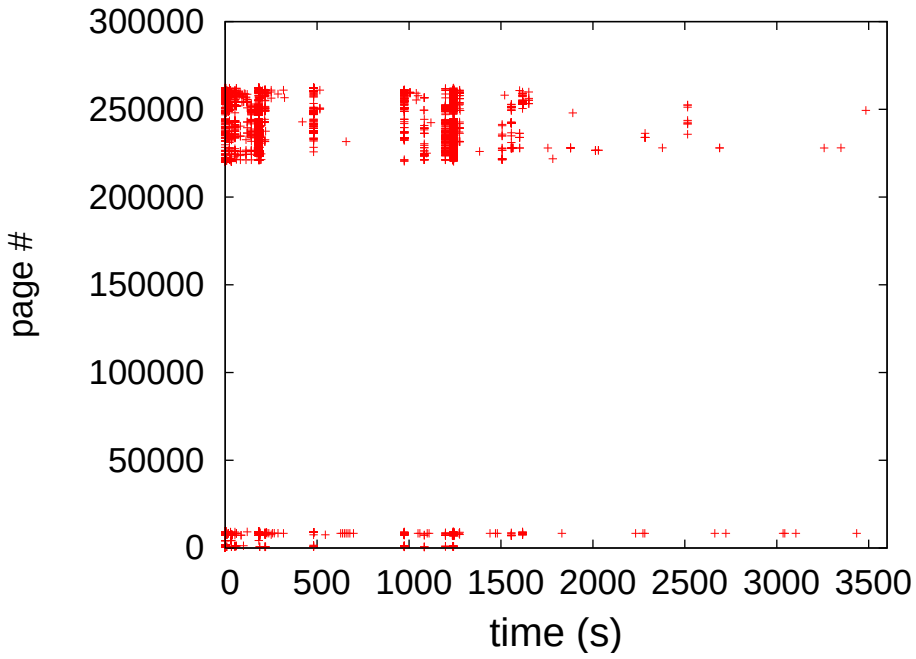


Figure 4.3: Memory page migration for IM workload under Linux.

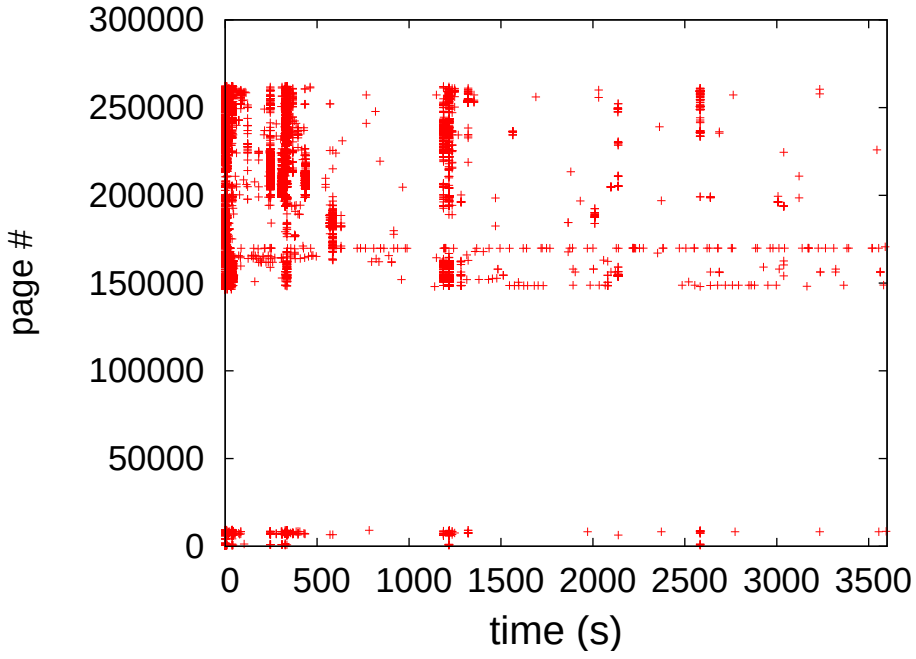


Figure 4.4: Memory page migration for Multitask workload under Linux.

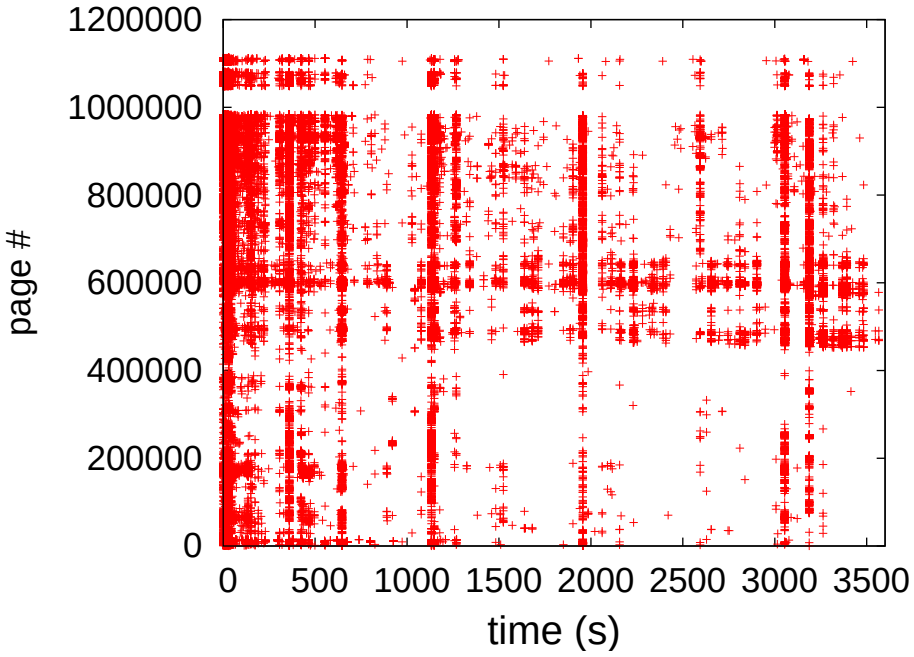


Figure 4.5: Memory page migration for IM workload under Windows.

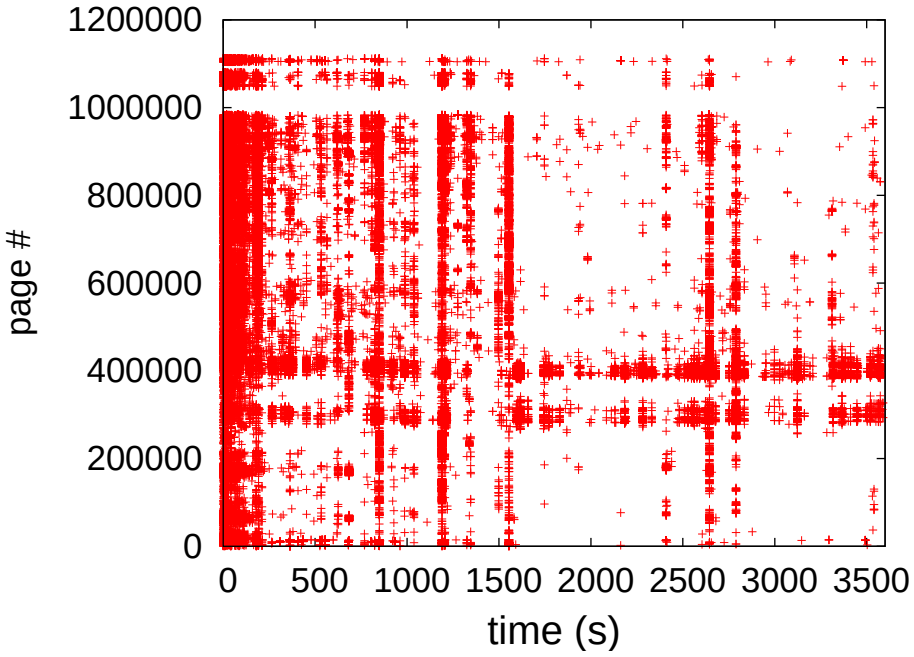


Figure 4.6: Memory page migration for Multitask workload under Windows.

workloads not shown here present similar patterns. These patterns, confirm the findings of Section 4.2, that page migrations are bursty: there are periods with no migrations, and when migrations do occur, often, a large number of pages are migrated at a time. Similarly, Figures 4.5 and 4.6, show that migrations for the Windows VM are also bursty.

Using the page migration patterns above, we now show that there are intervals in which no page is migrated, and that the desktop can take advantage of these intervals to enter a low power state. To identify these sleep opportunities, we divide the hour-long page migration traces into one second intervals. We label each interval in which a page is migrated as an active interval, and all others as inactive. We then identify all periods made of consecutive inactive intervals, that are long enough for the desktop to transition into low power and back to full power before the next page migration takes place. These periods are the sleep opportunities. Our analysis assumes that S3 is the low power state in use, and that the desktop transitions to that state in 7.24 s and back to S0 state in 8.77 s, as is the case with the Dell XPS 7100 desktops we profiled in Section 2.2. As a result, sleep opportunities are periods of at least 16.01 consecutive inactive seconds. Shorter or longer transition times decrease or increase the period of inactivity required for sleep opportunities, respectively.

Figures 4.7, 4.8, 4.9, and 4.10 show sleep opportunities available to a desktop hosting the Linux VMs over each hour-long run of the workloads in the server. The figures show sleep opportunities in red and awake periods in white. The figures show that, for most workloads, the desktop can spend a large fraction of time sleeping. For the micro-workloads, it can sleep, on average, for 41.39 to 44.38 minutes, and for the multitask workload it can sleep for a total of 16.81 minutes out of the 60-minute idle period. Each sleep interval lasts, on average, 59.74 to 77.79 seconds for the micro-workloads, and 16.44 seconds for the multitask workload, and their duration increases over time, as the replica VM requires less and less additional pages. While these workloads migrate between 6,413 and 21,777 pages, Figures 4.7,4.8, 4.9, and 4.10 show that the desktop only wakes up to

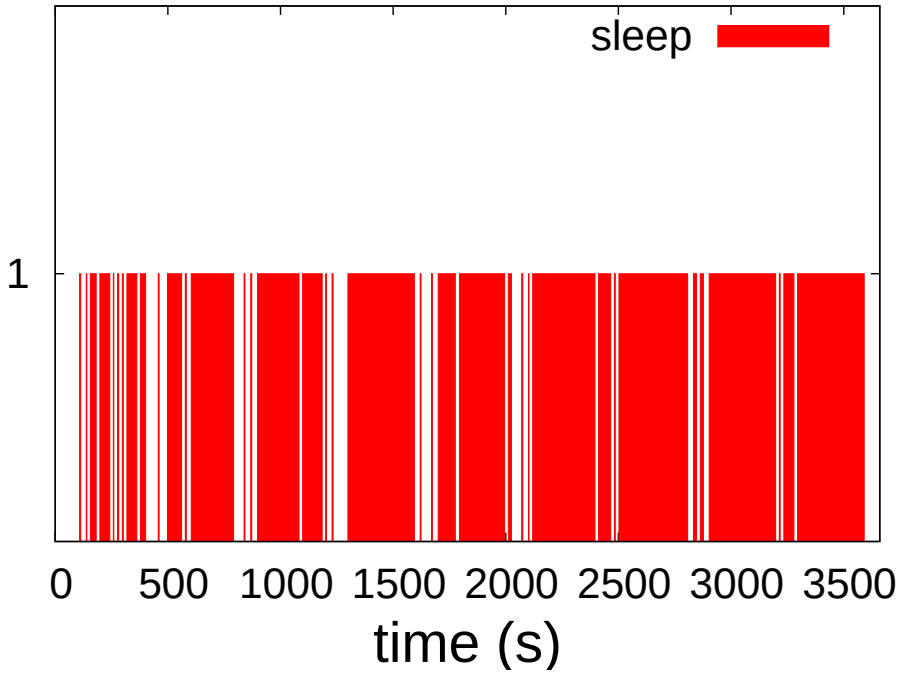


Figure 4.7: Potential desktop sleep intervals for Login under Linux.

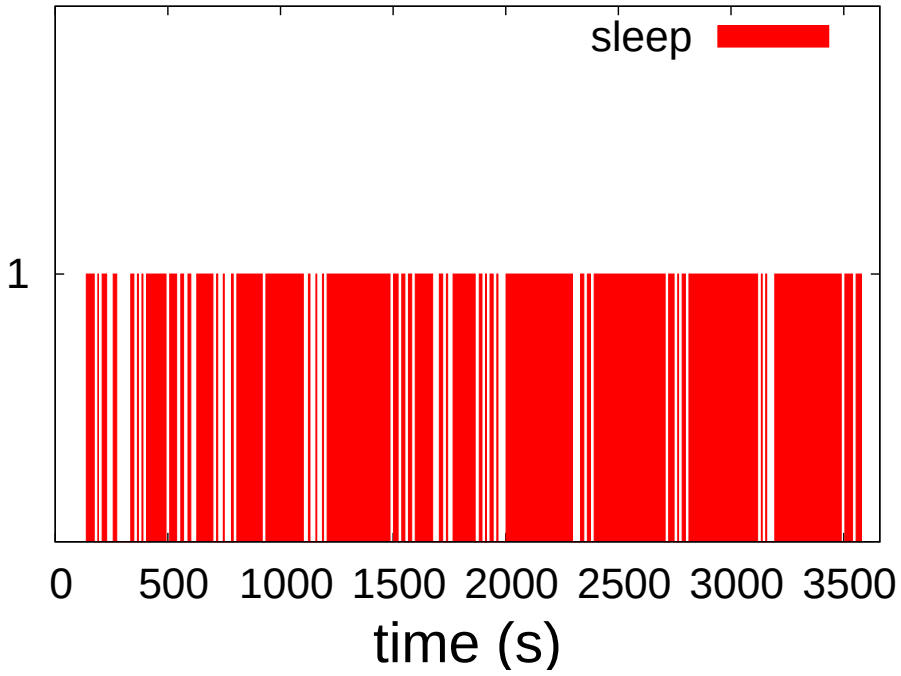


Figure 4.8: Potential desktop sleep intervals for E-mail under Linux.

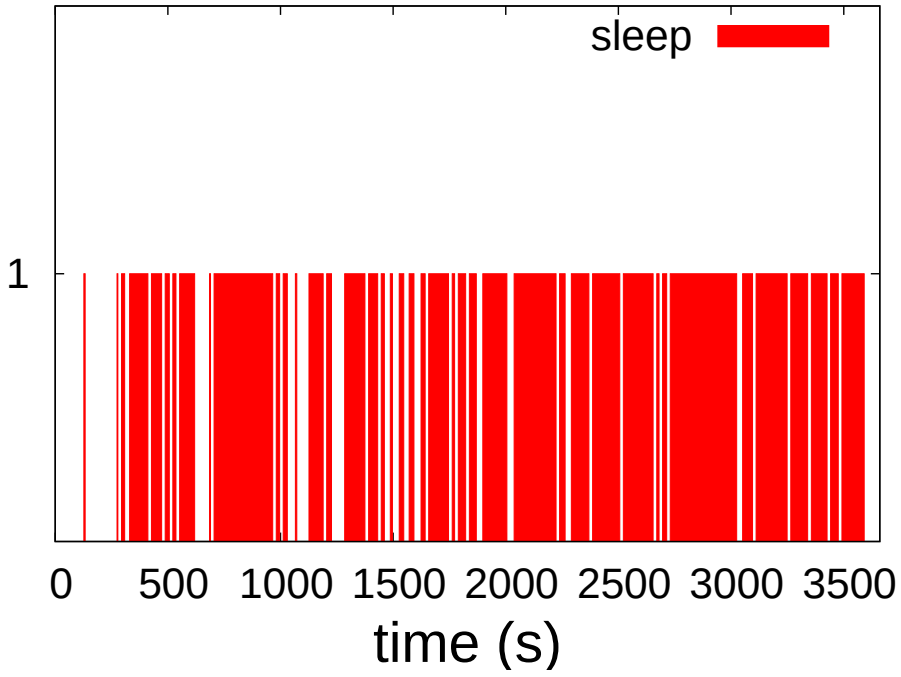


Figure 4.9: Potential desktop sleep intervals for IM under Linux.

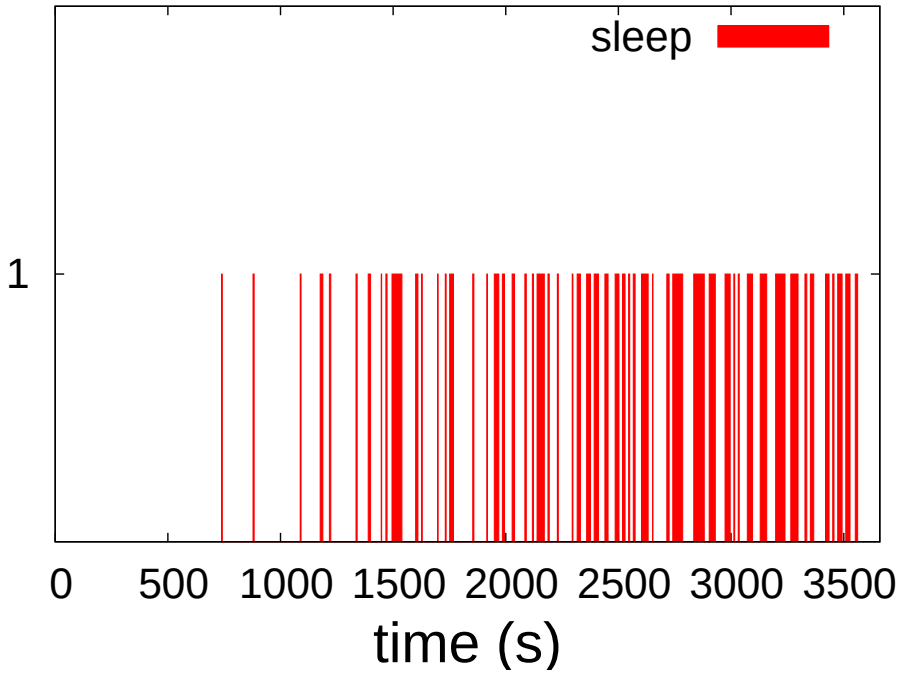


Figure 4.10: Potential desktop sleep intervals for Multitask under Linux.

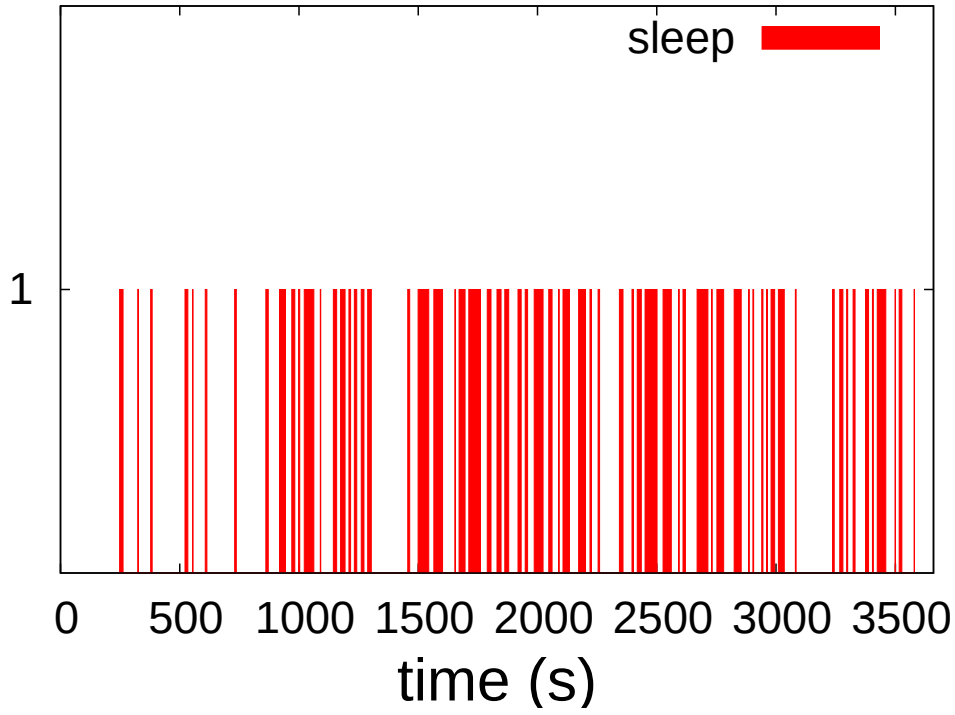


Figure 4.11: Potential desktop sleep intervals for Login workload under Windows.

service pages between 37 and 81 times. This confirms that the desktop is able to batch multiple page migration requests in each wake up.

Figures 4.11, 4.12, 4.13, and 4.14 show the sleep opportunities available to a desktop with the Windows VM. The desktop is able to sleep for 16.23 minutes, 10.95 minutes, 12.09 minutes, and 10.34 minutes, respectively for the Login, E-mail, Chat and Multitask workloads. The average durations of each sleep interval are 14.35 seconds, 12.50 seconds, 13.77 seconds, 12.12 seconds for the four workloads, respectively. While the Windows workloads present fewer opportunities than the Linux micro-workloads, these results are in line with those of the multitask workload in Linux. One of the differences between the Windows and Linux workloads is that, for the Linux micro-workloads, we were able to disable most system processes, including the desktop system, enabling us to run only the X Server and the target application under investigation.

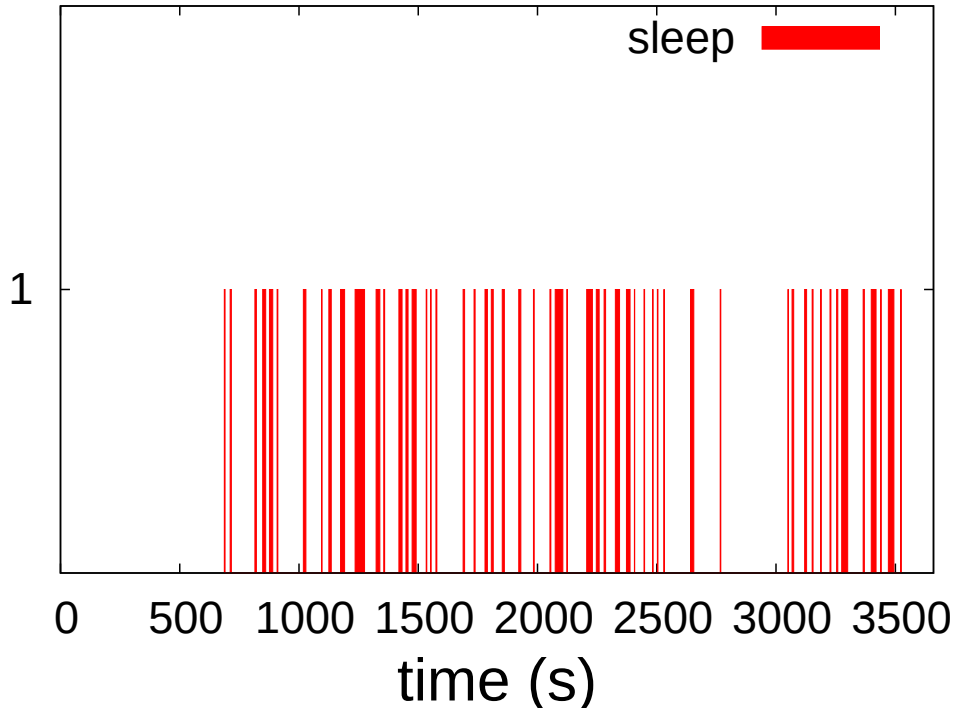


Figure 4.12: Potential desktop sleep intervals for E-mail workload under Windows.

4.4 Summary

The results obtained with Linux and Windows 7 VMs show that idle VMs have working set sizes that are far smaller than the VM's memory allocation (typically an order of magnitude smaller). These findings indicate that by migrating only the working set of idle VMs, we can reduce migration traffic, reduce migration latencies, and increase consolidation ratios on the consolidation servers. To ensure that we migrate only pages that are part of the idle working set, our proposal is that we migrate pages on-demand. Begin execution of the VM on the consolidation server with minimal state, and allow the VM to fault on access to pages that have not been migrated to the server. When faults occur, fetch pages from the desktop PC.

This approach requires that the desktop be up and running when pages are needed. We explored the potential for the desktop PC to sleep only in between page requests

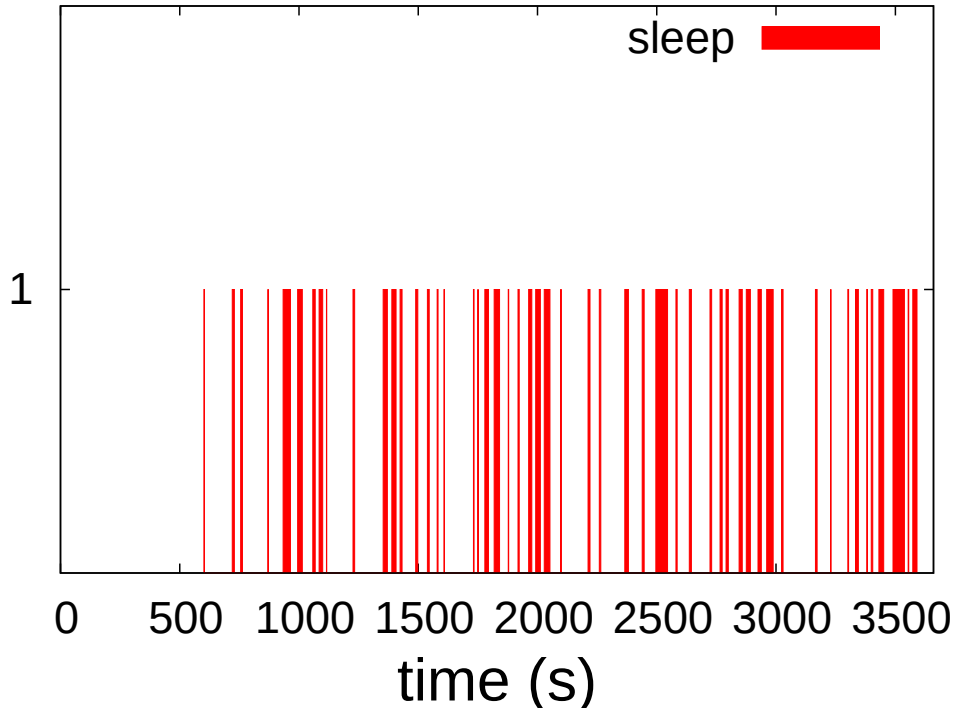


Figure 4.13: Potential desktop sleep intervals for Chat workload under Windows.

and found that there are significant opportunities to do so, particularly with low activity workloads. With multitasking workloads, while there are opportunities to sleep, we found them to be fewer.

In the following chapters, we introduce in detail our approach to idle desktop VM consolidation, that migrates memory pages on-demand. We devise strategies to ensure that desktops are able to sleep and save substantial amounts of energy, even in the presence of multitasking workloads, which we expect to be common on every day desktops.

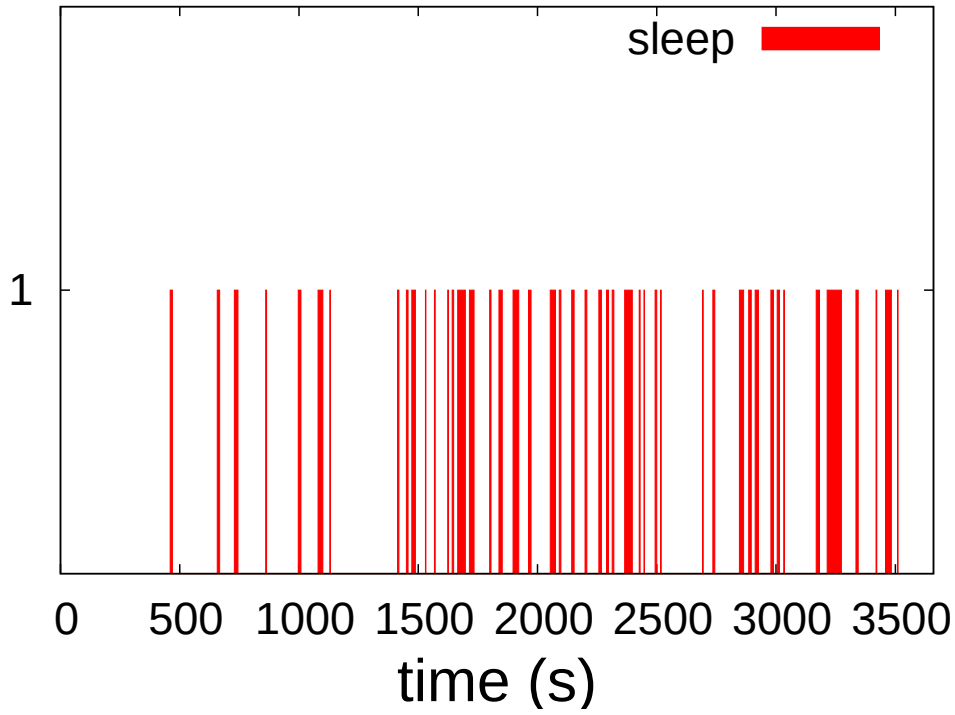


Figure 4.14: Potential desktop sleep intervals for Multitask workload under Windows.

Chapter 5

Partial Migration of 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 3.3, 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 only the memory and disk state that is accessed by the VM on-demand. The VM's pre-consolidation state remains as a residual on the desktop in anticipation of a reverse migration. When the user becomes active, partial VM migration transfers only dirty state (memory pages and disk blocks that have been modified on the server) to the desktop, and integrates it into the pre-consolidation state. Because state is transferred from the desktop to the server on-demand, the desktop only enters low power state in between transfers. We call these intervals *microsleeps*. Figure 5.1 shows an overview of the approach.

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, the desktop has all of the VM's state, which provides full system perfor-

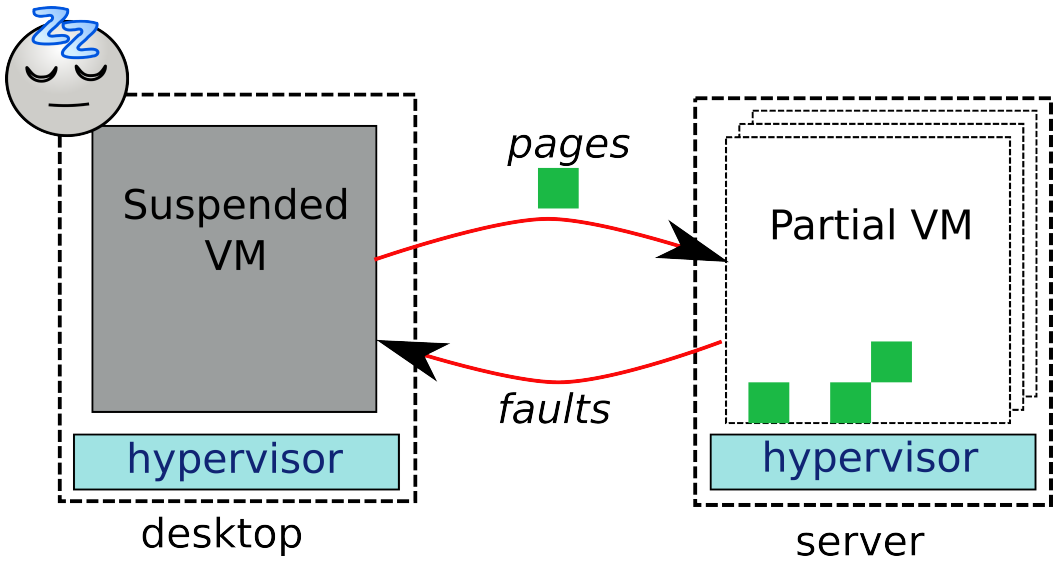


Figure 5.1: Overview of Partial VM Migration.

mance 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 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

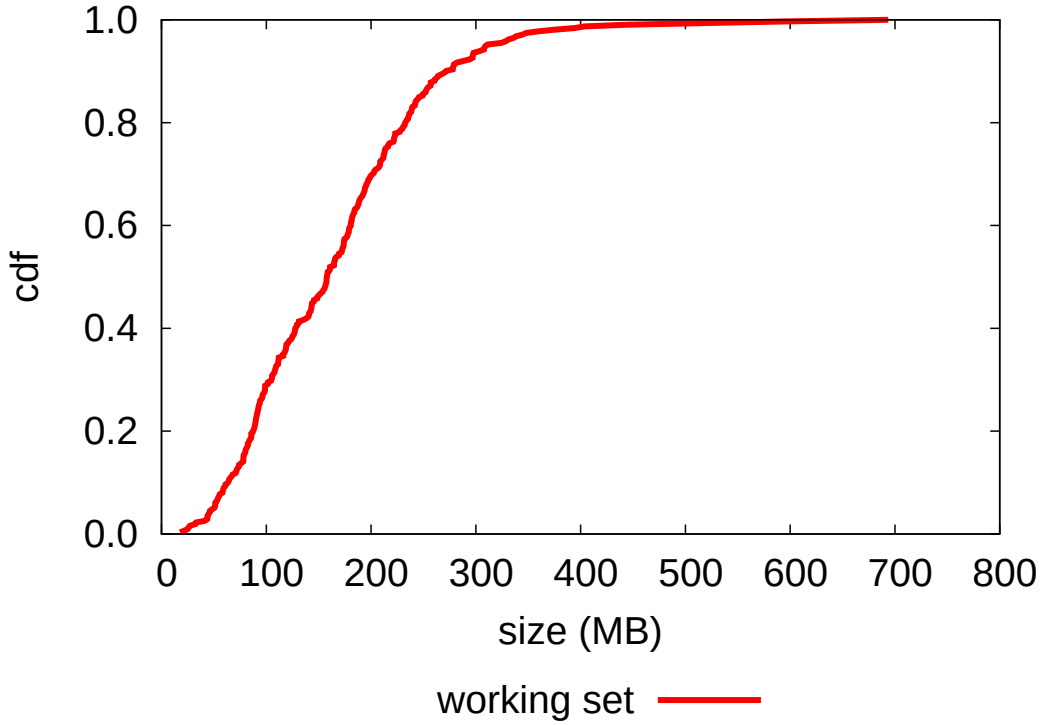


Figure 5.2: Distribution of working set sizes of idle Linux VMs with 4 GiB of memory.

desktop microsleeps only when it will save energy.

In the next sections, we first describe a deployment of desktop VMs we used to inform the development of our approach. We then describe the functional requirements for hypervisor support of partial VM migration, and the policies 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 5.5, we develop page prefetch strategies that maximize these microsleep opportunities.

5.1 State Access Traces

We collected traces of idle VMs memory and disk accesses in a deployment of our prototype implementation of partial VM migration on the desktops of three university researchers, over a period of seven weeks. We describe the details of the prototype in

Chapter 6.

5.1.1 Platform

Each user was given a desktop VM configured with 4 GiB of memory and 12 GiB 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, etc. 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 Dell Studio XPS 7100 systems, with 3 GHz quad-core AMD Phenom™II X4 945 processor, and 6 GiB of RAM each. The server was a Sun Fire™X2250 system with two quad-core 3 GHz Intel Xeon® CPUs and 16 GiB of memory. The desktops connected to the server over a shared GigE switch.

5.1.2 Methodology

Each user used their VM as their primary desktop over the duration of their participation. Every 15 seconds our system monitored keyboard and mouse activity, and when inactivity was detected a dialog, warning of an impending migration, was displayed for 5 seconds. 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.

5.1.3 Results

The first observation we make is that these traces confirm our hypothesis from Chapter 4 that the working set of idle VMs represents a small fraction of the VM's nominal memory.

Figure 5.2 shows the distribution of working set sizes of the three user VMs over 313 idle periods. The mean memory working set was only 165.63 MiB with standard deviation of 91.38 MiB. The mean working set size is barely 4.0% of the VMs nominal memory. The mean size of disk accesses during these idle times was 1.16 MiB with standard deviation of 5.75 MiB. The implications of a small memory and disk footprint are: (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 terms of reduced infrastructure costs.

5.2 Functional Requirements

To support partial VM migration, the hypervisor and its administrative tools must provide the following functionality:

(i) On-demand memory allocation. To achieve high consolidation ratios on the consolidation server, the hypervisor must support on-demand allocation of memory frames to VMs. The allocations must be performed transparently, without the aid of the VM's kernel, to ensure support for a variety of desktop OSes. When the idle VM begins execution on the server, the hypervisor must allocate only a minimal amount of memory required for the VM to run. As the VM requires more pages, the hypervisor must allocate these transparently, without an explicit request from the VM.

(ii) On-demand memory migration. To limit migration footprints, the hypervisor must support on-demand migration of memory state. At consolidation time, the hypervisor migrates only the VM's CPU context and configuration state (device configuration, memory limits, etc) and initiates execution of the partial VM replica on the server. Ac-

cesses to missing pages (remote faults) must be serviced transparently by the hypervisor, by migrating them from the desktop and re-executing the VM's instruction that caused the fault. On-demand memory migration ensures that VMs are migrated rapidly and do not lead to network congestion, a problem that can be acute in enterprise networks with hundreds of desktops.

(iii) On-demand storage. For VMs with disk images hosted on the desktop PC rather than network accessible shared storage, disk state must also be migrated from the desktop on-demand. At consolidation time, only device configuration is transmitted as part of the configuration state. First time access to a disk block, must cause the hypervisor to fetch the block from the desktop.

(iv) State residues on the desktop. As the partial VM runs on the server, memory and, optionally, disk state must remain on the desktop for two purposes: first, to allow the desktop to serve page and disk accesses; and second, to speed up future VM migration back to the desktop. For memory, this means maintaining the VM's full memory footprint in RAM, even as the desktop goes into low power state.

(v) Logging and reintegration of new state. First, the hypervisor must provide a mechanism to efficiently track pages and disk blocks that are updated when the replica VM runs on the server. Access to dirty pages and blocks enables migration of only the dirty state to the desktop, ensuring fast migrations with little network traffic. Second, the hypervisor must offer a means to update residual memory and disk state on the desktop, by integrating the dirty state migrated from the server.

(vii) Network migration. The hypervisor must provide a mechanism to support network migration across hosts to allow open network flows to persist across migrations. The goal of partial VM migration is to enable applications that maintain network presence

to continue to do so as the VM is migrated.

(viii) Support for a low power state and wake-up on-demand. To be able to reduce energy use of the idle desktop with partial VM migration, the hypervisor and desktop hardware must provide support for low power states, to be used when the VM is consolidated. Further, because the consolidated VM replica accesses desktop borne state on-demand, the network interface of the desktop must support on-demand wake-up from remote hosts.

(ix) 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 desktops 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) [48]. TLS is particularly suitable as it is a widely used application layer protocol that can be used to authenticate the endpoints of a stream, and encrypt the flows for the transport layer. The establishment of connections between a client and server using TLS follows a handshake process that establishes the authenticity of the server and client, and the parameters for encryption of the data to be transferred. Authentication is established through the use of certificates that we envision will be issued by the enterprise's IT administrator.

The third concern raised from migration of VMs into a shared consolidation server is if a VM containing malicious code is able to breach its memory, disk or device driver container to access other VMs' state. This problem is not unique to partial VM migration,

but rather common to multitenant virtualized environments. As a result, hypervisors have been designed to provide robust isolation between VMs, which we expect to be appropriate for our use.

Because many of the requirements listed here are not satisfied by existing hypervisors, in Chapter 6, we describe our modifications to the Xen hypervisor to provide support for partial VM migration. While our prototype supports most of the functional requirements, we leave the implementation of network security for future work.

5.3 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. Below we describe the conditions for consolidation and reintegration of desktop VMs.

5.3.1 Consolidation Conditions

With partial VM migration, we consolidate a desktop VMs 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 cancel 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. Because a partial VM requires only a fraction of its nominal memory, we must estimate the size of its working set before migration. Initially, we estimate the working set size from the observed sizes of VM traces. Over time, this estimate can then be tailored to each individual VM on the VM's previous history on the server. If during server-side execution, the VM requires less memory, some, though minimal, server memory is left un-utilized. If 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.

5.3.2 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., she 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., VM requests memory when server has full commitment). In addition, we must migrate a VM back to the desktop if (iv), the **VM generates a UI event** that may require the user's attention. For example, when a VoIP client running on the consolidated VM receives a phone call, and issues a ring tone through the sound device, we must intercept this event and trigger a migration to the desktop so the user can hear the incoming call.

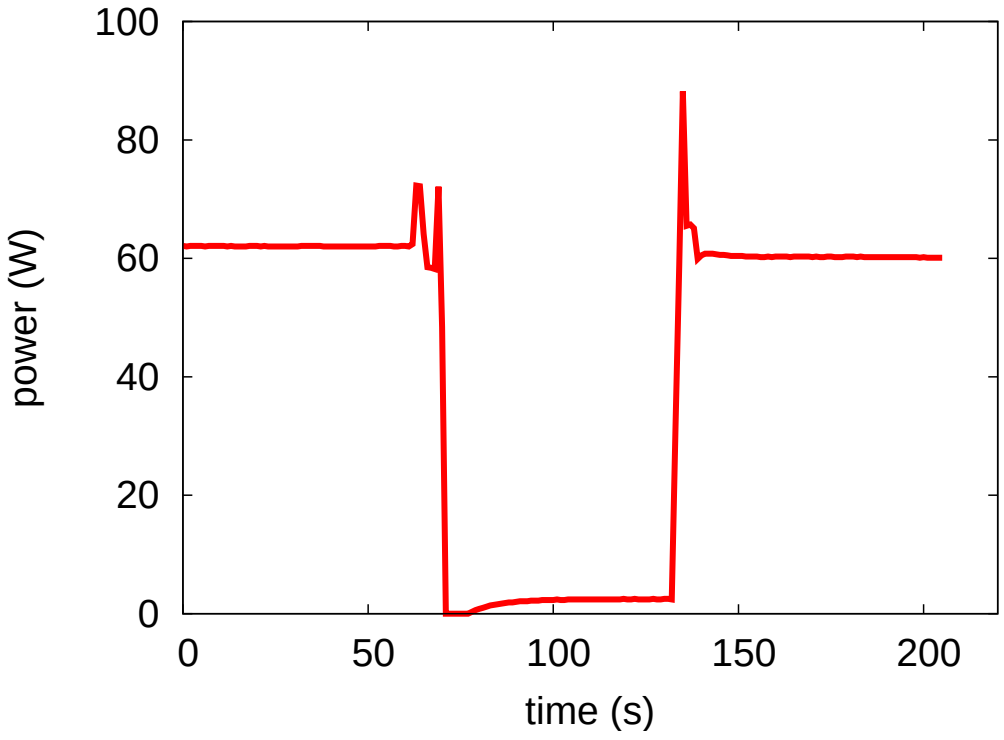


Figure 5.3: Power used by a Dell Studio XPS 7100 desktop during various power states. The desktop is initially idle for 60 s, then enters S3 state, where it remains for 60 s. Subsequently the desktop resumes to idle state.

5.4 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. However, deciding when desktops microsleep is not trivial because 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. In this section, we design a strategy for determining when the desktop can microsleep that seeks to minimize energy waste. It uses the observed page request interarrival distribution of idle desktop VMs, and the power profile of the desktop to reduce the probability of initiating a microsleep that is too short.

A desktop system experiences increased power use during transitions to low and on power states. Figure 5.3 shows the power used by a desktop that is idle for 60 seconds, then transitions into low power S3 mode. The desktop remains in this state for another 60 seconds, 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 62 W when idle up to 72 W and 88 W 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 use 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:

$$E_i = P_i t_i \tag{5.1}$$

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:

$$E_\mu = P_i t_{i'} + P_o t_o + P_s t_s + P_r t_r \tag{5.2}$$

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 interval the system wastes no time awake, so $t_{i'} = 0$, and the interval is given by:

$$t_b = \frac{-P_s(t_o + t_r) + P_o t_o + P_r t_r}{P_i - P_s} \tag{5.3}$$

Plugging in the power profile measurements for the Dell Studio XPS 7100 desktops in Table 6.1, we find that, for these systems, $t_b = 32.22$ seconds. Thus, these desktops should microsleep only when there is an expectation that no remote faults will arrive within the next 32.22 s.

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 inter-arrival I being energy inefficient.

Figure 5.5 plots the conditional probability that the next remote fault will arrive in less than 32.22 s based on remote fault inter-arrival times shown in Figure 5.4 from our

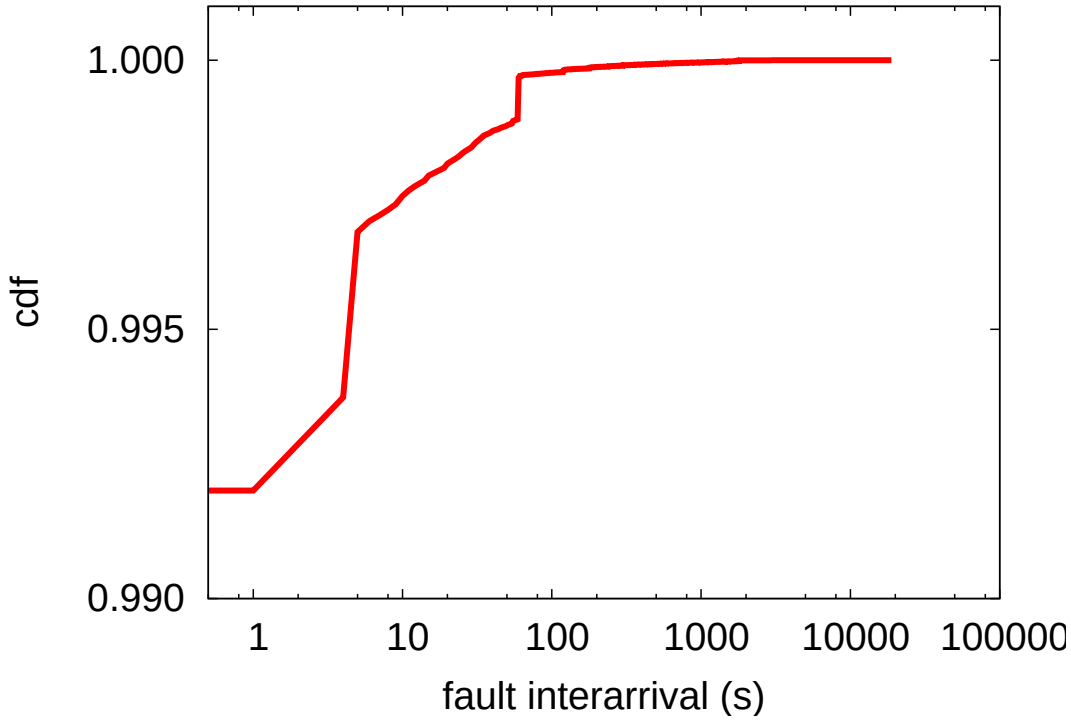


Figure 5.4: Remote fault interarrival distribution of idle desktop VMs.

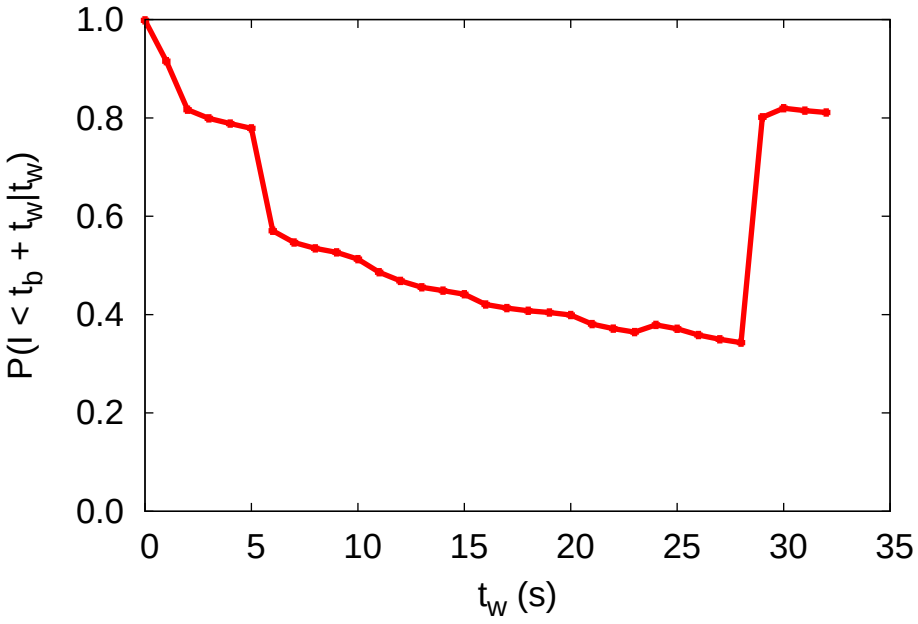


Figure 5.5: Conditional probability that the next remote fault will arrive in less than 32.22 s as a function of the wait time.

prototype deployment. Figure 5.5 shows that as the wait time increases up to 28 s, the likelihood of seeing the next remote fault in less than 32.22 s decreases rapidly. This is because faults are highly correlated, and indeed, as shown in Figure 5.4, more than 99.23% of remote faults occur within one second of previous faults. The implication is that for the vast majority of faults, when the desktop wakes up to service one fault, it will likely be able to service faults that follow immediately, avoiding many inefficient microsleeps. With wait times immediately above 28 s, the probability of seeing a remote fault increases significantly because 60 s inter-arrivals 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:

$$\min E_{waste}(t_w) = t_w E_i + p(I < t_b + t_w | t_w) E_\mu \tag{5.4}$$

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 $t_{i'} = 0$.

With our desktops' energy profile, E_{waste} is shown in Figure 5.6, and it is clear that the energy waste minimizing t_w is 6 seconds.

5.5 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, not requiring the desktop to be awake.

Most state transfer (in excess of 99%), hence remote faults, is caused by memory accesses (see Section 5.1.3). 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 5.7 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 which runs through page access traces we collected in our deployment, and allows us to impose indirectly different limits to the size of memory that is 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 83.62 seconds to 136.64 seconds. On-demand prefetch increases it to 150.82 seconds.

Figure 5.8 presents a CDF of microsleep durations for both strategies, with settings that migrate close to 500 MiB of memory (on-demand: $pivot\ window = 20\ pages$, $migration\ size = 492.19\ MiB$; hoarding: $limit = 480\ MiB$, $migration\ size = 490.83\ MiB$). The figure shows that while hoarding reduced the incidence of energy inefficient mi-

crosleeps (shorter than 32.22 seconds) from 55.51% to 48.43%, on-demand prefetch led to more microsleeps with long durations. Without prefetching, 10.34% of microsleeps had durations longer than 47 seconds. Hoarding was only able to increase these modestly to 11.11%. On-demand prefetch doubled the share of these microsleeps to 20.13%.

Figure 5.9 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 5.2. 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 microlseep durations, energy savings also improve. Frequent long lasting microsleeps deliver better savings for on-demand prefetch, albeit modestly.

We also experimented with an approach that combines hoarding and prefetch, and found it to yield an improvement in energy savings for a given migration size of only 1 to 4% over on-demand prefetch alone. However, 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, on-demand prefetch is preferable.

Separately, we also explored maintaining VM residuals on the server and found it to have limited success in reducing remote faults. In our experiments, page content reuse across subsequent consolidations of the same VM, averaged 28%. Fewer than one third of a VM's page requests could be serviced from residuals stored from previous server instances of itself. While we did not explore page content sharing across different VMs, we expect its effectiveness in reducing remote faults to be no better than with same VM residuals.

In subsequent chapters 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 50 MiB, as prefetched pages are commonly used within a short period, and with such buffer we see little re-fetch of evicted pages.

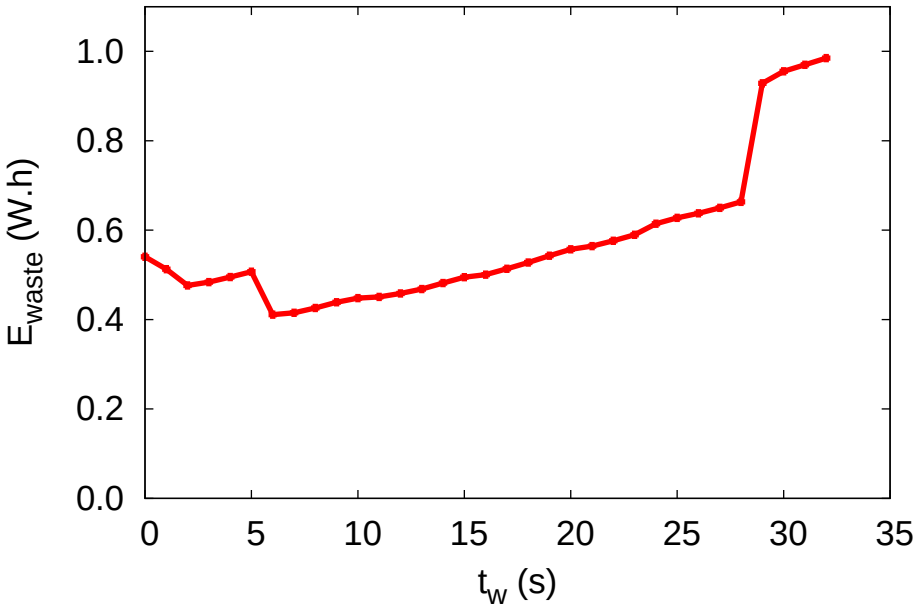


Figure 5.6: Expected energy waste as a function of sleep timeout (t_w).

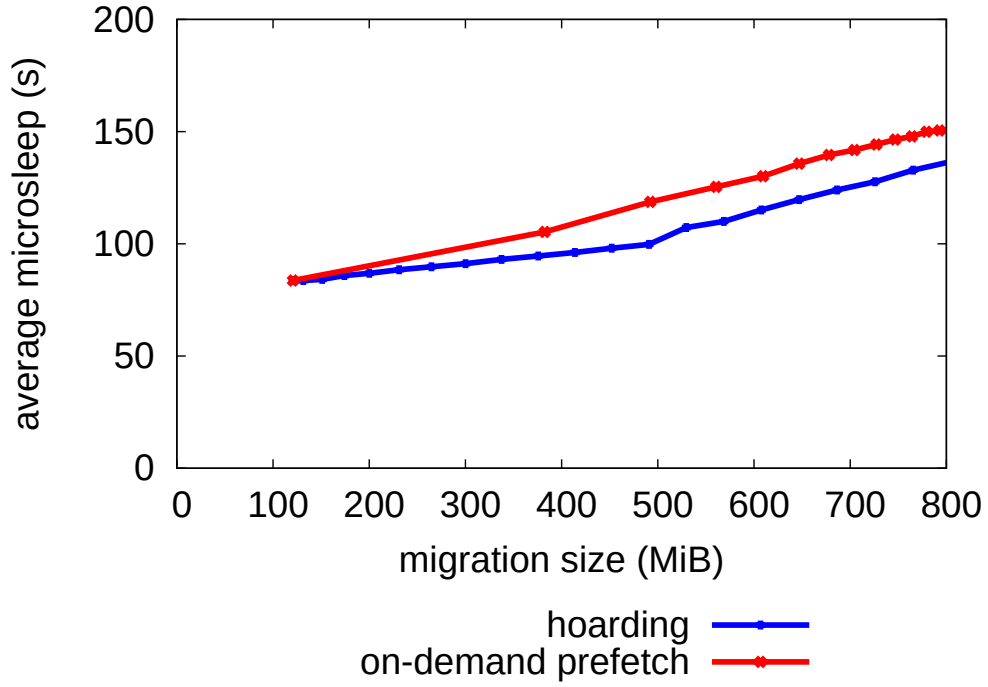


Figure 5.7: Effect of hoarding and on-demand prefetching on microsleep lengths.

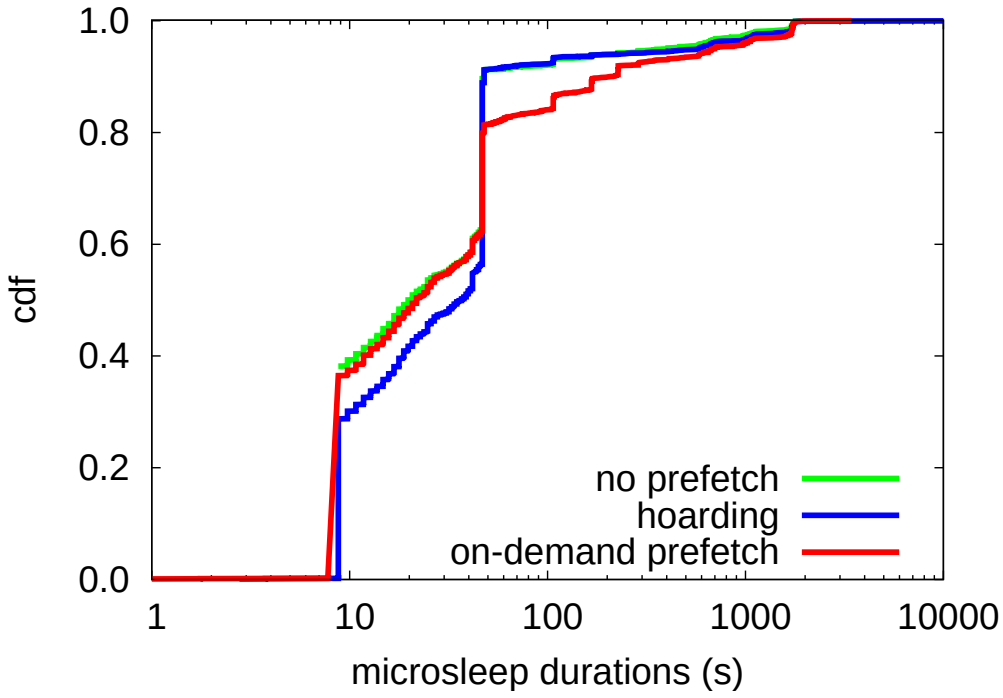


Figure 5.8: CDF of microsleep durations.

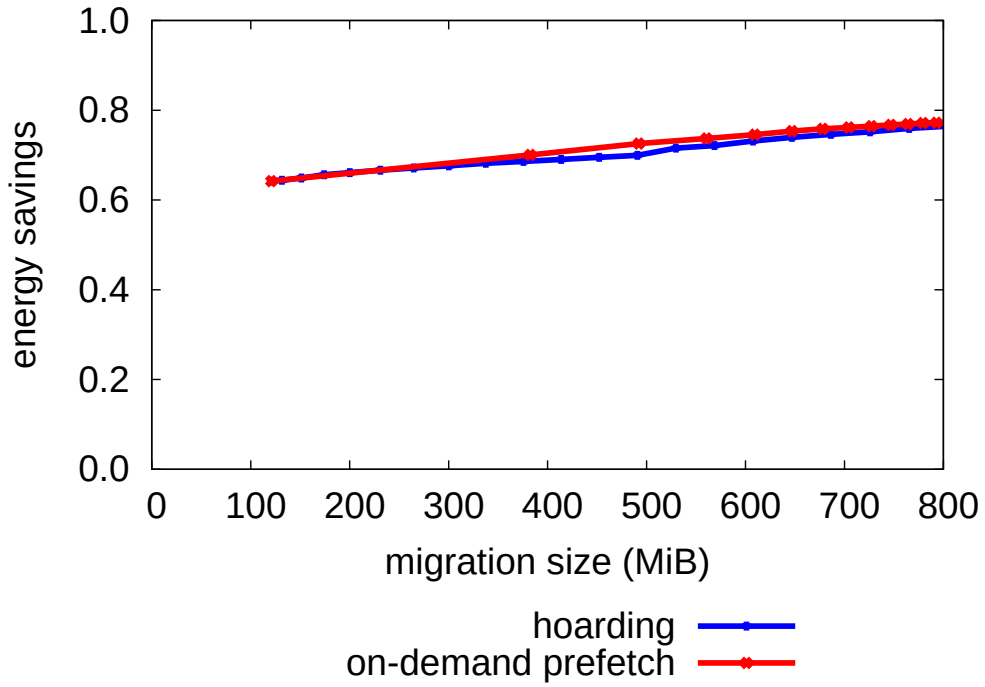


Figure 5.9: Effect of hoarding and on-demand prefetching on energy savings.

Chapter 6

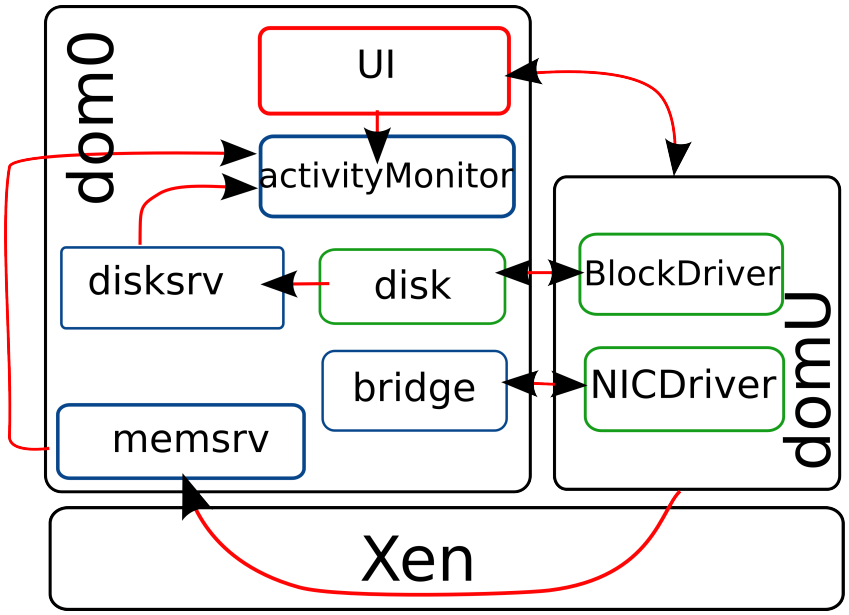
Idle Desktop Migration with Jettison

In this chapter, we present details of Jettison [39], 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.

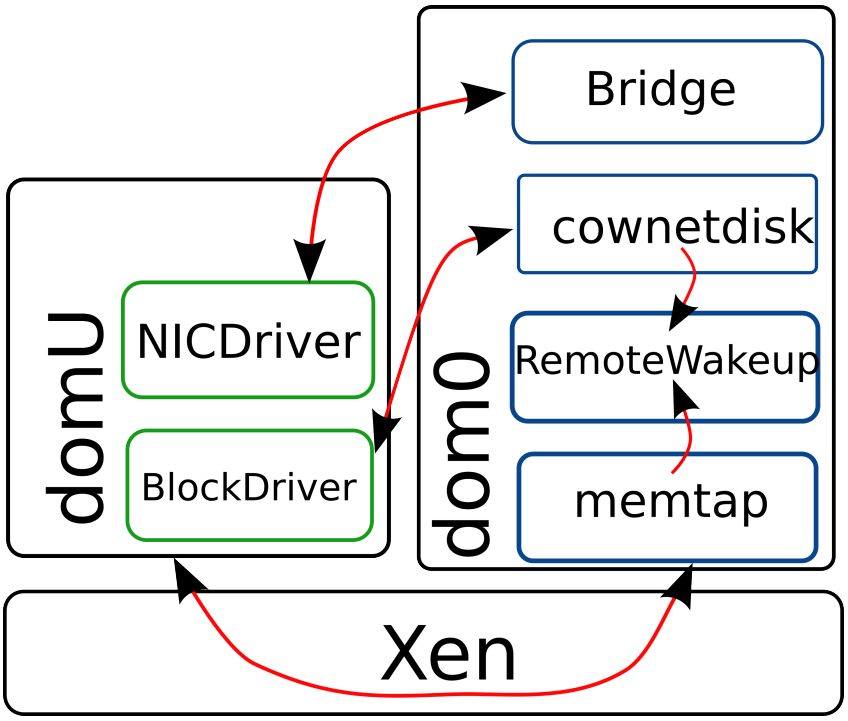
6.1 Jettison Prototype

Jettison extends the functionality of Xen 3.4 hypervisor [35]. Xen is a Type 1 or native [56] hypervisor which runs in the highest processor privilege level and relegates guest domains to lower privilege ones. Xen supports an administrative guest, domain 0 (henceforth dom0) and multiple unprivileged guests, 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



(a) Desktop



(b) Server

Figure 6.1: Architecture of Jettison.

supports paravirtualized guests, and we leave support of fully virtualized VMs for future work. In the discussion below, 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 6.1. On the dom0 of the desktop systems, Jettison runs the following components.

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 is run only when the VM is consolidated.

disksrv. A daemon responsible for serving disk blocks to the server over TCP. disksrv opens the VM's disk image in read-only mode.

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 5.4, 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 pre-configured 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. While 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.

At reintegration time, Jettison also performs updates to VM memory and disk state using received dirty state. Reintegration of memory state is supported in the hypervisor as well as with a *memory_restore* utility.

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

memtap. A process that monitors VM page faults, notifies remoteWakeup (described below), and issues page requests to memsrv running on the desktop and, on response, 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 re-schedules the vCPU. The migration avoidance code in the domU ensures that overwriting a page (page allocation), does not cause a notification to memtap.

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 backend, implementing the functionality of the device, runs in dom0. cownetdisk is our instantiation of the block device backend for consolidated VMs with desktop local storage. It is based on the blocktap interface [103] 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. 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 are whole block writes, which cause only a promotion. On a remote fetch, cownetdisk also notifies remoteWakeup first, to ensure that the desktop is awake.

remoteWakeup. A daemon responsible for waking up a sleeping desktop via Wake-On-LAN [71] 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 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 *dirty state tracking* mechanism. 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 are the page table pages. The descriptor is migrated to the server which creates a new domain and begins its execution.

On the desktop, a disksrv process and a memsrv process are instantiated and device backends 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 inter-domain 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 re-schedules 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, we only need to migrate back the new state. For this, we use the disk and memory dirty state tracking mechanisms described above. 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 local disk. Once all state has been updated, device backends are started and the VM is allowed to begin execution.

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 address 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 above 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 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 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 2 MiB at a time, while still replacing ghost MFNs one at a time.

While we have not implemented dynamic memory allocation and remote fault han-

dling 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, *maximum*. 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, unmaps and returns them to the cache. For our purposes, we will modify the PoD cache so it starts with a chunk-size 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 we use to fetch missing state on-demand. That is, when a fault occurs, first, we commit the backing page to the VM, and then notify memtap to fetch the content from the desktop.

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 needs 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?

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

Table 6.1: Power profile of Dell Studio XPS 7100 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 15 seconds. Once a VM was found to be inactive, an on-screen warning was displayed for 5 seconds before migrating the VM to the consolidation server, allowing a nearby user to cancel an impending consolidation, if needed.

Each VM was configured with 4 GiB of memory and 12 GiB of disk. The VMs ran Debian GNU/Linux 5.0 with kernel version 2.6.18.8 for x86_64, the GNOME desktop configured with Mozilla Firefox and Thunderbird, OpenOffice.org, Pidgin IM client, OpenSSH, 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, while 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.

The desktops were Dell Studio XPS 7100 systems, with a 3 GHz quad-core AMD Phenom™ II X4 945 processor and 6 GiB of RAM. Table 6.1 presents the desktop's

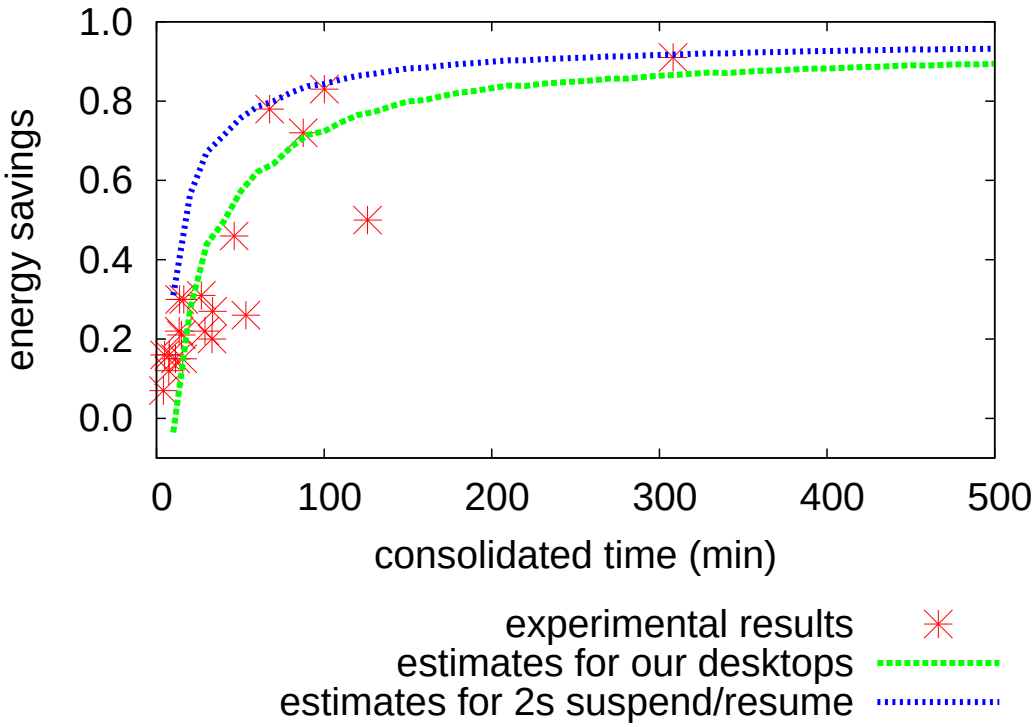


Figure 6.2: Desktop energy savings.

power profile obtained with a GW Instek GPM-8212 power meter. These numbers are comparable to those of other published systems [29, 47]. The server was a Sun Fire™ X2250 system with two quad-core 3 GHz Intel Xeon® CPUs and 16 GiB of memory. Its idle power averaged 150.70 W. 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 813.44 Mbps. Power use of the desktops during the deployment was measured with Watts up? PRO power meters.

6.2.2 Energy Savings

Figure 6.2 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. First, it shows finer grained estimates of expected energy savings for the desktops used in the deployment

over varying lengths of consolidation time. Second, similar estimates for a desktop with similar profile characteristic as those in the deployment, except for having faster suspend and resume times of 2 seconds rather than nearly 8.5 seconds of our systems. These estimates were computed from memory access traces, as described in Section 5.5. The estimates match our experimental data well.

The experimental results show that our users were able to see reductions on their desktop energy use from as short idle periods as 4 minutes. While in short idle times of under 10 minutes we see savings of 7% to 16%, in longer idle times the savings were significant. In idle times of 67 minutes, we see 78% savings and, in idle times of 308 minutes, we see even higher savings of 91%. We were unable to collect results for longer idle times because of a bug we encountered in the BIOS of our desktops. On occasion, after suspending and resuming an arbitrary number of times, the desktop would lock-up during power-on self-tests. In these instances, we had to power cycle the desktop.

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. The reason is that the desktops we use have very slow suspend and resume times (nearly 8.5 seconds). Still, we can see that, in sufficiently long idle times, where faults are extremely rare (not recorded), the energy savings converge towards $(P_i - P_s)/P_i = 96.82\%$.

We note here that recent laptops have demonstrated short resume times, such as approximately 2 seconds in both the MacBook Air [11] and Acer Aspire S3 [6]. We will show in Chapter 7 that with Context-Aware Selective Resume, which initializes only the necessary devices and system components on wake-up, desktops can achieve fast suspends and resumes that are comparable to those of optimized laptops. Further, in Chapter 8, we show an alternative, which uses a low power network accessible page cache to obviate the need for desktop wake-ups for page migrations, which yields substantial savings in short idle times. The estimates for a desktop with 2-second suspend and resume in Figure 6.2 shows that, faster transitions can lead to higher savings in short idle times. Intervals

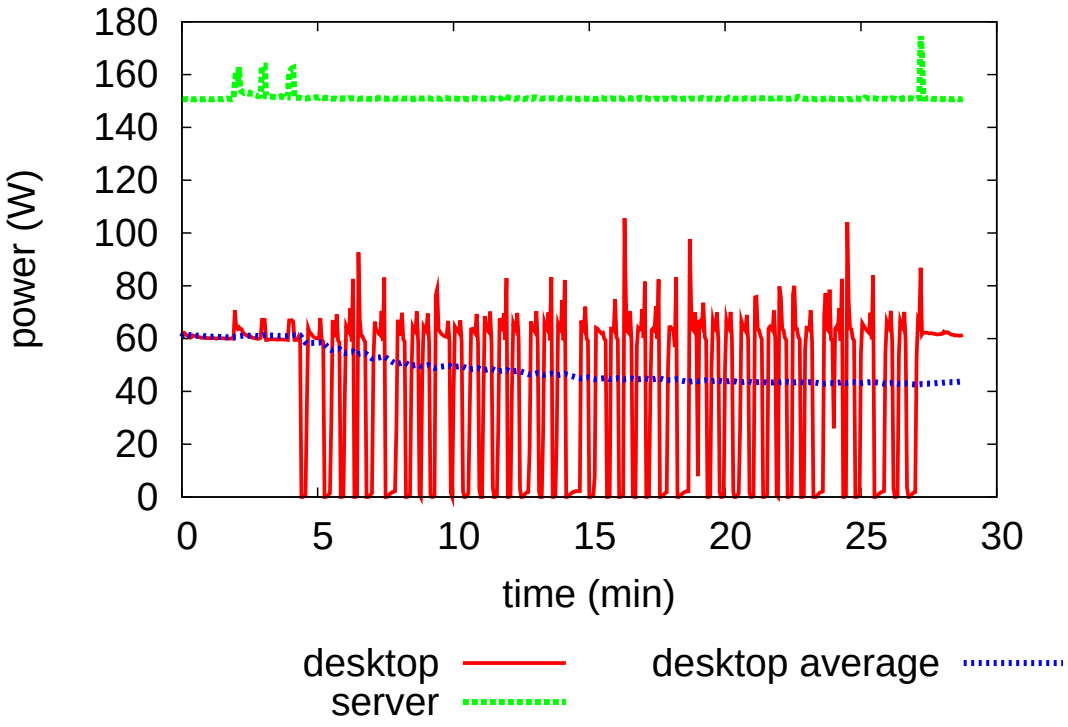


Figure 6.3: Power usage of a desktop and server during partial migration. The figure shows a reduction in the average energy use over time.

under 10 minutes would see savings that are closer to 30%.

Figure 6.3 shows detailed power usage of one desktop and the consolidation server over a 30 minute period in which the VM is consolidated to the server for 25 minutes. The figure shows the power usage patterns as the desktop performs microsleeps. At 1 minute and 57 seconds the VM begins migration to the server. This 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 4 minutes and 21 seconds, 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, 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 61.4 W to 43.8 W, a savings of 28.8%.

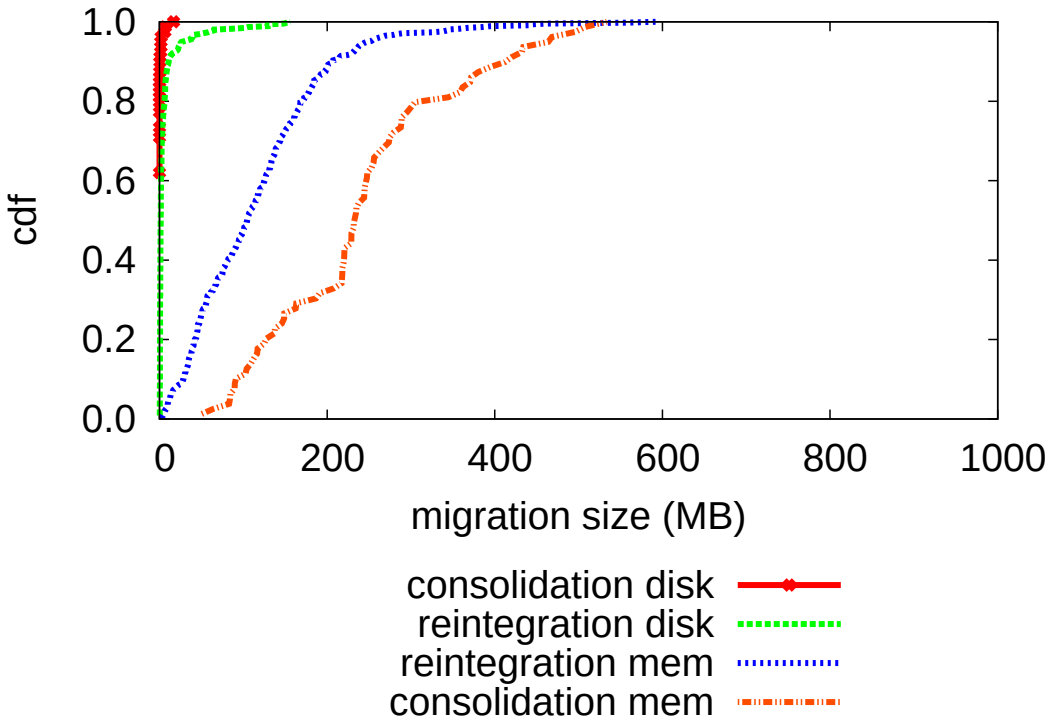


Figure 6.4: Distribution of migration sizes for VMs with 4 GiB of memory. Plot order matches legend top to bottom.

The energy savings of the desktop and the length of each microsleep increase over time.

While 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 it's power use per VM amounts to less than 2 W. This power can be driven further down by increasing only the memory capacity of the server.

6.2.3 Network Load

Figure 6.4 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 mean amount of memory migrated (including data migrated by on-demand prefetching) to the consolidation server was 242.23 MiB, 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 ± 25.94 MiB.

The average disk state migrated to the consolidation server was much smaller at 0.50 MiB. Similarly, on average, each VM migrates 114.68 MiB of memory and 6.81 MiB of disk state back to the desktop; 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 there, on user request.

Our experiments show that the time to migrate a VM back to the user's desktop is small. On average, users can expect their VMs to reintegrate in 4.11 seconds. Similarly, the average time to consolidate a VM is 3.78 seconds. These results exclude the desktop hardware suspend and resume times, found in Table 6.1.

6.2.5 Summary

Our evaluation shows that partial migration is able to achieve significant energy savings while generating only minimal loads on the network and providing low migration latencies to the user. These benefits are made possible by migrating barely more than the working set of idle VMs, and by taking advantage of microsleeps, short sleep times in which the desktop's attention is not required.

6.3 Scalability and Comparison with Full VM Migration

Next, 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. We address the following questions: (i) How does partial migration compare against full migration in terms of network usage, overall energy savings, and the desktop reintegration latency experienced by users? (ii) Do the techniques scale with the number of desktops? (iii) Can they weather "resume storms" present in actual usage patterns? and most importantly, (iv) Do the energy savings exceed the capital costs required to deploy each technique?

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 5 seconds. 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 screensaver 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 2086 person day traces from which a sample of 500 are shown in Figure 2.1. Of the full traces, 1542 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 does 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. In the interest of space, we report only on weekday results unless otherwise stated. The simulations assume a shared GigE network, desktop VMs with 4 GiB of RAM, and the same energy profile as the desktops used in our experiments (Table 6.1). 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 pre-copy rounds or disk accesses, and assuming exactly a 4 GiB 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 6.4. 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 6.2 that estimates the savings as a function of consolidation time for our desktops.

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

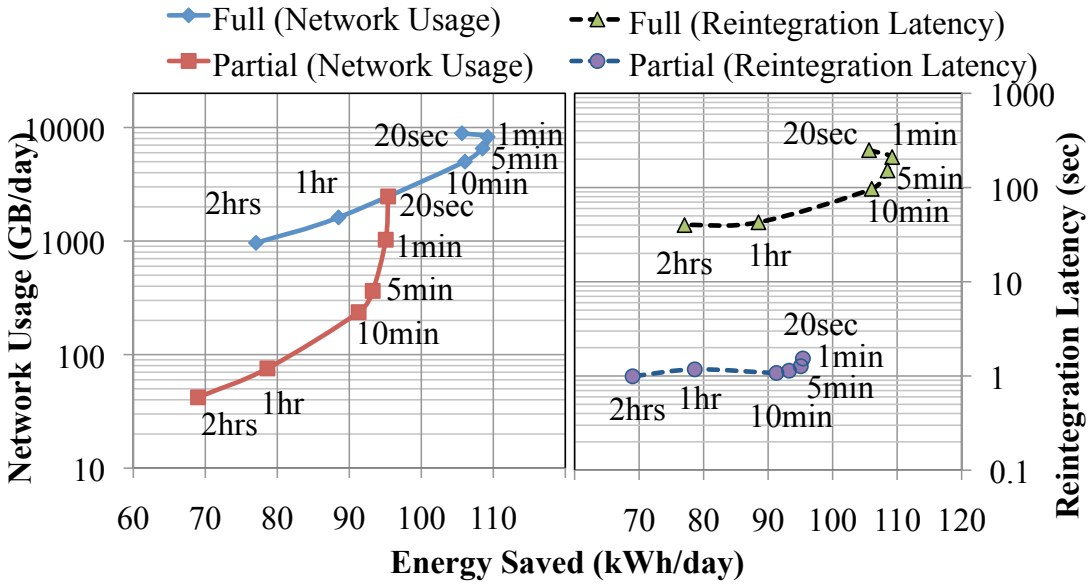


Figure 6.5: Energy savings in kW-h per day vs. reintegration latency and network utilization for 100 desktops and varying idle timeout values.

the idle timeout to migrate less aggressively, thus reducing network load and improving reintegration latencies, but reducing sleep opportunities and energy savings. Figure 6.5 shows that the answer to this question is an emphatic no. It shows a scatter-plot 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 5-10 min for the idle timeout. Higher values significantly reduce energy savings, while lower values dramatically increase network load and reintegration latency without increasing energy savings much. For full migration, energy savings actually reduce for small idle timeouts because the aggressive migrations entailed lead to a lot of energy wasted in aborted migrations and oscillations between the desktop and consolidation

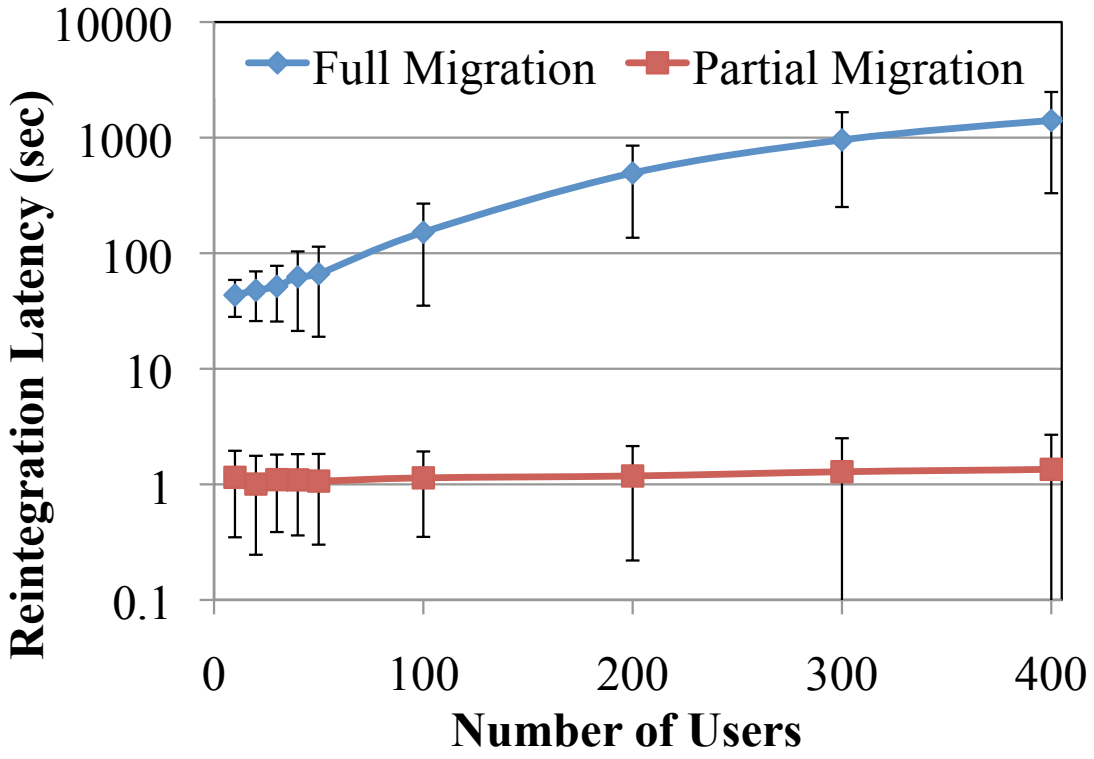


Figure 6.6: Reintegration latency of partial migration and full migration as desktops increase. The error bars show the standard deviations.

server. Similar graphs for 10 to 500 desktops show that an idle timeout between 5 and 10 min 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 5 min for these experiments.

Figure 6.6 shows an over 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 151 s reintegration latency of full migration could be intolerable for users. Das et al. [47] propose using a remote desktop solution to provide immediate reintegration access to users to mask long reintegration latencies of full

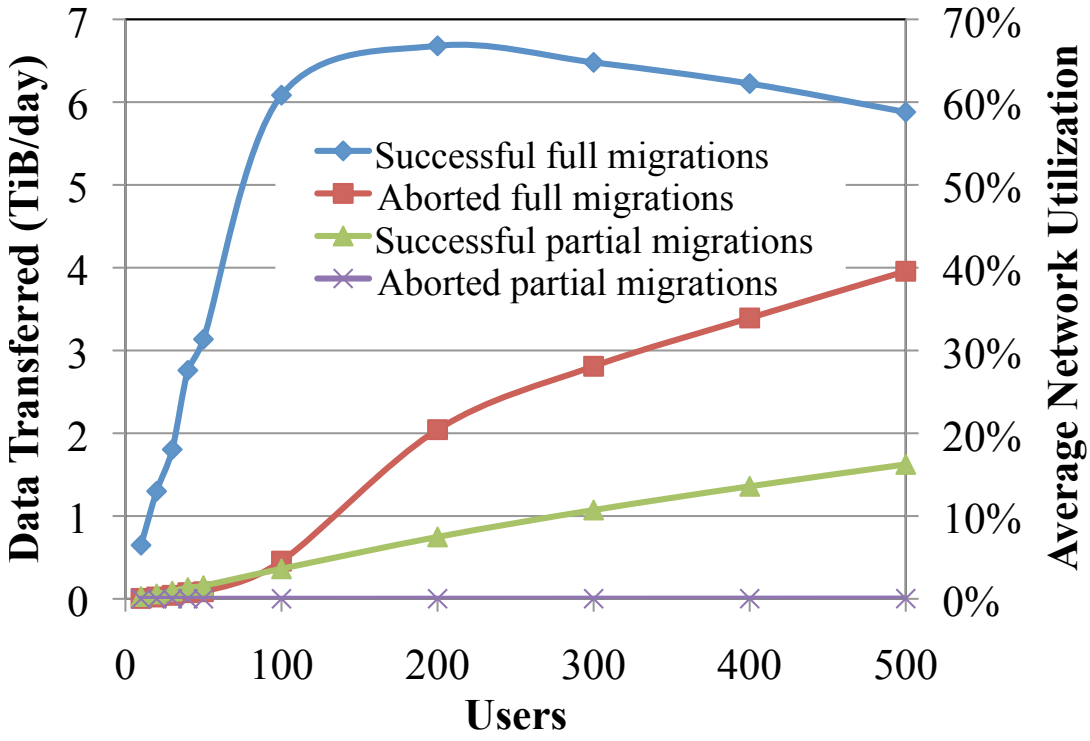


Figure 6.7: Network load of partial migration and full migration. Aborted migrations occur when a new migrations overrides an ongoing one.

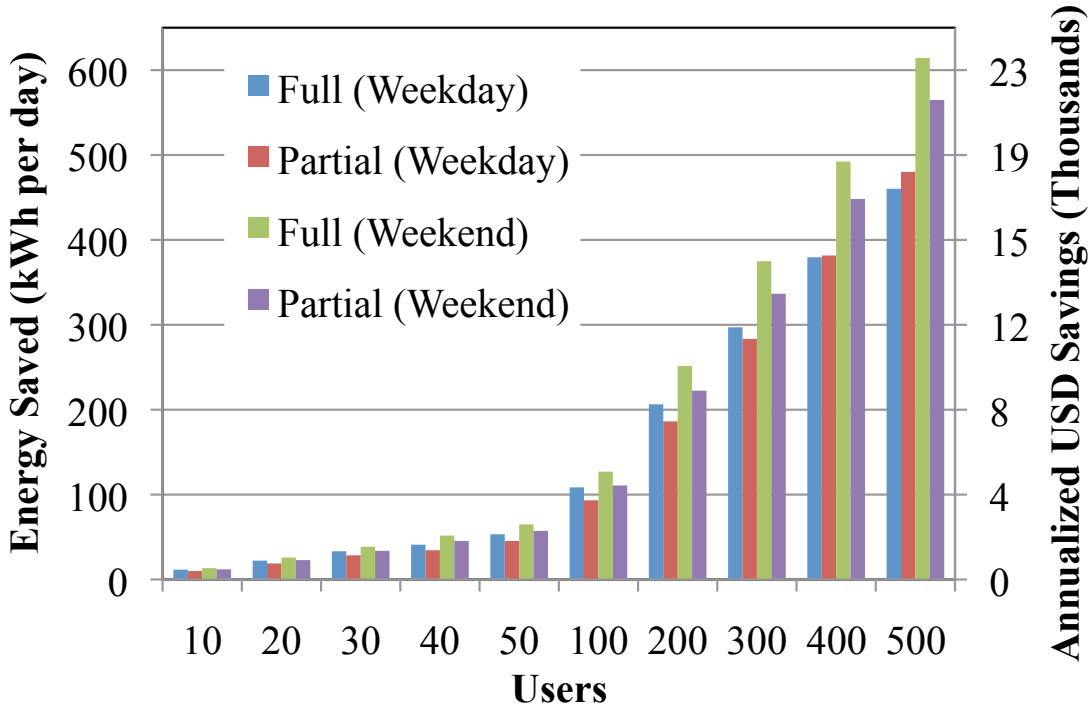


Figure 6.8: Total energy savings in kWh and USD by partial migration and full migration.

migration. However, remote desktop access has many limitations, such as the 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 6.7 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 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 6.8 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, 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 idle timeout of 5 minutes, 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:

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

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 16 GiB of memory, similar to our testbed system. Since fully migrated idle VMs are memory constrained on the server side, we assume 4 4 GiB VMs on a single server giving us a break-even 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 16 GiB 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 6099.[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.

6.4 Challenges with Partial VM Migration

While 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, causing frequent transitions between power states on the PC.

(i) Limited energy savings during short periods. The first challenge, is that state transitions reduce total sleep time, and therefore energy savings of the PC. As discussed in Section 5.4, the PC tries to reduce the number of power state transitions by servicing consecutive page requests at a time, before returning to low power. Even so, results from our deployment of Jettison, presented in Section 6.2, found that during short idle times, where faults are frequent, the energy savings are low. For example, while Jettison achieved savings of up 78% in idle periods lasting an hour, in idle periods of about 30

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

minutes, savings ranged from 20% to 31%. In idle periods under 10 minutes, 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 15 minutes of user inactivity, and transition into low power mode within 30 minutes [16]. 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 20 minutes, and turn off the computer when away for at least 2 hours [15]. The default power management plan for Windows 7 PCs (the balanced plan) turns the display off after 10 minutes of inactivity, and enters low power mode after 30 minutes. For the Mac OS X 10.6, these defaults are both of 10 minutes. These policies make a distinction between power management modes that allow applications to run (turning off the monitor) and those that don't (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 20 minutes. We must, therefore, ensure that partial VM migration can deliver savings, even during idle periods of ten minutes. 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 that are 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 wake up times of PCs reduce responsiveness of applications 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 to 30 minutes, which has proven far more than sufficient to deal with the 8 to 17 seconds remote fault latency, in the worst case, in our usage of the system.

In the next Chapters, we present two alternative solutions that address these problems caused by on-demand memory migration. The first, Context-Aware Selective Resume, 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 a wake up context to the PC. It has the dual benefits that wake-ups are fast, increasing energy savings, and most hardware components are not re-initialized for on-demand page migration.

The second solution, deploys a low power page cache to enable PCs in low power mode to offload VM memory state to network accessible hardware caches, ensuring that

outside hosts have access to the PC's state even when the latter is network inaccessible. The page cache, ensures that the PC is never woken to serve pages on-demand.

6.5 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 (ii), what happens in case of failures.

Storage Placement. As described in Chapter 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 Chapter 5 more than 99% of state accessed by idle VMs is memory (165.63 MiB), while disk represents less than 1% (1.16 MiB). An environment with shared network storage reduces the number of faults that must be serviced from the desktop, and potentially increase the energy savings with partial VM migration, though 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 migration of VMs for two reasons. First, some devices found on the desktops may not be available on the consolidation server. And 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 device passthrough [70, 111, 27, 63]. 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 [67, 113], and some newer devices provide support for Single-Root I/O Virtualization and Sharing (SR-IOV) [64, 50], which virtualizes the device itself, allowing context switching between VMs, and more importantly, 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, while 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 3D acceleration interface may require the VM to be migrated back to the desktop. These policies may be implemented in the hypervisor on a per device basis. Moreover, a given device may support different access modalities, and a policy may be in place to decide. For example, disk accesses may be forwarded to the desktop when infrequent. However, frequent accesses, caused, for example by a running virus scanner, could cause the VM to migrate to the desktop, as they do not allow the desktop to sleep. Our prototype implementation supports paravirtualized devices, and forwards only accesses to block devices to the desktop. Other device accesses are handled on the server. We leave a thorough implementation of device access policies and support for device passthrough for future work.

Failure Semantics. The semantics of failure of a consolidated VM on storage consistency is similar under the local storage or the network storage configurations, and is based on checkpointing. When a VM is consolidated, memory and disk state is 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 that is 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 which do not corrupt the host's file system and from which the host can recover, for example by rebooting, dirty disk state may still be recovered. Presently, 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.

6.6 Summary

In this chapter, 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 4 s. With traces of PC user activity, we showed that in small to medium 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. Finally,

we discussed the challenges of the approach and considerations that implementers shall make when deploying partial migration of idle desktop VMs in enterprise environments.

Chapter 7

Efficient Migration of Pages On-Demand with CAESAR

7.1 Introduction

Migration of pages on-demand enables partial VM migration to limit network traffic and deliver high consolidation ratios on the server. At consolidation, partial VM migration 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 up 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 to full power and back to low power mode dozens of times per hour (see Section 4.3).

Our experience with Jettison, our partial VM migration prototype, has shown that energy saving ACPI power states have not been designed for frequent transitions. While the ACPI standard [59] 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 (Sec. 2.2), and result in high rate of power use (Sec. 5.4).

Our observation is that partial VM migration forms a new class of desktop applications that require fast transition times, coupled with substantial energy savings. This new class consists of applications that require intermittent services from the PC, while maintaining low energy footprint. Aside from partial VM migration, members of this class include remote sensing applications, such as remote aggregation of temperature and hardware inventory tracking; applications that maintain presence with low frequency, such Dynamic Host Configuration Protocol (DHCP) lease renewal clients; and simple network responders, such as Address Resolution Protocol (ARP) probe and network plug and play service discovery responders. Partial VM migration may also take advantage of fast resume times to forward UI activity generated in the consolidated VM to the desktop PC, which can enable delivery of notifications to users who may be nearby. For example, when a call arrives at the VoIP client, the ringtone is heard on the user's desktop. If the user is nearby he can choose to take the call, causing the VM to migrate to the desktop. The common characteristic of these applications is that they require local resources of the PC *intermittently*.

This class of applications has motivated a study by Wright et al. [110] on the causes of the long resume-suspend cycles of commodity PCs. Wright et al. find that 87% of the cycle times, the time taken by PCs to resume from low power state, send a network packet, and return to low power state, is taken by OS and BIOS activity, and only 13% comprises hardware suspend and re-initialization. When suspending into S3 low power state, the OS iterates through devices and saves their state in DRAM. When resuming from low power, 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 pre-suspend state. The observation that Wright et al. make is that applications providing intermittent services often require functionality from only a

small fraction of 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 the OS subsystems and devices required to provide the service. Wright et al. propose *Context-Aware Selective Resume* [109], an approach that selectively suspends and resumes devices and OS code paths based on the wake-up context. The wake-up context informs the PC of the service causing the wake-up. For example, for partial VM migration, this context would inform the PC on wake-up that only the memory server functionality is needed. As a result, the PC needs only to re-initialize DRAM, CPU and the network interface.

In this chapter, we evaluate the benefits of Context-Aware Selective Resume in reducing on-demand page migration cost, the energy cost of forcing the PC out of low power mode. High cost of on-demand page migration makes consolidations over short intervals, when faults are frequent, only mildly successful in reducing desktop energy use. We experiment with a memory server implementation built on CAESAR, the Context-Aware Selective Resume framework. 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 9.0 seconds to 3.2 seconds, and increase desktop sleep time over an hour-long consolidation by 26% to 103%. Context-Aware Selective Resume increases energy savings over the full resume approach throughout the hour-long interval. For consolidations lasting five minutes, it increased energy by 13% to 66%. In ten minutes, the increase was by 43% to 54%. In twenty minutes, it increased savings by 38% to 49%, and finally over an hour, the approach increased energy savings by 14% to 65%.

The remainder of this chapter is organized as follows. In Section 7.2, we motivate the need for fast resume-suspend cycles with results from our deployment of Jettison. In Sec-

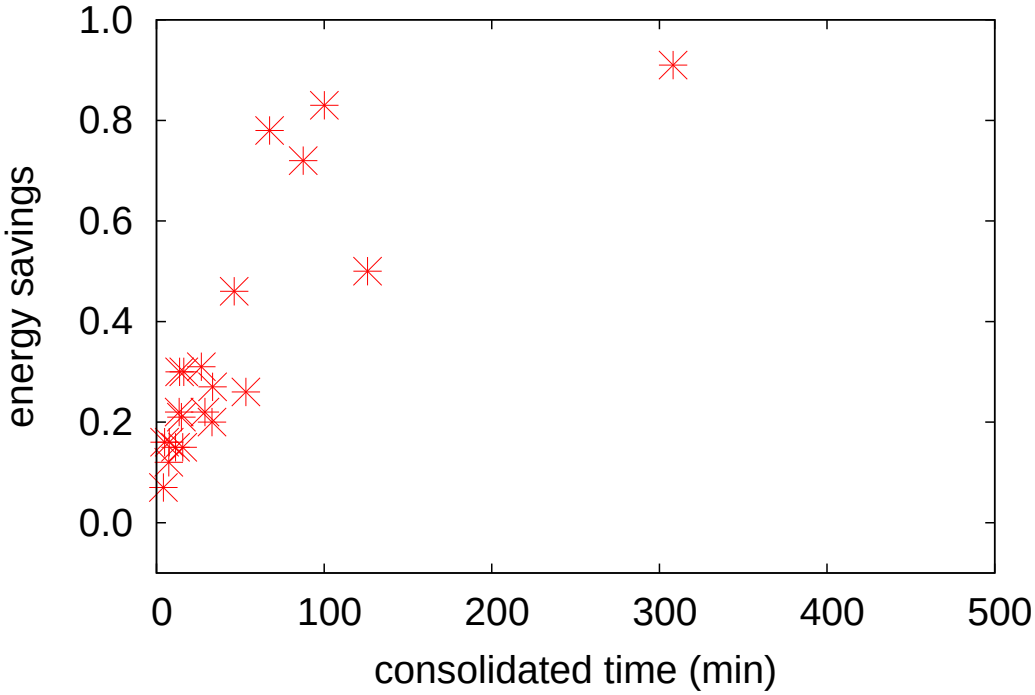


Figure 7.1: Energy savings of desktops with Jettison (reproduced from Section 6.2).

tion 7.3, we discuss the architecture of the Context-Aware Selective Resume framework, and present our implementation of a fast memory server. In Section 7.4, we present experimental results demonstrating the benefits of Context-Aware Selective Resume when migrating memory on-demand. Finally, in Section 7.5, we conclude the chapter.

7.2 Motivation

We have shown in Section 6.2 that partial desktop VM migration is an effective approach to reduce energy use of idle desktop PCs. Figure 7.1, which reports the energy savings achieved for users participating in our deployment of Jettison from Section 6.2, shows that when users are idle for at least an hour, Jettison is able to save up to 78% of the PC's idle state energy use. When users are idle for five hours, it is able to save up to 91% of the energy. However, in shorter idle times these savings were modest. Users that

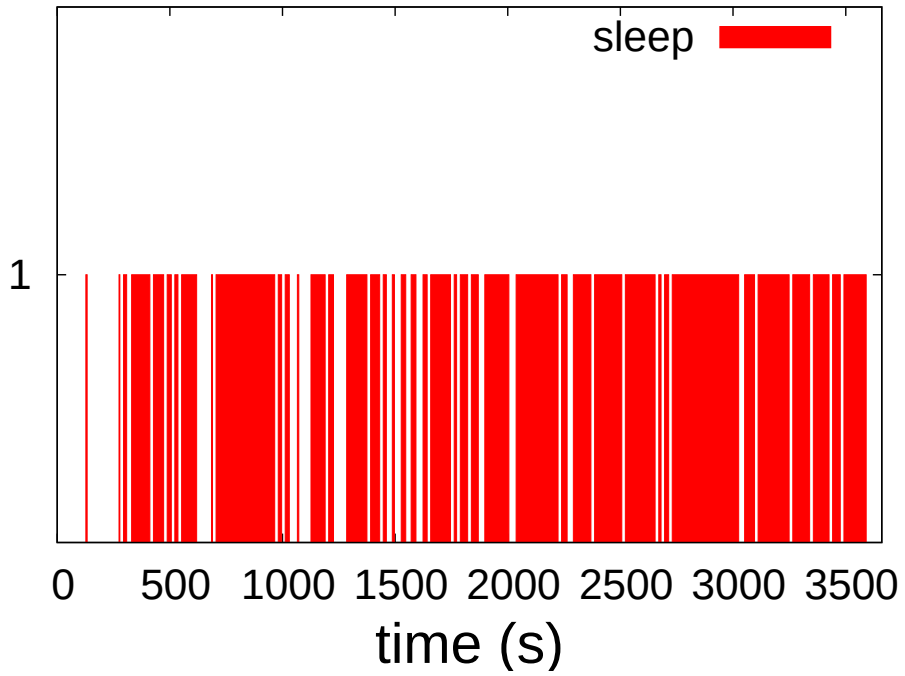


Figure 7.2: Potential desktop sleep intervals for IM (reproduced from Section 4.3).

were idle for about 20 minutes saw savings of 15% to 30%. Users that were idle for less than 10 minutes saw savings of 7% to 15%.

Why are savings modest in short idle times? Two factors help explain the low savings in short idle times. First, as we have shown in Section 4.2, the majority of page migrations occur early in the consolidation. Indeed, for the four idle desktop workloads we studied in that section, namely, the desktop login screen, the e-mail client, the instant messenger, and the multitasking desktop, the bulk of page migrations occur within the first 22 minutes. Thereafter, page faults become sporadic. Figure 7.2 shows sleep opportunities, the moments in which no faults occur and desktop may be able to sleep, for the instant messenger workload. The figure clearly shows that, as faults occur with high frequency in the first minutes, the desktop must wake up often.

The second factor contributing to low energy savings in short idle times is long desktop power state transition times. Even in the presence of sleep opportunities, desktops with

long transition times are unable to take advantage of them. Instead, they squander these opportunities cycling between states. Recall from Section 6.2, that the Dell Studio XPS 7100 desktops have combined cycle times (resume and suspend) of 17.6 seconds. Worse, because during state transitions power use surges, desktops can only save power if microsleep durations are long enough to offset the transient cost. We have shown in Section 5.4 that these desktops only save power during sleep opportunities of at least 32 seconds.

We alleviate the first challenge, namely frequent transitions, with pre-fetching and the use of a microsleep timer to serve multiple page requests during a power up. In this chapter, we turn our attention to the second challenge, namely long resume-suspend cycles. It is imperative that we reduce the energy cost of serving pages to ensure considerable energy savings even when faults occur with high frequency, as is the case in the initial phase of consolidation, and we use Context-Aware Selective Resume to do just that. Context-Aware Selective Resume, shortens transition times by initializing only devices and software subsystems needed to serve pages during the resume procedure. During the suspend procedure, only state of devices that were initialized during the previous resume process are stored. In the following sections, we demonstrate how Context-Aware Selective Resume improves energy savings in short idle times.

7.3 The Context-Aware Selective Resume Framework

Context-Aware Selective Resume provides a framework designed to expedite resume-suspend cycle times of desktop PCs. When an event causes the PC to wake up from 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 that is 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, the 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, Wright et al. [110] report BIOS initialization to take under 1 second, 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 sixteen 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 indicates that this is a context-aware selective resume packet. And 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 page 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 to on resume. It is in this function where 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 resume function of the application 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, 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 which 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 re-initialization of the NIC, CAESAR locks the NIC's link speed at suspend time, which avoids re-negotiation during resume. Normally, when the desktop enters low power mode, the Intel NIC reduces its link speed from 1 Gbps to 10 Mbps, which uses modestly less energy. At resume time, the NIC re-negotiates the link speed with the switch at the opposite end of the wire, in a process known as ethernet autonegotiation. We found this re-negotiation causes a delay in NIC initialization of up to 4 to 6 seconds. CAESAR prevents link speed re-negotiation by setting NIC registers which disable re-setting 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 page frame number (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 machine frame number (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.

7.4 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 with Context-Aware Selective Resume, particularly during short idle times?

7.4.1 Methodology

To evaluate the benefits of CAESAR, we collected traces of page requests during the deployment of Jettison described in Section 5.1. These traces contain time stamped sequence of page numbers (PFNs) requested each time VMs were consolidated. Recall that in our deployment of Jettison, three university researchers were given desktops running

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 7.1: Power profile of our custom PC.

Jettison and employed these as their primary desktops. Each desktop ran a VM configured with 4 GiB of memory and 12 GiB 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 5.4, that we use a wait timer (t_w) that keeps the desktop awake for a pre-determined 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 six seconds. 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 two seconds was sufficient to ensure that minimal energy is wasted in transitions for microsleeps that are too short to save energy.

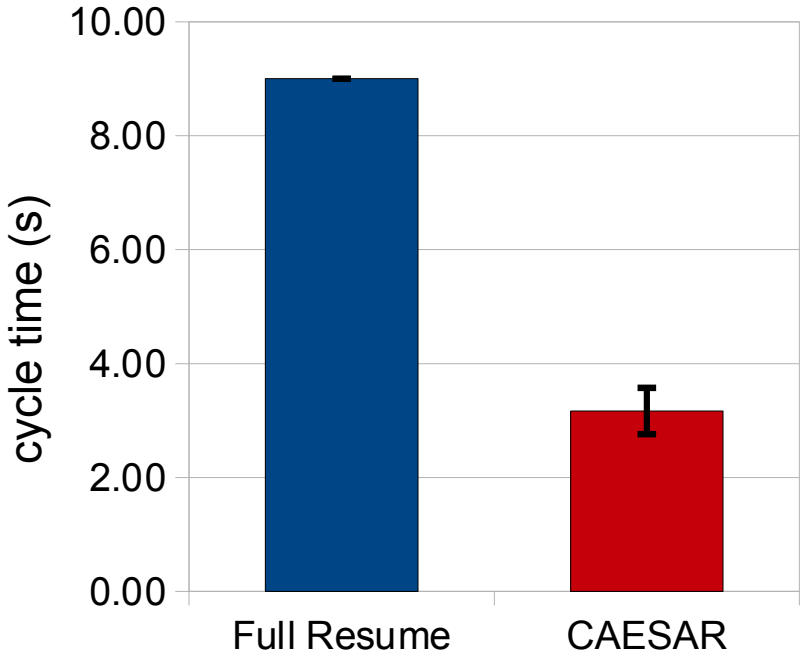


Figure 7.3: Resume-suspend cycle times.

For these experiments, we used a custom desktop PC with a 2.70 GHz Dual-Core Intel Celeron® CPU, and the power profile described in Table 7.1. 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 traces of all three users over a period of one hour.

7.4.2 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 one minute. 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.

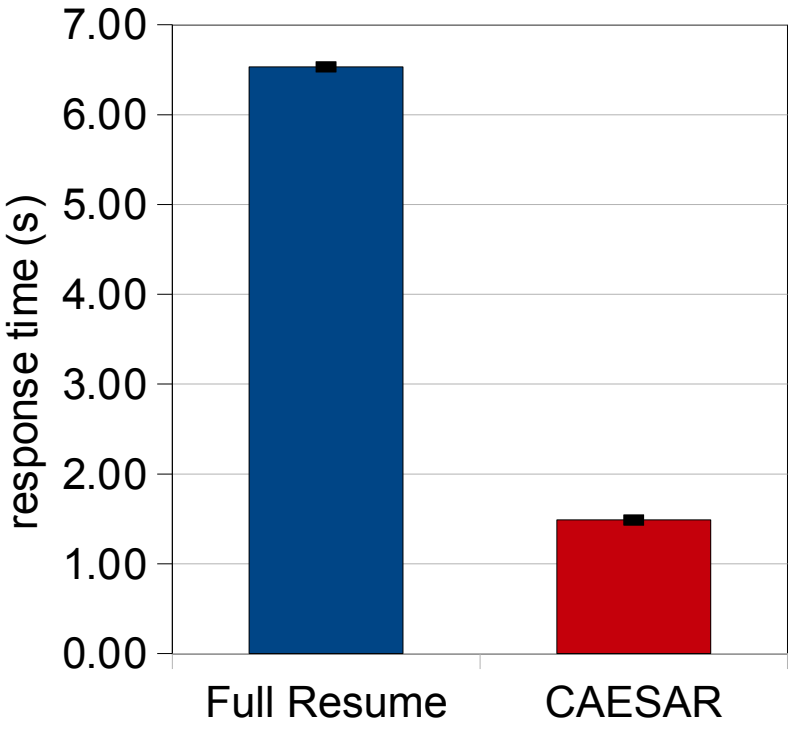


Figure 7.4: Page response times.

Figure 7.3 shows the cycle times of the desktop using either full resume or Context-Aware Selective Resume. With full resume, the desktop spends, on average, 9.0 seconds out of low power to serve a single page. CAESAR reduces cycle times by 64.78% to 3.17 seconds. The standard deviation in both cases was under 0.42. 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 6.53 seconds to 1.49 seconds. Figure 7.4 shows the response times measured on the consolidation server. Short response times are useful in ensuring that applications running in the consolidation server run uninterrupted.

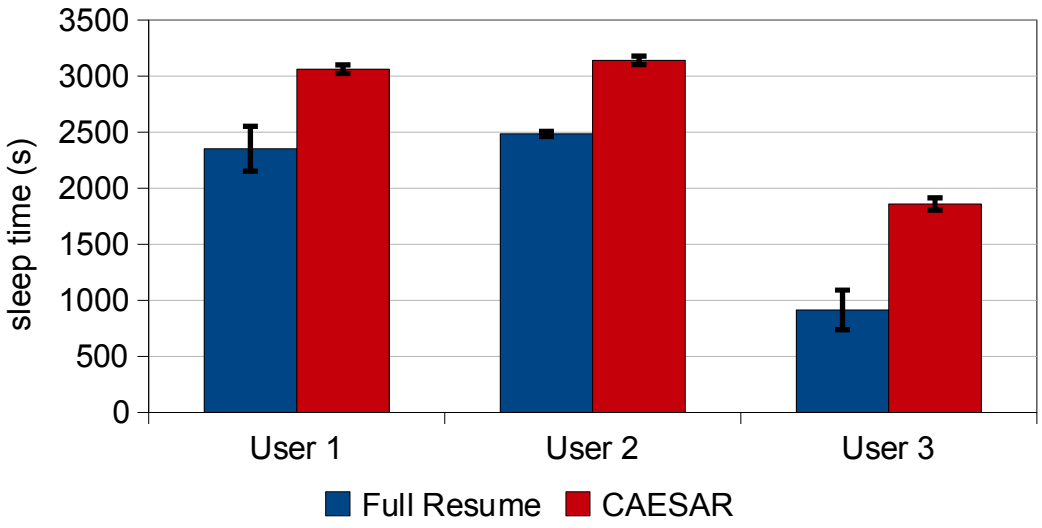


Figure 7.5: Desktop sleep times over hour-long consolidations.

7.4.3 Sleep Times

Shorter cycle times increase total sleep the desktop can perform during consolidation. Sleep time refers the time in which the desktop is in low power mode while its VM is consolidated. Figure 7.5 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 one second in which the desktop used less than 8 W of power. The figure shows that, for User 1, CAESAR increases total sleep time from 39.19 minutes to 51.01 minutes. For User 2, it increases total sleep time from 41.41 minutes to 52.32 minutes. And finally, for User 3, CAESAR increases total sleep time from 15.24 minutes to 30.98 minutes. Across users, CAESAR increases sleep times by 26% to 103%, during the hour-long period of consolidation.

7.4.4 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.

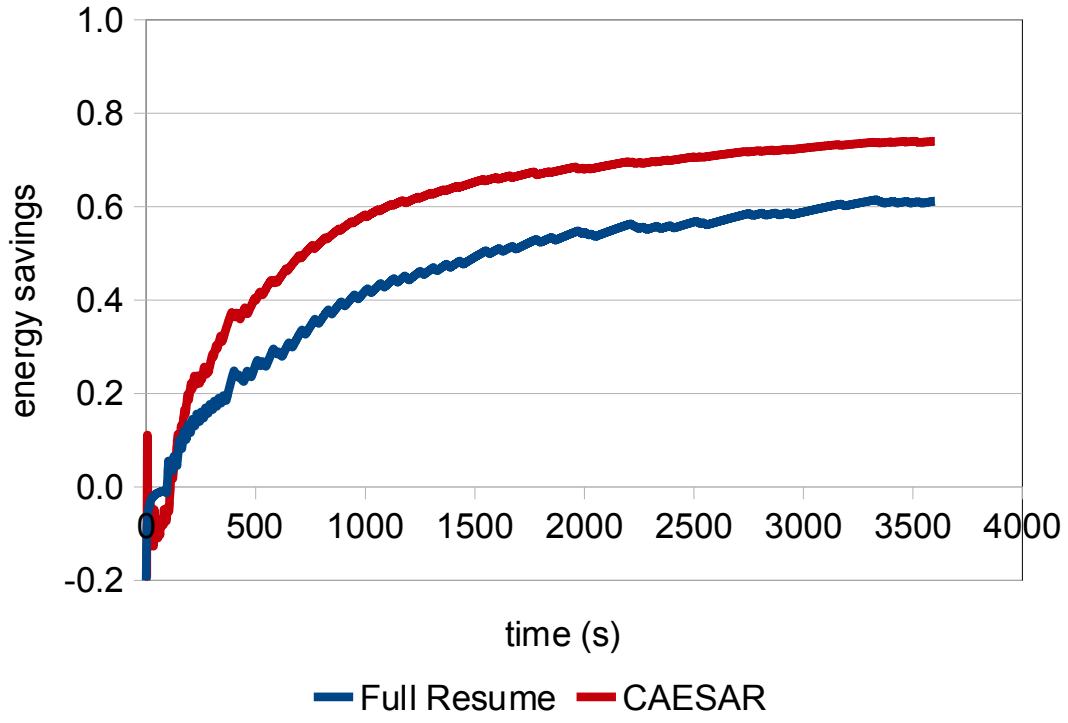


Figure 7.6: User 1 desktop energy savings.

Figures 7.6, 7.7, and 7.8 compare energy savings when desktops resume fully to serve pages, with savings obtained CAESAR invokes only the memory server. The figures show the energy savings of the desktop when its VM is consolidated over a period of one hour.

Figure 7.6, shows that for User 1, CAESAR increased energy savings from 17% to 28% in the first five minutes. At ten minutes, the savings increased from 28% to 44%. At twenty minutes, CAESAR increased the savings from 44% to 61%, and finally, at the end of the hour, CAESAR increased from 61% to 74%. Initially, during the first minute, CAESAR leads to a loss of energy as our implementation first transitions the PC into low power, incurring 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.

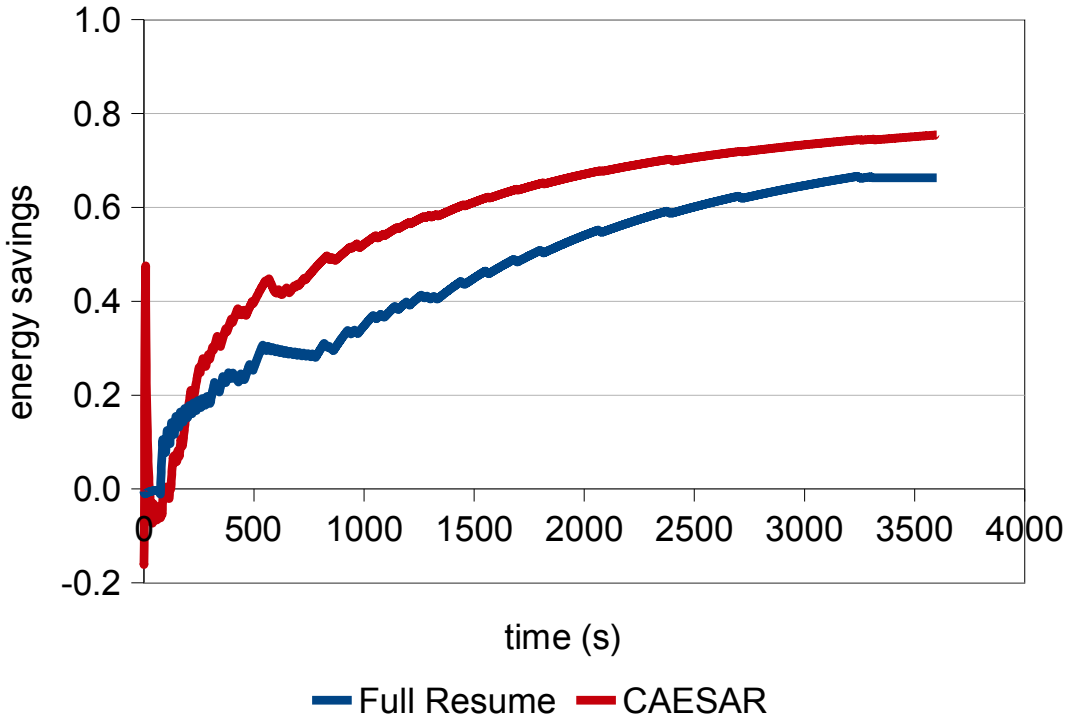


Figure 7.7: User 2 desktop energy savings.

Figure 7.7, shows that for User 2, CAESAR increased energy savings from 18% to 28% in the first five minutes. At ten minutes, the savings increased from 29% to 42%. At twenty minutes, CAESAR increased the savings from 39% to 57%, and finally at the end of the hour, CAESAR increased from 66% to 75%.

Figure 7.8, shows that for User 3, CAESAR increased energy savings from 14% to 16% in five minutes. In ten minutes it increased savings from 17% to 24%. In twenty minutes 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, while savings for User 3 are lower across the consolidation, CAESAR is still able to improve savings significantly.

Across users, CAESAR delivers improvements of 13% to 66% over five minutes, 43% to 54% in ten minutes, 38% to 49% in twenty minutes, and 14% to 65% in an hour.

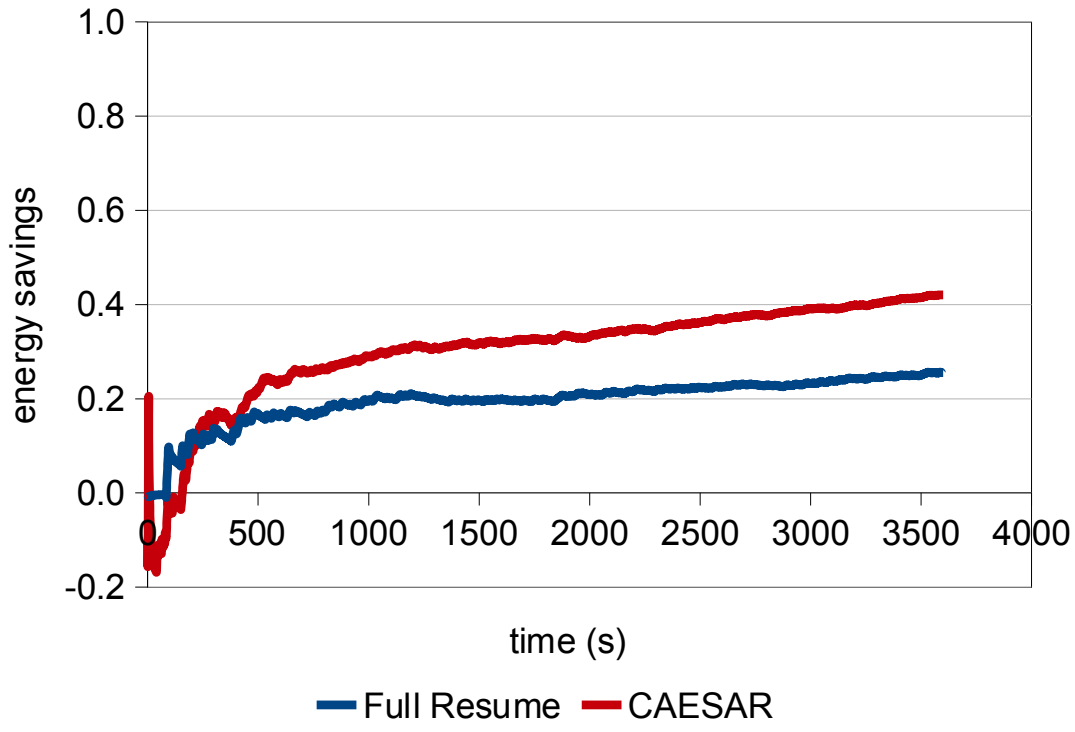


Figure 7.8: User 3 desktop energy savings.

7.5 Summary

In this chapter, we presented Context Aware Selective Resume, an approach that reduces desktop resume suspend cycles. Current desktops have slow power state transition times with high rate of power use. We showed that 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, which informs 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 CAE-

SAR reduces resume-suspend cycle times by 67% from 9 seconds to 3.17 seconds. It increases total desktop sleep time by 26% to 103% over an hour-long consolidation. Most importantly, our experiments have shown that CAESAR increases energy savings, over short idle times between 5 minutes and an hour by up to 66%.

Chapter 8

Low Power On-Demand Page Migration with Oasis

8.1 Introduction

Migrating pages on-demand, as performed by partial VM migration, wakes the PC whenever a fault occurs on the partial VM. Frequent transitions to and from low power mode are undesirable because, first, they reduce total sleep time, and second, they cause a surge in power use. Both outcomes result in reduced energy savings. As discussed in Section 5.4, the PC tries to reduce the number of power state transitions by servicing consecutive page requests at a time, before returning to low power. Even so, results from our deployment of Jettison, presented in Section 6.2, found that during short idle times, where faults are frequent, the energy savings are low. For example, while Jettison achieved savings of up 78% in idle periods lasting an hour, in idle periods of about 20 minutes, savings ranged from 15% to 30%. In idle periods of about 10 minutes, energy savings ranged from 7% to 15%. Traces of desktop user activity collected in a research lab, and presented in Section 2.1, indicate that short idle periods constitute a significant share of total idle times: 17% of all idle time (including overnight hours) is made up

of idle periods lasting less than an hour. This indicates a failure of partial VM migration to address significant opportunities to save energy. Compounding the challenges of using partial VM migration in short idle times, is the concern that the frequent power state transitions can lead to the premature demise of hardware components that are not designed for frequent power cycles.

In this chapter, we introduce a low power page cache, an approach that reduces the number of PC power state transitions, by offloading the task of serving pages on-demand to a network accessible cache. Before entering low power mode, the PC uploads the memory pages of the idle VM to its local page cache, and when a fault occurs on the consolidation server, the server fetches the page from the cache. This allows the PC to remain undisturbed in low power.

We envision such cache to be an integral part of PC hardware. Such hardware may be embedded in the network interface card (NIC) of the PC, or may be implemented by a general purpose microprocessor on the PC's motherboard. Modern network interfaces are endowed with general purpose controller hardware capable of performing high throughput I/O with little assistance from the CPU. These controllers also enable standalone functionality, such as Wake-on-LAN, the ability for the NIC of a PC in low power mode to inspect arriving network packets and wake-up the main system on receipt of a designated packet.

This approach differs from hierarchical power management architectures, such as Turducken [94], Somniloquy [29], and Reflex [72], where user application functionality migrates to and executes on the low power component. In contrast, desktop applications do not execute on the low power cache, only a general purpose page server does. As a result, the low power page cache requires no application modifications or protocol aware proxies. The low power device is also simpler and requires minimal working memory: the device only reads opaque pages and transmits them over the network.

We present Oasis, a low power page cache prototype that runs on the Gumstix Overo

platform [17], an off-the-shelf computer-on-module that is endowed with a low power processor and local storage. Our experiences with the Gumstix platform find it to be hampered by low bandwidth interfaces to the PC, in the form of an 80 Mbps USB interface and a 100 Mbps ethernet interface. Such bandwidth results in long VM memory upload times, and reduces the desktop's ability to enter low power mode. The desktop is unable to enter low power mode until VM memory upload to the cache completes. We expect bandwidth and capacity of the storage medium to remain a challenge even with NIC-based implementations of the cache. We use several strategies to cope with these challenges. First, we pool together the bandwidth of multiple transmission interfaces and storage media to speed up uploads. We do so by partitioning the VM's memory image into multiple chunks, and transmitting these chunks over separate interfaces to multiple storage in the cache. Second, before uploading chunks to the cache, we compress pages to make efficient use of the bandwidth and storage available. Third, we persist memory pages in the cache, and during each upload, transmit only deltas consisting of pages that have been updated since the previous upload. And fourth, we use efficient garbage collection to address limitations of storage capacity. The enabling technology is the use of a page table data structure, which enables fast lookup of pages over multiple chunks, and query of page staleness.

Our experiments show that, with a multi-tasking workload, Oasis is able to produce energy savings in excess of 34% over consolidations as short as five minutes, where Jettison only produces savings of 2%. After thirty minutes, Oasis reduces energy use by 74%–81% compared to 12%–23% under Jettison, and after an hour, reduces by 83%–88% compared to 26%–28%. When uploading VM memory images from the desktop to the cache, page compression reduces transfer sizes by 93%, and selective upload of deltas by another 57%. Pooling of USB and ethernet interfaces reduces upload times for a 4 GiB VM by at least 29% from 25 to 50 minutes for a single interface to about 18 minutes. Compression and selective delta uploads reduce upload times further to 91 seconds and 30 seconds,

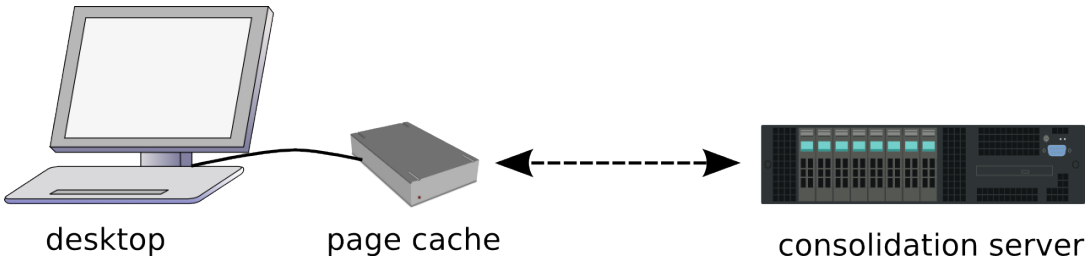


Figure 8.1: Architecture of Low Power Page Cache.

respectively, making Oasis practical for short idle times. Our experiments also show that Oasis ensures low page fault response times, as the consolidation server need not wait for the desktop to resume from sleep mode, to access missing pages.

The remainder of this chapter is organized as follows. In Section 8.2, we discuss the use of a low power page cache to improve the efficacy of partial VM migration. In Section 8.3, we describe the approach we take to implement Oasis, our low power page cache prototype, with off-the-shelf hardware. In Section 8.4, we discuss our evaluation, and show that Oasis achieves substantial energy savings, far above what is possible with Jettison, and makes consolidations that are as short as five minutes practical. In Section 8.5, we explore strategies to deploy page caches with commodity PC hardware. And finally, in Section 8.6, we conclude the chapter.

8.2 The Low Power Cache Architecture

A page cache is a low power, network accessible, cache of VM memory that is embedded in the hardware of the desktop PC. It responds to requests for memory pages made by the consolidation server, even as the PC is in low power. The page cache extends the functionality of ACPI low power states to enable the PC to provide limited functionality in states that otherwise support no execution, and without impacting the power use of the low power state significantly. Recall from Section 2.2 that modern PCs support at least two low power states: S3 or suspend-to-memory and S4 or suspend-to-disk. In both

states, the CPU and devices are powered off. In S3 state, DRAM receives self-refresh power in order to retain execution state, including CPU context that has been flushed to DRAM. In S4 state, all execution state is flushed to disk or persistent storage, allowing all components to be powered off. Because the CPU is off, all PC execution is halted when in low power state. With a page cache, a low power microprocessor is embedded in the hardware of the PC, and when entering low power, the main processor transfers DRAM state to the low power co-processor.

Figure 8.1, shows an overview of the low power cache architecture. Conceptually, the page cache architecture consists of three entities: the desktop PC, the cache, and the consolidation server. The cache is designed to support low power consolidation of idle partial VMs. The desktop and consolidation server have hypervisors that run VMs. The cache runs a memory server that is able to send VM pages from the desktop to the consolidation server.

Users run their VMs on the desktops. When a user becomes inactive and his VM is idle, the desktop initiates partial migration of its VM to the consolidation server, so that it can enter low power S3 state to save energy. It initializes the cache and grants it access to the VM's memory pages. The cache invokes its memory server and begins accepting network requests. The desktop halts local execution of the VM and migrates a partial VM replica to the consolidation server, notifying the server of the cache's IP and listening port. The desktop then transitions into S3. As the partial VM executes on the consolidation server, it faults on accesses to missing pages. The consolidation server, halts the faulting vCPU and requests the pages, by sending the page frame numbers to the cache's memory server. The memory server on the cache responds with the contents of the requested pages, which enables the consolidation server to service the faults and schedule the faulting vCPU to continue execution.

When the user or VM become active, the VM is reintegrated back to the desktop. First, the desktop returns to running state. Then, the consolidation server halts execution

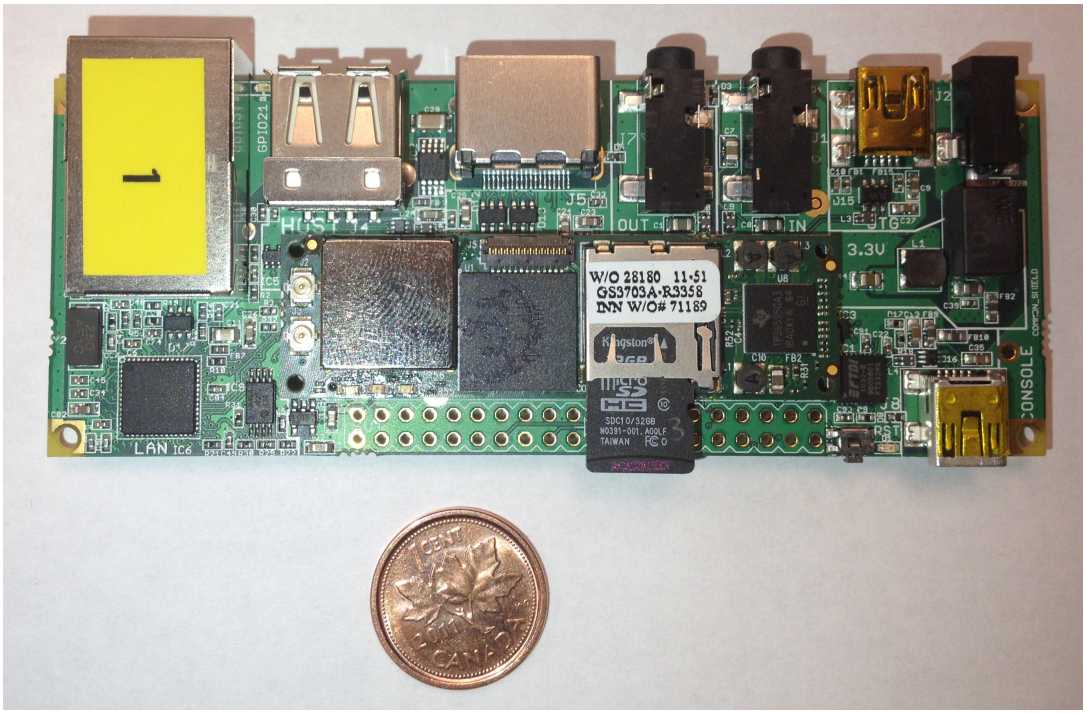


Figure 8.2: A Gumstix Overo AirSTORM computer-on-module and a one Canadian cent coin.

of the partial VM, transfers deltas consisting of dirty pages back to the desktop, and releases all resources allocated to the replica. The desktop applies the deltas, initiates execution of the VM, and notifies the cache of the reintegration. And finally, the cache shuts down execution of its memory server, and enters its low power state.

8.3 Oasis: A Low Power Page Cache

In this section, we present Oasis, our prototype implementation of a low power page cache, built using a Gumstix Overo AirSTORM computer-on-module (COM) platform. We discuss the challenges we encountered due limited upload bandwidth of the platform, and detail the techniques we use to address them. Finally, we present the architecture of our implementation.

The Gumstix Overo AirSTORM is a platform with a low power 800 MHz ARM-

based system-on-chip, 512 MB of DRAM and 512 MB of onboard flash storage. It has an expansion slot for a MicroSD flash card, a 100BASE-TX ethernet adapter, three USB 2.0 ports, an 802.11b/g network interface, as well as a bluetooth interface. Its three USB ports serve different tasks. The first, the host USB port, is an unpowered port that can support peripheral USB devices such as keyboard, mice and flash drives though a powered USB hub. The second, the on-the-go (OTG) interface, allows the Gumstix to act as a peripheral device so that it can be connected to a PC host. And the third port is a serial-over-USB port, providing serial access to the console over USB. We have configured the Gumstix Overo with a 32 GB Kingston SDC10/32 MicroSD card, a 32 GB Corsair Voyager USB flash drive, and set up the Angstrom Linux distribution 2011.03 as its operating system, running kernel version 3.0.0. Refer to Figure 8.2 for an image of the Gumstix Overo AirSTORM, in size comparison with a Canadian cent coin.

The Gumstix Overo is well suited for a low power page cache for the following reasons: (i), it is network accessible; (ii), it requires minimal power, averaging 2.8 W when idle; (iii), it supports large local storage within its MicroSD slot and USB flash; and (iv), it supports direct transfers from the PC. The OTG USB port allows the Gumstix to behave as a mass storage device, akin to a portable storage drive. In addition, the ethernet interface can be used with a local network switch, linking it to the desktop, to provide additional bandwidth, while isolating the transfer traffic from the shared network.

However, as we will show in the next section, the bandwidth provided by the USB and ethernet interfaces can be a limiting factor to taking advantage of short idle times, and in the next section we discuss strategies we have devised to cope with the bandwidth limitations.

8.3.1 Page Uploads to Cache

At VM consolidation time, the PC copies all VM memory pages in its DRAM and uploads them to the page cache. DRAM retains its copy of the pages as the PC enters low power

mode, with the goal of speeding up a future migration of the VM back to the PC. To copy the VM's memory pages, a process running in the administrative domain on the PC maps the VM's memory and reads the pages, writing a copy of each to a file in disk — the *VM memory image*. Subsequently, the memory image is transferred to the page cache. We first copy the image to disk before uploading it to the cache for simplicity of our implementation.

We begin by profiling the transfer rates available to upload memory pages from the desktop to the page cache over the USB and ethernet interfaces. For these experiments, we pair the Gumstix with one of our Dell Studio XPS 7100 desktops. We transfer a file of up to 256 MiB from the desktop to one of four storage media: the DRAM memory of the Gumstix, the onboard flash storage, the MicroSD card, and an attached USB flash drive. USB transfers are made by mounting the Gumstix's mass storage on the desktop. Ethernet transfers are performed using the Netcat TCP/IP utility. Each experiment was repeated three times.

Interface	Storage Medium			
	Memory	Onboard Flash	MicroSD	USB Flash
Local	182.5	1.39	5.97	6.95
USB	6.97	1.01	5.53	6.99
Ethernet	8.77	0.35	3.43	6.48

Table 8.1: Transfer rates to Gumstix (MB/s).

Table 8.1 shows the transfer rates achieved from the desktop to the Gumstix. The "Local" interface refers to a file being transferred from the Gumstix's own memory. We use this to benchmark the performance of the storage medium, independent of the transfer interface. The table presents two important findings. First, interface bandwidths, measured with transfers to the high speed Gumstix memory show the limitations of both, the USB and ethernet interfaces. USB is only able to sustain transfer rates of 6.97 MB/s,

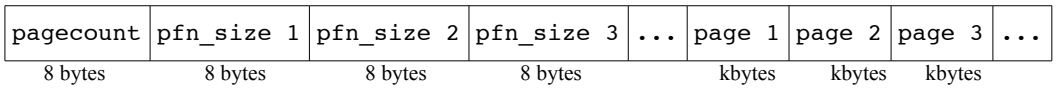


Figure 8.3: Memory image chunk format.

and ethernet, rates of 8.77 MB/s. With these throughputs, the latencies for transferring a 4 GiB memory image over USB or ethernet, exceed 10 minutes and 8 minutes, respectively. This means that the PC is unable to sleep for at least 8 minutes after desktop inactivity is detected. Second, the storage medium bandwidths, observed with local writes show that memory provides much higher write throughputs than the alternative storage media. Memory sustains local writes in excess of 182.5 MB/s, whereas onboard flash, MicroSD, and USB flash sustain only 1.39 MB/s, 5.97 MB/s, and 6.95 MB/s, respectively. Memory is the only medium with sufficient write bandwidth to saturate both, the USB and ethernet interfaces combined. However, memory has small storage capacity, insufficient for VM memory images. Recall that the Gumstix is equipped with 512 MB of memory, and this memory is used by the OS as well as all processes necessary to implement the page cache. In contrast, MicroSD and USB flash drives can provide high capacity storage, which we rely on for page uploads. These findings have lead us to the implementation of the bandwidth pooling, fast page compression, and selective delta upload schemes we discuss next.

Bandwidth Pooling. During memory uploads, we pool interface and storage bandwidth to improve upload throughput. This strategy partitions the memory image into *image chunks* on the PC, before transmitting them in parallel over the USB and ethernet interfaces to the storage media. Each chunk contains a sequence of pages and is transmitted as a file. The number of pages per chunk can vary as can the size of each page. Our defaults, however, are at most 2^{17} pages per chunk which, with standard pages of 4096 bytes, results in chunks with at most 512 MiB.

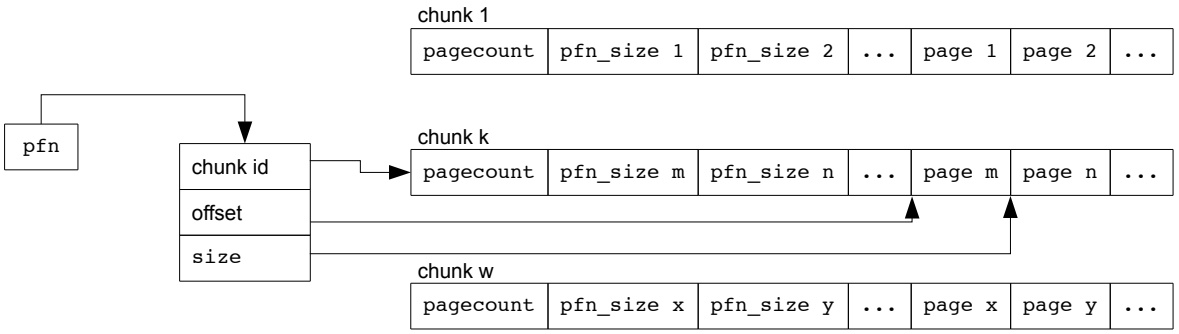


Figure 8.4: Page cache page table structure.

Figure 8.3 presents the structure of a chunk. Each chunk begins with a header containing a pagecount, the page frame number (PFN) and page size for each page in the chunk. The pagecount occupies the first 8 bytes and specifies the number of pages contained in the chunk. It is followed by an array containing pagecount fields of 8 bytes each named pfn_size. Each pfn_size field contains the PFN of the page in the top 4 bytes, and the size of the page in the lower 4 bytes. Finally, the file contains an array of pages arranged in matching order to the pfn_size array.

When the page cache receives a request for a given PFN, it locates the chunk where the corresponding page is stored and reads the page content. To do so, it relies on a page table. The page cache builds a page table each time it receives a chunk from the PC, by parsing the headers of the chunk files. The page table consists of the ID of the chunk containing the page, the byte offset from the start of the chunk file, and the page size, which, together, tell the page cache where to find the page for a given PFN and how many bytes to read. Figure 8.4 shows a representation of the page table.

Fast Page Compression. We have implemented fast page compression on the desktop to reduce data transfers to the page cache. Each page is compressed using the Lempel-Ziv-Oberhumer [80] real-time compression library before it is written to the image chunk. The Lempel-Ziv-Oberhumer library implements fast lossless data compression for use in

real-time applications, and provides high compression ratios.

When generating the image chunks, the PC stores the size of the compressed page in the corresponding pfn_size field for the page, shown in Figure 8.3. The page cache treats compressed pages as opaque data. Only the consolidation server decompresses pages when handling page faults.

Selective Upload of Deltas. Desktop VMs experience multiple migrations to the consolidation server and back to the PC. At each consolidation migration, the memory image of the VM is uploaded to the page cache. Not all pages being uploaded to the page cache will have been updated, however. As a result, only pages that have been modified need to be uploaded. To selectively upload only updated pages, we require three capabilities: (i), the page cache must persist previous pages; (ii), the consolidation server must track pages that are modified as the VM runs on the server; and (iii), the PC must track the pages that are modified in the PC, after reintegration. The first requirement is satisfied by the use of persistent storage in the page cache. For the second requirement, we already track dirty pages on the consolidation server, in order to provide dirty state reintegration on the PC. Finally, for the third requirement, we use shadow page tables to track pages that are dirtied on the PC.

We also considered uploading incomplete memory images to reduce upload sizes. In this model, the cache is allowed to have only a fraction of the VM's pages (e.g., the most frequently accessed pages). When the consolidation server requests a page that is not present in the cache, the cache can return a zero byte page to indicate that the page is non presence. The consolidation server must then trigger the desktop to resume and request the page from there, as we already do with disk blocks. Our experience with the previous strategies has demonstrated they are sufficient to make the approach practical, even in short idle times, and we leave an exploration of incomplete image uploads to future work.

Garbage Collection. Persisting chunks in the page cache leads to the inevitable saturation of storage capacity. We reclaim storage space with periodic garbage collection. When storage availability falls below a threshold, the garbage collector works to consolidate image chunks such that stale pages are removed. Stale pages are pages for which a newer version exists in a more recently uploaded chunk. Because within a chunk only a fraction of pages may be stale, we must ensure that the active pages are not removed. We do so by consolidating active pages from various chunks into a new chunk, before deletion of the old chunks.

The garbage collector is invoked whenever the VM is reintegrated back to the desktop, and the page cache is notified. At such point, the VM no longer runs in the consolidation server and, as a result no longer requests pages from the page cache. As the garbage collector runs, any attempt to upload additional chunks by the PC is delayed by the page cache until the end of garbage collection. The garbage collector operates on two thresholds, a high watermark, which indicates an imminent shortage of space, and a low watermark, which is the target availability level of the garbage collector. When invoked, the garbage collector, creates a thread for each storage medium. The thread first checks that usage levels do not exceed the high watermark of the storage medium. Otherwise, the thread will consolidate the chunks within the medium until the usage level drops to at most the low watermark. A natural high watermark is one that ensures that each medium can accept its fraction of a full VM image. Thus, for our setup of two media (MicroSD card and a USB flash drive) that have comparable transfer rates, and VMs with 4 GiB of memory, we set the high watermark to 2 GiB of free space. For the low watermark, we set it such that in the worst case, the medium may store all of the active pages of the VM, in our case 4 GiB.

The garbage collection algorithm sorts chunks based on the cumulative size of stale pages, and consolidates the chunks for which stale pages occupy the largest space first, which ensures the largest savings with the fewest chunk consolidations. The cumulative

size of stale pages for each chunk are computed by the memory server as it builds the page table data structure. When the server reads the header of each chunk, it tallies the cumulative size of all pages in the chunk, and initializes the size of active pages with the same value. Before a page table entry is updated to reference a newer copy of the page in a new chunk, the memory server decreases the size of active pages for the chunk by the size of the page. The cumulative size of stale pages is the difference of these tallies. The garbage collector walks through the page table to extract pages that have not become stale in chunk that is scheduled for deletion, and copies those into a new chunk.

8.3.2 Oasis Implementation

Oasis builds upon the functionality of Jettison by moving the memory server from the PC to the low power cache. Recall from Chapter 6 that the consolidation server runs the following components:

memtap. A process that monitors page faults by the idle VM and fetches pages from the memory server. In Oasis, page requests are directed to the server running on the page cache.

remoteWakeup. A daemon that wakes up a PC that is in low power mode when it receives a request for a page or disk block. In Oasis, the PC is only woken-up for disk block requests originating in the network disk driver. RemoteWakeup uses the Wake-on-LAN protocol to wake the PC up.

cownetdisk. Our network disk driver backend, that migrates disk blocks from the PC on demand, and tracks block writes on the consolidation server for reintegration on the PC. Cownetdisk first notifies remoteWakeup of block requests to ensure that the PC is awake.

On the administrative domain of the PC, Oasis runs the following components.

activityMonitor. A daemon that monitors user activity. When it detects inactivity, it initiates migration of the VM to the consolidation server, and the upload of pages to the page cache, by invoking cacheManager.

cacheManager. A daemon that manages chunk uploads to the page cache. It extracts the memory of the idle VM from DRAM, compressing each page, partitions the memory image into chunks suitable for uploading over multiple interfaces, and transfers them to the page cache, with the help of the cacheDaemon running in the cache. On VM reintegration on the PC, it notifies the cacheDaemon so it can perform any garbage collection.

disksrv. A network accessible disk block server, provides access to the VM disk image to the consolidated VM over TCP.

And finally, on the page cache, Oasis runs the following components.

cacheDaemon. A TCP daemon responsible for coordinating the transfer of VM chunks. It listens for connections from the cacheManager and, on connection, receives a chunk list, indicating the names of chunks about to be uploaded and the interfaces to be used. In response, it sets up the interfaces for chunk reception and notifies the cacheManager of the location to send each chunk. For chunks to be sent over the ethernet interface, it sets up TCP servers and notifies of their IP and port combinations. For chunks to be sent over USB, it sets up the medium for USB access and notifies the cacheManager of the directory location to store the chunks. On reception of all chunks from the PC, it initializes the memsrv page server to run on the cache. Finally, when notified of reintegration of the VM on the PC, it manages the orderly exit of the memsrv process and garbage collection.

memsrv. A TCP page server. On initialization, it assembles a page table of all pages contained in chunks received, ensuring that page table entries always point to the most recently received version of pages. The page table maps each PFN to the chunk and offset containing the page, as well as the page size. It allows fast lookups of pages when processing a page request. When a VM is consolidated, an instance of memtap is created in the consolidation server, which establishes a TCP connection to memsrv. When a fault occurs in the consolidated VM, memtap issues a request containing the PFN of the faulting page. In response, memsrv finds the storage location containing the page, by traversing the page table, and sends a response containing, the PFN, the page size and the page itself.

8.4 Experimental Evaluation

In this section, we show that a low power page cache increases the energy savings of partial VM migration, that it is practical for use even in short idle periods, and that the use of a cache reduces worst case page response times, a benefit for applications that rely on timely delivery of events. Specifically, our experiments answer the following questions:

1. How long are memory uploads to the page cache? Long memory uploads limit the use of Oasis during short idle periods.

2. Does Oasis improve energy savings? We are especially concerned with short idle periods, where we found Jettison to deliver only modest savings.

3. How long are response times for page faults? Long page response times result in slow application execution, and may have adverse effects on time-sensitive applications.

Workload	Applications
Workload 1	Mozilla Thunderbird mail client; Pidgin IM client; OpenOffice.org with one presentation, one word document and one spreadsheet; Evince PDF viewer with an open PDF document; Firefox Web browser with CNN.com, Slashdot.com, Maps.Google.com, the SunSpider JavaScript benchmark [12], as well as Acid3 Web standard compliance test [7] loaded.
Workload 2	After reintegrating a VM running Workload 1, we add the following documents: Firefox opens Shopping.HP.com, CDW.com, BBC.co.uk/news and TheGlobeAndMail.com; OpenOffice.org opens one additional presentation, one additional spreadsheet and one additional word document; Evince opens an additional PDF document.

Table 8.2: Desktop workloads. All experiments use Workload 1. Workload 2 is used to evaluate the size and duration of selective uploads of deltas.

8.4.1 Methodology

We use the two desktop workloads described in Table 8.2 to evaluate the performance of Oasis. These workloads mimic a heavily multitasking user, with many applications running at a given time. In all experiments, we run the applications of Workload 1 inside our VM, allow the VM to idle for five minutes before migrating it to the consolidation server. To measure the size and duration of uploads of delta images, after consolidating the VM running Workload 1 for a period of thirty minutes, we reintegrate the VM back to the desktop and load the applications from Workload 2. We idle the VM for five minutes and subsequently migrate it to the consolidation server for a second time. We ran all experiments three times.

These experiments use the same desktop, server and VM configurations as in Section 6.2.1. Specifically, the desktop was a Dell Studio XPS 7100, the server a Sun Fire™

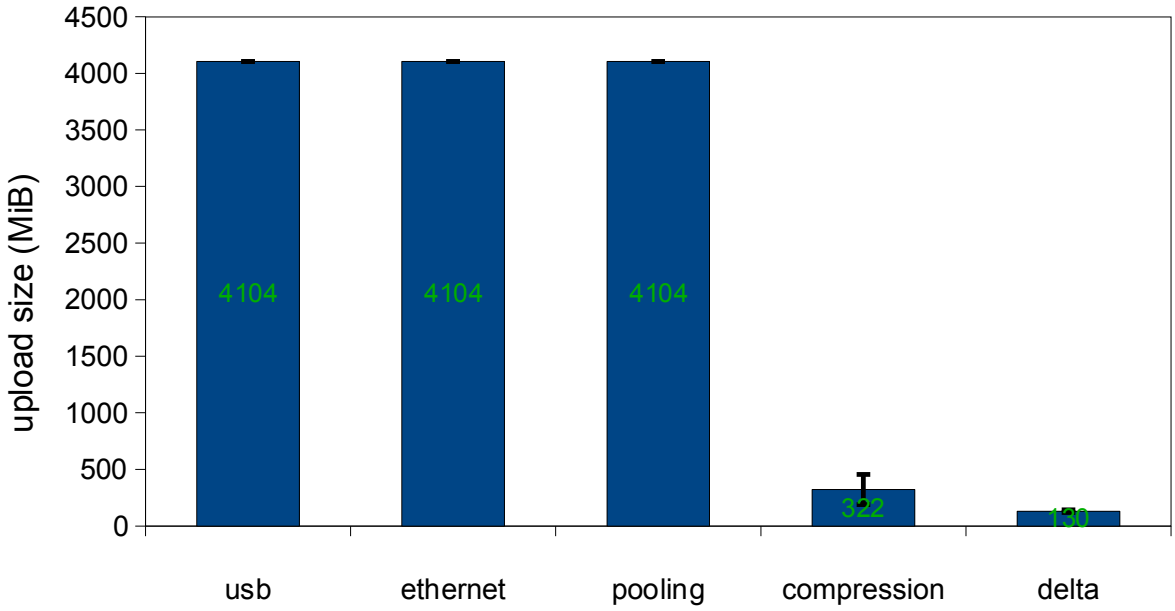


Figure 8.5: Sizes of memory images uploaded to the cache. Compression and selective upload of deltas reduce reduce upload sizes considerably.

X2250, and the VM had 4 GiB of RAM, 12 GiB of disk, and ran the GNOME desktop environment.

8.4.2 Memory Uploads

To demonstrate the benefits of the schemes we developed to address the shortcomings of the upload bandwidth of our platform, we compare the sizes of the memory images uploaded from the desktop to the cache and the upload duration. Crucially, upload duration reduces the desktop's ability to save energy, as the desktop must remain awake for the duration of the image upload.

Figure 8.5 shows the size of the memory image of a VM that is uploaded after running the tasks in Workload 1. The upload sizes are shown for the five upload schemes we used: *usb* transfers all VM pages over the USB interface alone; *Ethernet* transfers all VM pages over the ethernet interface alone; *Pooling* uses both the USB and the ethernet interfaces to transfer pages to the MicroSD card and the USB flash drive on the cache;

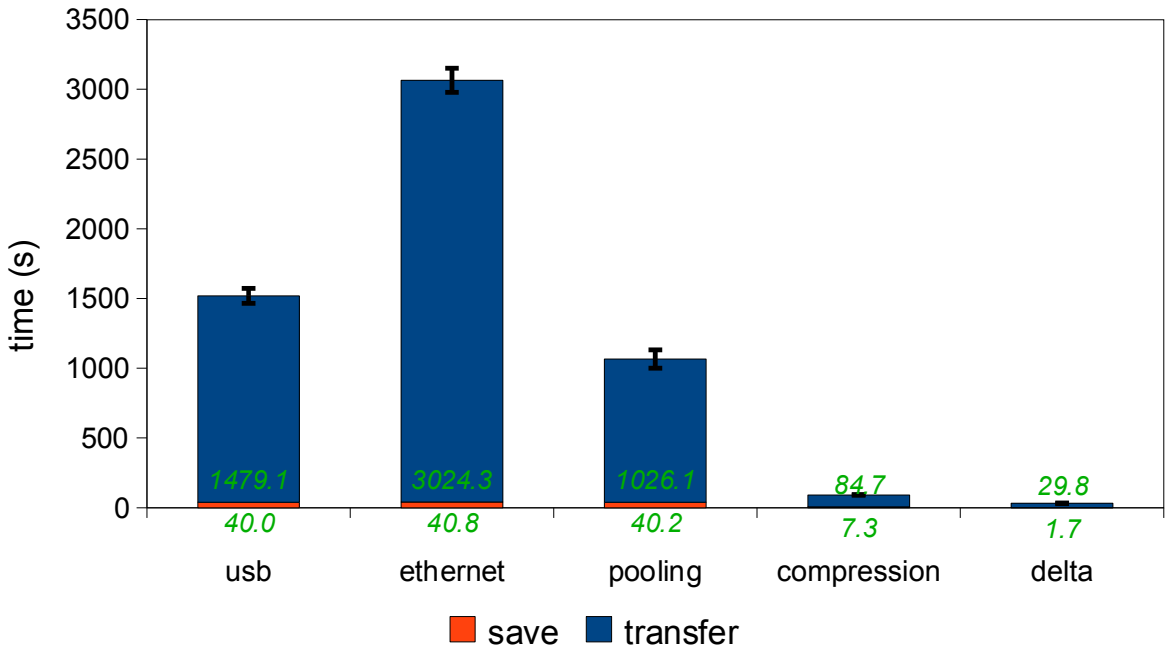


Figure 8.6: Upload durations under various upload schemes. Pooling, compression and selective delta uploads reduce uploads to durations that are usable in short idle times.

Compression applies per page compression before transferring the pages to the cache over both interfaces; And *delta*, transfers only compressed pages that have been updated during execution on the server or during execution of Workload 2 on the desktop.

Figure 8.5 demonstrates the benefits of compression and delta uploads in reducing upload sizes. Compression reduces upload sizes by 92% from 4104 MiB to 322 MiB. Selective upload of deltas reduces upload sizes by an additional 60% to 130 MiB.

Figure 8.6 shows how the three schemes reduce upload durations. The figure breaks down upload duration into save time and transfer time. Save is the time taken to copy the VM pages from the desktop's memory to disk, and transfer, the time used to transfer the image from disk to the cache. Save time corresponds to a small fraction of upload times, never exceeding 8% in all cases. The figure shows that uploading images over a single interface makes the cache unusable in short idle times: upload times exceed 25 minutes over USB and 51 minutes over ethernet. Bandwidth pooling reduces upload times to

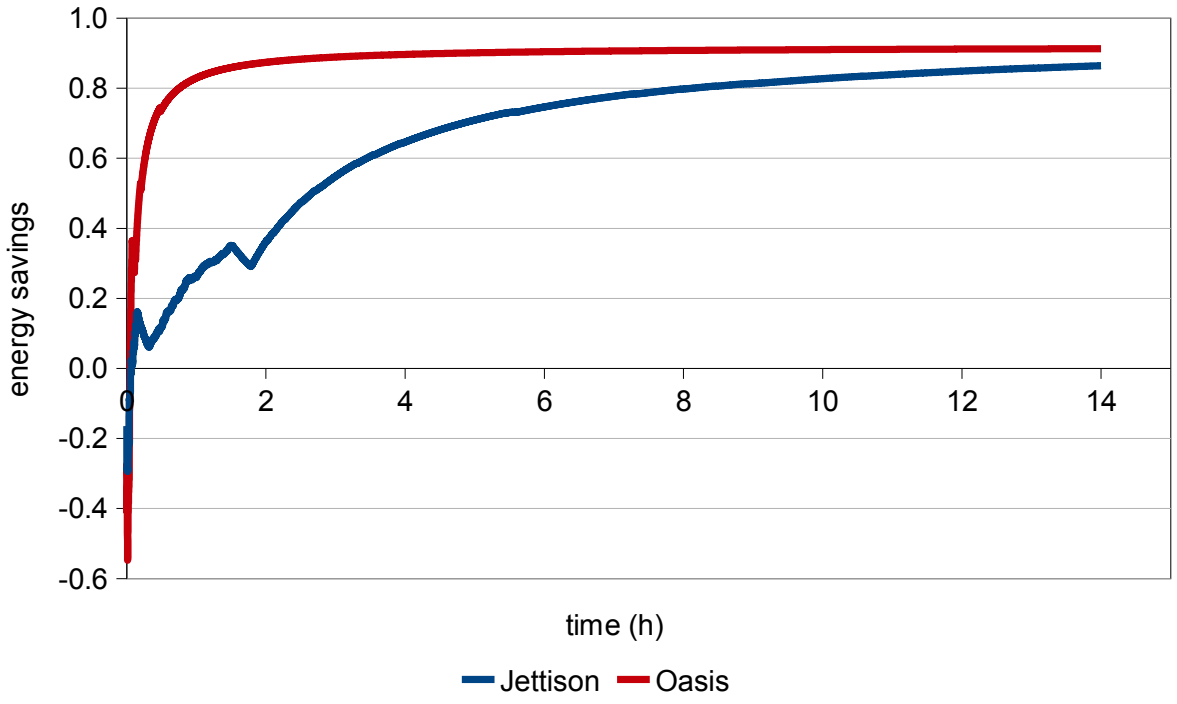


Figure 8.7: Energy savings of desktops with Oasis and Jettison over a 14 hour consolidation of their VMs. Oasis achieves higher savings and makes consolidation practical during short idle times.

under 18 minutes. Compression is effective at reducing upload times to 92 seconds, which makes the approach usable in short idle times. Uploading deltas selectively reduces upload times further to under 32 seconds. We expect typical upload times to fall between the last two values, depending on the workloads the user is running. These upload times are short enough to enable the desktop to sleep in short idle times that are in the order of minutes.

8.4.3 Energy Savings

Figure 8.7 shows the energy savings over time of a desktop whose idle VM, running the tasks of Workload 1, is consolidated for a period of 14 hours. The figure compares the energy savings we measured for the same VM consolidated with Oasis and with Jettison.

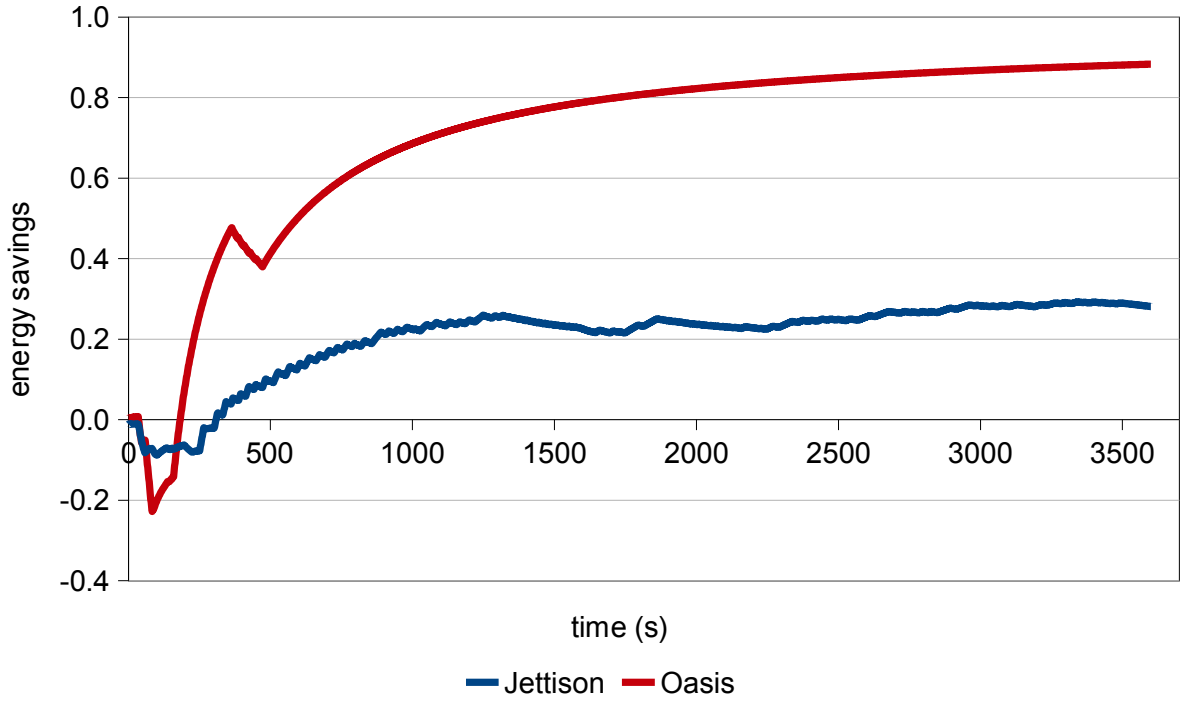
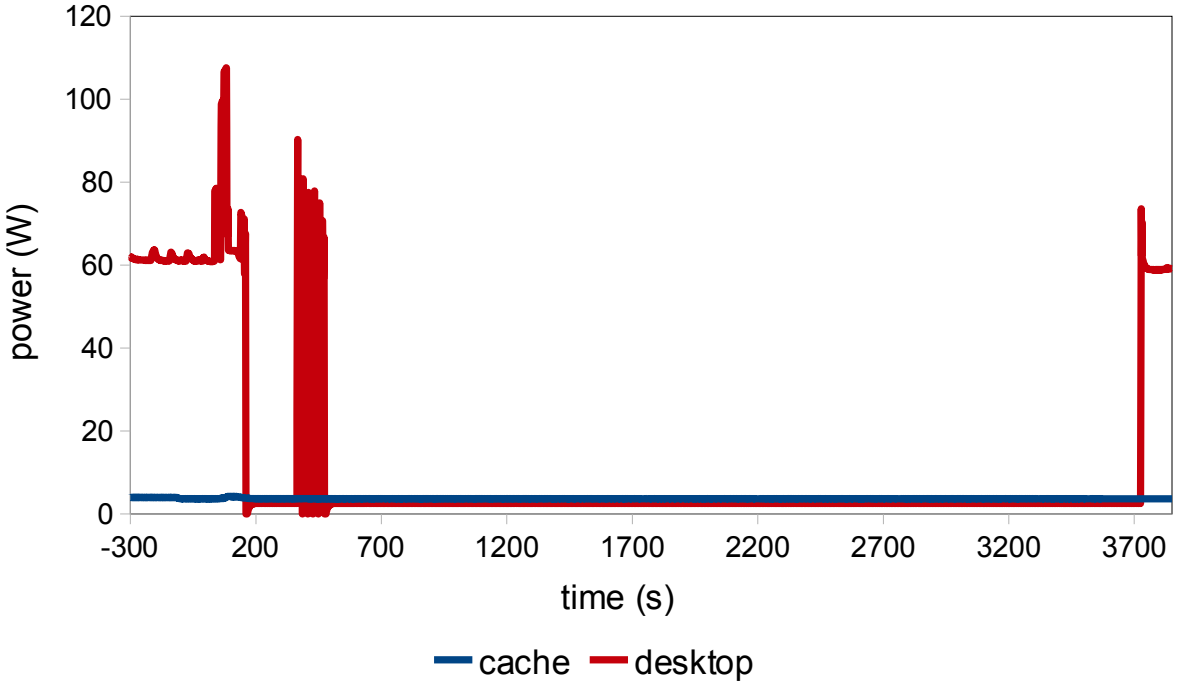


Figure 8.8: Energy savings of desktops with Oasis and Jettison over an hour-long consolidation of their VMs. Oasis achieves higher savings and makes consolidation practical during short idle times.

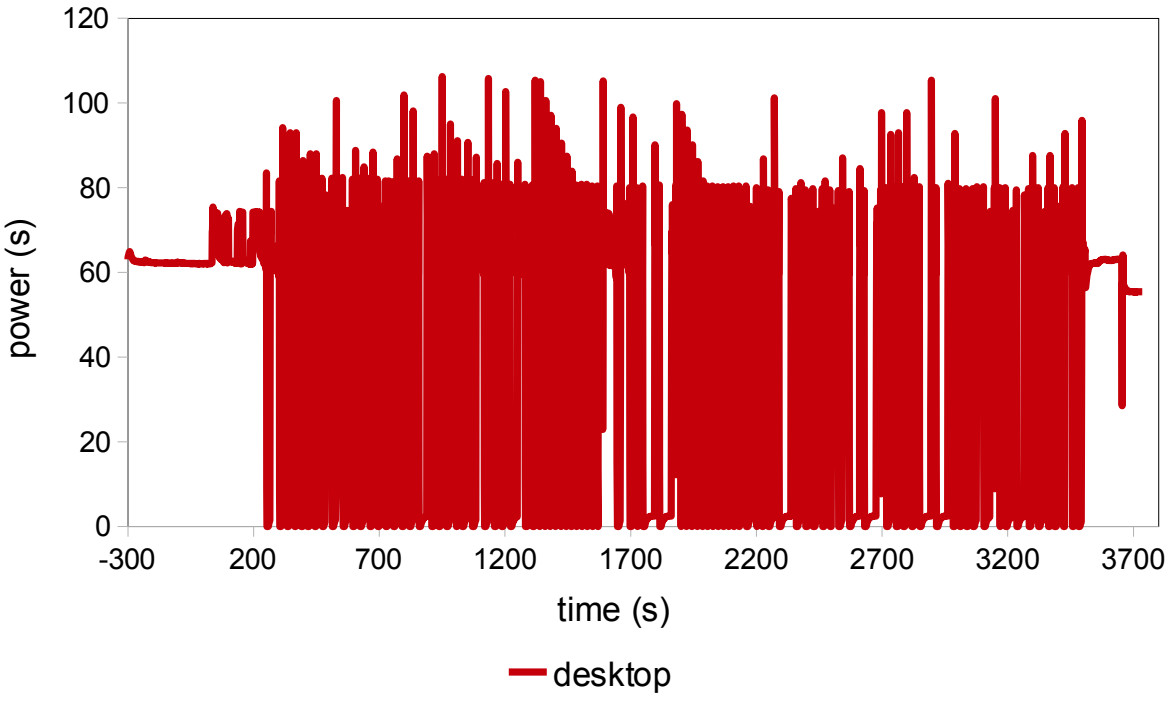
The savings are normalized over the idle power of the desktop of 61.4 W. For Oasis, we measured the power used by the desktop, the Gumstix platform, and USB hub hosting the USB flash drive. The figure shows that Oasis delivers substantial savings very early in the consolidation, and continues to sustain high savings throughout the fourteen hours. In the first five minutes, Oasis delivers savings of 34%. In ten minutes, the savings increase to 45%. In twenty minutes, it produces savings of 67%. After an hour, the savings reach 83%, and finally, after 14 hours they reach 91%. In contrast, Jettison produces less substantial savings early on, improving only with the duration of the idle time. After five minutes, Jettison delivers savings of only 2%. At subsequent marks of ten minutes, twenty minutes, one hour and fourteen hours, Jettison delivers savings of 15%, 7%, 26%, and 86%, respectively. In long idle times, close to fourteen hours, savings of both approaches begin to converge.

Figure 8.8 shows energy savings for a separate, hour-long consolidation. Within 5 minutes, Oasis delivers 38%, compared to a deficit of 2% for Jettison. After 10 minutes, Oasis saves 50% of the energy, in contrast to 13%. After 20 minutes, Oasis yields savings of 73%, compared to 25% under Jettison. And finally, at the end of the hour, Oasis achieves savings of 88%, far above Jettison's 28%. We note in the figure, that in the interval between 365 seconds and 469 seconds, savings with Oasis fell from 48% to 38%. This was caused by disk accesses that woke the desktop several times during that interval.

To help us understand how the energy savings are achieved, we look next at the power used by the desktop and the cache during consolidation. Figure 8.9(a), shows the power usage of the desktop and the Gumstix-based Oasis cache for the same hour-long consolidation shown in Figure 8.8. The x-axis begins five minutes before consolidation to show the systems' idle power. Throughout the experiment, the cache uses a near constant power at a rate that averages 3.65 W. The desktop shows an initial surge at the zero minute mark, during image upload, and soon after, it enters low power. The desktop later wakes up five times starting at the six minute mark to service disk requests,



(a) Oasis



(b) Jettison

Figure 8.9: Power usage.

which correspond to the reduction in savings in Figure 8.8, after which it is able to sleep uninterrupted. By comparison, in Figure 8.9(b), we see that the Jettison desktop is frequently transitioning between low and full power mode for the entire duration of the idle time, which limits its energy savings.

The savings delivered by Oasis are possible because the desktop does not wake up to service page requests. Rather, it wakes up only for the rare disk requests. The penalty for using Oasis with the Gumstix platform is the early energy expenditure to upload the memory image, a cost seen in the increasing energy deficit up to 23% between 0 s and 85 s. These results show that Oasis makes consolidations over short periods, as small as five minutes, practical, yielding savings of 34% to 91%. In intervals of five minutes, Oasis increases savings 14 to 20 fold over Jettison. In intervals of ten minutes to an hour, it increases savings by two to five times.

8.4.4 Page Response Time

We now show that serving pages from a low power cache has the added benefit that it limits page fault response times, something that is desirable for the performance of the consolidated applications. The consolidation server no longer has to wait for the desktop to wake up in order to service page faults. Response time, is the time it takes to request, receive and commit a page to the partial VM on the consolidation server, whenever a fault occurs.

Figure 8.10 shows the average response times for a VM consolidated with Oasis and one consolidated with Jettison as well as their standard deviations. The average response time is shorter with Jettison than with Oasis at 4.26 ms versus 16.79 ms. This is expected because, as discussed in Section 5.4, for Jettison, more than 99% of page requests are serviced by a desktop that is already awake. And the desktop serves pages from DRAM over a 1 Gbps NIC, unlike the cache which serves from flash storage and over a 100 Mbps NIC. However, as shown by the standard deviations, the variations in response times

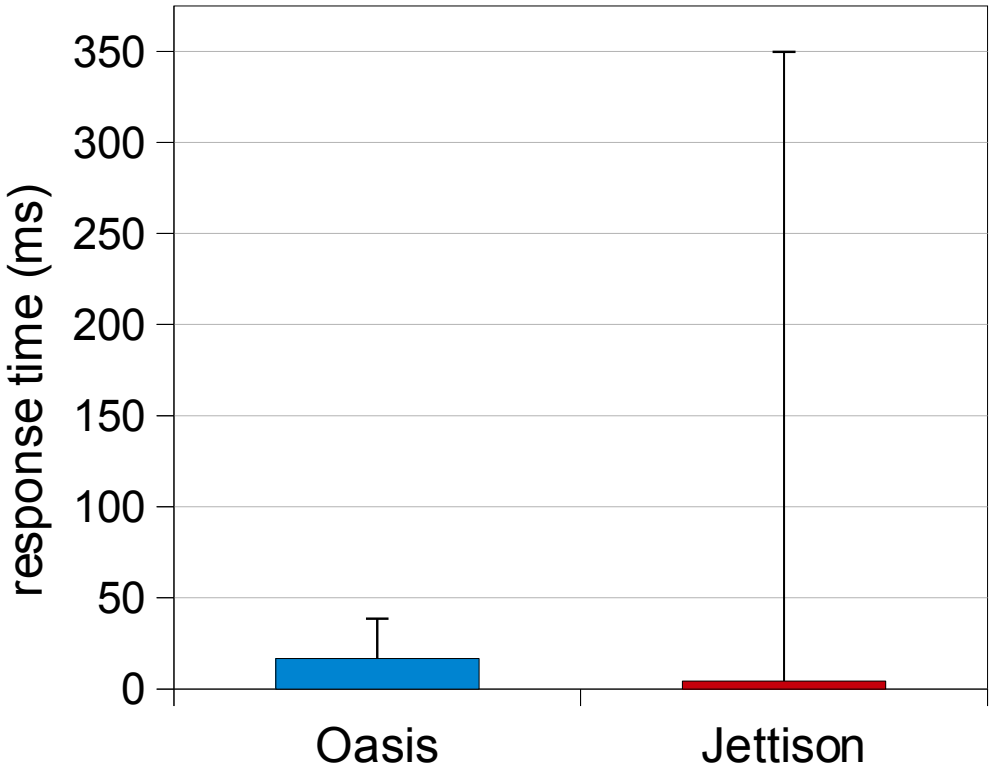


Figure 8.10: Page fault response times. With Jettison, response times vary widely, as a function of the desktop's power state.

with Jettison is much larger. There were 48 requests with response times above one second, all of which exceeded 6 seconds, with the longest lasting 92 seconds. The median response times for these requests was 24 seconds. With Oasis, no request had a response that was above sub-second time.

8.5 A Blueprint for Commodity Page Caches

We have presented a low power page cache prototype that couples a Gumstix computer-on-module platform with our desktop PC. We expect PC vendors to embed cache functionality in the hardware of the PC. An embedded cache need not be constrained by transfer bandwidth from the PC's DRAM and may be made simpler and even more energy efficient as it can take advantage of components already present in the desktop.

We explored approaches to implement the low power cache vision within off-the shelf desktop hardware. A viable cache requires a low power microprocessor that is embedded in the hardware of the PC, and that is able to function when the PC is in low power state. A likely candidate is the microprocessor that is embedded in network interfaces of desktop PCs. NICs are attractive hardware for low power caches for the following reasons: (i) NICs are network accessible, even when the PC is in low power; (ii) they require little power to function; (iii) they have sophisticated controllers capable of processing network packets unaided by the CPU; (iv) they are widely deployed and virtually all PCs have network interfaces capable of stand-alone operation.

Two significant tasks have led the development of microprocessors in NICs: the need to transfer data to and from memory without CPU assistance, provided by direct memory access (DMA), and the need to wake the PC from low power mode via network requests. The NIC controller continually inspects network packets and, on receipt of a prescribed packet issues an ACPI power management event that brings the PC out of low power.

In the next sections, we explore two approaches to leverage NIC functionality to realize

the vision of a low power page cache, and identify the limitations of current hardware. An important consideration in implementing the cache with NIC hardware is the placement of VM memory image. In the first approach, we explore the use of DMA to serve pages directly from DRAM, and in the second, we explore the use of storage that is local to the NIC.

8.5.1 Serving pages from DRAM

DMA enables peripheral devices to access DRAM memory, with minimal CPU intervention. Traditionally, to transfer data from DRAM to a peripheral (e.g., NIC) and vice-versa, the CPU copied each byte from source to destination. DMA enables the peripheral itself to access designated DRAM buffers via a shared system bus, freeing up CPU cycles. DMA requires three entities. The CPU, which sets up DRAM buffers, the system bus, which arbitrates access to DRAM, and a controller capable of DMA in the peripheral device. The device controller performs the transfer of data over the bus, and when finished notifies the CPU with an interrupt. DMA can enable direct access to VM memory without the penalty to upload images to the page cache. This in turn enables immediate availability of the cache. However, as we will show below, DMA functionality is currently incompatible with low power mode operation, and the trend in hardware development, which integrates memory controllers with CPUs and graphic devices inside a single die makes implementation of low power DMA more difficult.

We begin with a step-by-step example of DMA transfer of data from DRAM to the network by a network interface. This example highlights the tasks performed by the three main entities involved in DMA: the device driver (running in the CPU context), the bus controller, and the NIC controller. It assumes that a user application invokes the write system call on an open socket, and is used to illustrate current limitations of DMA.

1. When an application opens an initial socket connection, the driver requests an

interrupt line as well as a DMA channel from the bus controller and registers its interrupt handler with the OS kernel. The interrupt line is used by the device controller to notify the driver of DMA events.

2. When the application invokes the write system call, to write data to the network, the device driver allocates a buffer in DRAM for DMA and copies outbound data to this buffer. It configures the device controller with the buffer address and direction of DMA transfer (to the device). Finally, the driver enables DMA on the device to initiate the transfer, by setting an appropriate device register.

3. The device controller performs the transfer from the DMA buffer out to the network. When finished, it raises an interrupt to the CPU, to notify the driver that the transfer has finished.

4. The CPU invokes the driver's interrupt handler, which in turn disables DMA on the device, releases both the DMA channel on the bus and the interrupt line, and may release the DMA buffer.

The steps above illustrate two attributes of DMA that are presently not compatible with the page cache. First, with DMA, the CPU sends a single batch instruction to the device (i.e., send data in this buffer and when finished notify me). The page cache operates in interactive mode. It waits for page requests from the network, and performs DMA on-demand to respond with the requested page. Upon sending a response, the cache must continue to wait for additional requests. The functionality of the cache requires changes to the operation of the controllers which we expect to be feasible for vendors. Modern NIC controllers are general purpose microprocessors whose operating instructions reside in device firmware that is stored within the device's own read only memory (ROM) [5, 3], or that is loaded by the OS on device initialization [4].

Second, DMA requires services from the motherboard. Peripheral Component Interconnect (PCI) NICs, for example, require the chip implementing the PCI bus, as well as

the memory controller to remain powered while the page cache runs on the NIC. PCI buses are normally implemented in the "southbridge" chip and the memory controller in the "northbridge" chip of the motherboard. The northbridge is an interconnect chip linking the CPU to DRAM and video cards with fast interfaces (AGP and PCIe), and the southbridge a chip linking lower bandwidth peripheral devices (disk, audio and some NICs) to the northbridge. In order to provide the necessary buses, DMA requires at least part of the motherboard to remain powered. Recall from Section 2.4 that the motherboard consumes between 20% and 30% of the PC's peak power, and that its dynamic power range is limited. To make matters worse, the recent trend to integrate the memory controller (and more generally the functionality of the northbridge chip) inside the CPU core, seen in Intel Core processors since "Sandy Bridge" models, as well as in AMD Fusion processors, has the effect that access to memory now requires the CPU itself to remain powered. A DMA-based solution requires a rethink of motherboard and processor designs so that they are modular and can provide access to memory with minimal components powered.

8.5.2 Serving pages from NIC storage

To enable cache operation without motherboard and CPU services, during VM consolidation, desktops may place VM memory on storage that is resident on the NIC. When the desktop user is idle, the desktop uploads the VM pages from DRAM onto the NIC, before migrating the VM to the consolidation server. Once the VM is migrated to the server, the desktop PC enters low power mode, while its NIC remains powered. The NIC takes on the role of serving the memory pages out of its own local storage. When a fault occurs on the consolidation server, and the server issues a page request to the desktop, the NIC responds with the desired page.

This approach requires NIC storage, capable of accommodating the VM's memory pages. Unfortunately, desktop NICs have limited amounts of storage space. Normally,

these devices have a small ROM for storing their operating firmware and a small amount of volatile memory, known as the packet buffer, used by the microprocessor to process incoming and outgoing packets. For example, Broadcom BCM5751 and BCM5752-based gigabit NICs have ROM storage of 64KB and packet buffers of 96 KB [4]. Intel PRO/1000 GT gigabit desktop adapters, based on the Intel 82541PI processor, have 64 KB of packet buffer. In contrast, desktop PCs, and therefore desktop VMs, have gigabytes of memory. It is obvious that current desktop NICs are unable to host any significant fraction of VM pages. This is despite the fact that we do not consider using the cache to service disk requests, as disk storage requirements are even larger than memory, and idle accesses infrequent (less than 1% of requests per Section 5.1.3). For this approach to work, PC vendors must incorporate storage capacity that is accessible to the network interface. This can be accomplished by adding flash storage to NICs or by enabling NICs to access the desktop's own storage.

NIC-based implementations of the page cache may benefit from higher bandwidth between DRAM and the NIC. A high bandwidth bus, such as PCIe, can make it unnecessary to pool the bandwidth of multiple channels. However, storage limitations still require careful management of VM images. For example, flash storage has limited capacity and bandwidth. Our page lookup architecture, that is based on page tables, enables low page response times. Uploading only VM image chunks consisting of state deltas limits the storage requirements of the cache, as well as the time to write to storage. Page compression is also effective in reducing the size of the pages that are uploaded. Garbage collection enables the reclamation of storage from pages that have become stale. The architecture we developed is general enough to support caches based on devices with local storage, the more likely path for implementation of a low power page cache.

8.6 Summary

In this chapter, we introduced Oasis, a low power page cache that reduces the energy cost of serving pages on-demand, and increases energy savings of partial VM migration, particularly in short idle times. This approach deploys a low power platform within the hardware of the PC, that is able to serve pages to the consolidation server, even when the PC enters low power state. At consolidation time, the PC uploads a copy of the VM's memory pages to the page cache, before entering low power mode. We presented Oasis, a prototype that is built using the Gumstix Overo platform, and discussed the challenges that must be addressed to support a page cache functionality in commodity desktops. We found that the Gumstix platform provides limited upload bandwidth from the desktop and limited storage capacity. We expect bandwidth and storage capacity limitations to exist in even in NIC-based implementations, and have developed strategies to address the impact of these limitations on upload times, page response times, and storage exhaustion. We partition VM memory images into units of chunks, for upload and storage management. We use a page table data structure to enable fast lookup of pages in service of requests. We use fast page compression and selective upload of deltas are used to reduce transfer and storage sizes. And finally, we use garbage collection to ensure that storage remains available for upload of memory pages.

Our experimental results demonstrate that Oasis makes consolidations over short intervals, of as little as five minutes, practical, yielding savings of at least 34%, which increased to up to 88% after an hour for our heavy workloads. Oasis increases energy over Jettison 14 to 20 fold over five minute intervals. Over intervals of ten minutes to an hour it increases savings two to five fold. We showed that, when uploading VM images from the desktop to the cache, pooling of bandwidth reduces upload times by at least 29%. Page compression reduces upload times by an additional 93%, and selective uploads of deltas by another 57%. These strategies were all necessary to make Oasis practical in short idle times. Finally, we showed that Oasis provides page responses with low latency,

a desirable characteristic to ensure brisk performance of consolidated applications.

Chapter 9

Related Work

This dissertation 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, 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 presented two additional refinements that improve energy savings of partial VM migration, namely Context-Aware Selective Resume and Oasis, a low power page cache. Context-Aware Selective Resume reduces desktop resume and suspend times by re-initializing only devices and code necessary for a given wake-up context. Oasis, offloads page serving duties to a low power page cache that is network accessible, and enables desktop PCs to remain in low power state for long periods, without interruption.

In this chapter we discuss prior art in two areas to which our contributions apply: energy conservation in desktop systems, and virtual machine migration. We extend upon the discussions of Chapter 2 and Chapter 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 [79, 52, 112]. Recently, energy usage and heat generation has become a concern in data centers [36, 74, 51, 99]. In more recent years, energy use of desktop computers has also garnered interest of researchers, motivated by rising energy costs and the environmental impact of electricity generation [104, 58, 29, 77, 30, 47, 90].

9.1.1 Thin Clients

Thin clients, such VNC [85] and Sun Ray [88], 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 forward 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, while thin clients reduce energy use of the client, they do little to improve energy efficiency of the servers. This is because servers run user sessions with full state, and must be provisioned to accommodate the peak workloads of each user, 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 this enables the multiplexing of hardware among many user sessions.

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% [36, 74, 51]. A recent study com-

missioned by the New York Times, has found that between 88% and 94% of the power used by IT data centers is wasted in idle servers [55]. While 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 [36]. Rather than requiring fine grain power-performance tuning from all components, PowerNap [74] calls for systems that transition between high performance, active state, and low performance, nap state, rapidly in response to instantaneous load changes, and systems that support energy efficient sleep states. Studies performed by Meisner et al. [74] show that for this approach to work, state transitions must take no more than 10 ms, 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.

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 the 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.

Tolia et al. [99] approximate energy proportionality at the ensemble by taking advantage of dynamic voltage and frequency scaling, and when opportune, consolidating VMs and shutting down idle servers in the data center. Their approach relies on full migration of VMs, which we have shown to be slow, cause large network traffic, and not to inherently lead to high consolidation ratios.

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 [71]

was developed by the Intel and IBM Manageability Alliance to allow system administrators to wake up 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 up. Wake-On-Wireless-LAN [62] extends this functionality to 802.11 wireless networks. Wake-On-Wireless [91], is targeted at mobile computers such as laptops. It supplements the mobile device with a low power radio that is used to wake up the host, in lieu of maintaining the 802.11 NIC powered during sleep time. An external proxy is used to signal the device over the low power radio interface.

CellNotify [28] substitutes the low power radio interface for a cellular radio, enabling it to work over the wide area. While acceptable to mobile communication devices with a cellular interface, a widespread adoption of the approach is infeasible in desktop environments.

While 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 presence.

Exploiting short opportunities for sleep while a host is waiting for work is also explored in Catnap [49]. 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

To support applications with always-on semantics, Gunaratne et al. [58], Jimeno et al. [66], Nedevschi et al. [77], and Reich et al. [84], propose delegating basic functions of the desktop, such as maintaining online presence, to proxies whenever the desktop goes to sleep, and only waking up the desktop when its resources are required. Sen et al. [90] develop a distributed architecture which enables participating PCs to act as proxies for sleeping hosts in the same subnet. Somniloquy [29] delegates basic application functionality to application stubs that run on the 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 up when necessary. Turducken [94] is a more general instance of the proxy-based approach, which 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 up tier two accordingly.

Proxy based approaches require that always-on applications be re-engineered to support bi-modal operation between the host and proxy, and, even for solutions whose proxies only handle simple protocols and wake up the desktop on more sophisticated requests [77, 84], they are unable to support the plethora of cloud applications with architectures that are client driven. For example, Reich et al. [84] found that for cloud storage applications Microsoft LiveMesh and 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 [8], Google Docs [10], Gmail [9], are just few examples of applications that are growing in popularity, for which proxy triggered wake on-demand does not work.

SleepServer [30] 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 as 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 virtual machines has been used in the data center to increase efficiency of servers [100, 33, 73]. 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 [44], consolidation has been performed dynamically, allowing server footprint to expand and shrink as a function of the workload. Tolia et al. [99] use ephemeral VM migration to consolidate workloads an power off under-utilized servers.

Virtual Desktop Infrastructure (VDI), discussed in Chapter 3, 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, VDI 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 encoder hardware. These challenges have led to a slow adoption of VDI [53]. We discuss in Section 9.2.3 how partial VM migration may be used to improve consolidation ratios of VDI.

A recent proposal by Intel [65], places VMs on users' desktop PCs while maintaining VM provenance from a golden image stored on a server. The Intelligent Desktop Virtualization (IDV) proposed here, provides the ease of management benefits of VDI, but it 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 [47], which we discuss in detail in Chapter 3, uses consolidation of idle desktop VMs to reduce energy use. Active VMs run on desktop PCs, and idle VMs run on consolidation servers. Their approach uses full migration of virtual machines, which we have shown to be slow, congest enterprise networks, and does not inherently lead to high consolidation ratios. We have shown in Section 6.3 that migration patterns in offices with 300 users lead to migration latencies that exceed 17 minutes, 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 that are under 5 seconds, migrating less than 10% of the VM's memory state, and delivering consolidations ratios that are high.

9.1.6 Other Uses of Desktop Idle Times

Desktop idle times have been taken advantage of to run tasks that could otherwise impact the experience of the desktop user. IT administrators normally schedule long running maintenance tasks such as virus scans, file indexing and software patching for hours, during which the user is absent. Others use idle times to schedule computational tasks that are parallel in nature and can be distributed across many PCs [31, 92, 76, 108]. SETI@home [31] and Folding@home [92] are two such projects that distribute scientific workloads involving analysis of radio telescope signals in search for extra terrestrial intelligence (SETI@home), and folding of protein molecules (Folding@home).

Running both, administrative batch and distributed computation tasks on a VM that has been consolidated can cause the VM to become active, or it can exhaust the

consolidation server's resources, such as CPU cycles. Recall from Section 5.3.2 that a VM becomes active when it accesses large amounts of state that is resident on the desktop, a condition that causes the desktop to stay awake for a long period. Both, VM activity and server resource exhaustion cause the VM to be migrated back to the desktop. The result is that these applications that take advantage of idle times can continue to do so, and take advantage of desktop resources. However, when no such application is running, partial VM migration enables the desktop enter low power state.

9.2 Virtual Machine Migration

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 [87, 68, 106].

In an effort to reduce migration sizes, Surie et al. [97] propose the use of record and replay of UI interactions to synchronize two VMs over the wide area. They record only a log of user generated UI events and replay them on the outdated copy of the VM at destination. Although this approach can reduce synchronization time, especially in low bandwidth environments, as the authors have found, replay of interactions is insufficient to eliminate divergence of VM state. There are many events that lead to diverging state (timer, interrupts, or 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

Pre-copy live VM migration [44], improved the state of the art by migrating most state while the OS continues to run at the source, and ensuring minimal down times, that are as low as 60 ms. Pre-copy live VM migration is performed iteratively in three stages. In the first, it copies all memory to the destination and tracks any pages that are dirtied by the running VM at the source in the interim. In the second stage, 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 stage 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 stage, which can be large if the running workload is write heavy, Svärd et al. [98] compute per page deltas and compress them before transmission. Wood et al. [107] reduce the number of copy iterations by detecting when an iteration no longer reduces the number of dirty pages, and repeating the iterations only until a local minimum is found. It also uses content-based redundancy elimination and sub-page delta transmission to reduce migration sizes.

Remus [45] and SecondSite [83], take advantage of pre-copy live VM migration to perform rapid replication of VMs and deliver fault tolerance and high availability for critical services. It aggressively checkpoints and replicates the state of an executing VM by migrating only the updated state to the destination. Bickford et al. [37] consider 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 sub-page granularity.

Post-copy live VM migration [60], inverts the migration order. 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, post-copy live VM migration, pages in

the missing page from the source. To do so, post-copy live VM migration implements a network device, to which pages are swapped out at the source, and swapped in from 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 [69] 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, as 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 in which the performance of the clone is significantly degraded.

Bryant et al. [40] use SnowFlock to enable cloud micro-elasticity. Clouds grow and shrink at will in support of the changing workloads. An enabler of cloud micro-elasticity 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 to the desktop. Our work shows that on-demand migration is suitable for reducing energy use of idle desktops. We have shown that the working set of an idle desktop VM is small and consists mostly of memory, and that on-demand migration scales well in enterprise offices, unlike full VM migration. We have built support for on-demand migration in the presence of source hosts that enter low power mode. We have designed an approach to determine when it is appropriate for the desktop to enter sleep state, that is based on empirical study of page request interarrivals. We explored the use of prefetching of pages with spacial locality to increase microsleep durations. We have extended the approach to provide fast desktop resume and suspend cycles via Context Aware-Selective Resume, and to offload page serving duties to a low power page cache.

9.2.3 Other Uses of Desktop Virtualization

There are many virtualization platforms that target the desktop. Many enable users to run guest OSes directly on their desktops [23, 25, 19, 21, 26], while others provide a consolidated desktop infrastructure [101, 81, 43]. The first set is often used to enable users to colocate different operating systems for application or device compatibility [18], to isolate untrusted or unstable software and drivers [70, 54], or to enable desktop mobility [68, 87, 42]. The second set, embodied as Virtual Desktop Infrastructure (VDI), is used to simplify desktop management and deliver enterprise class storage services, such as improved reliability and backup.

Partial desktop VM migration tries maintain compatibility with many of these applications of desktop virtualization. Partial VM migration can work in environments

where users use VMs that are colocated on their desktops. Such setup, requires that our desktop utilities initiate consolidation of more than one VM, as we have implemented so far. It is each call to the consolidation utilities that copies the VM's context to the server and initiates the memory and disk server for the VM, each given a unique port. The consolidation server treats each VM independently, as if originated from different desktops, and initiates independent memory (memtap) and disk clients (cownetdisk) for each VM. Then, to support colocated VMs, we expect a simple change to the migration management daemon, the activityMonitor, so that it makes calls to consolidate and reintegrate more than one VM. When VMs colocated on the same desktop are consolidated, the desktop sees a higher rate of page faults than if only one VM was consolidated. These faults can limit the potential for energy savings if the desktop has to wake up to service each fault. In desktops with low power page caches, however, this is not a problem, as the desktop remains in low power even as faults are serviced.

While the technical details for supporting colocated VMs are simple, it is also necessary to consider whether simultaneous migration of the VMs is necessary. For example, if only one VM encapsulates applications that require network presence, then it may be desirable to consolidate only that VM during idle times, and suspend other VMs when the desktop enters low power.

Compatibility is also maintained with applications of desktop virtualization that enable desktop mobility [68, 87, 42]. These are designed to relocate desktop state to enable subsequent resume in different locations. Instead, our goal is to enable the desktop to save energy whenever the VM becomes idle while running in a given location. During these idle times, the VMs are partially migrated to a nearby consolidation server, and its applications continue to maintain network presence. For example, while a user of Internet Suspend/Resume [68] may use their VM at work during office hours, suspend the VM and resume at home after work, there will be plenty of idle times during work hours. It is in these times partial VM migration can allow the desktop quickly and efficiently to

go into low power.

In environments where desktops are deployed in VDI, partial VM migration may be used to improve consolidation ratios of idle VMs. As discussed in Section 3.2, because desktop VMs require large amounts of memory, and VDI is provisioned to support their peak workloads, consolidation ratios are low, and the costs of running servers high. Idle VMs may be consolidated more aggressively with partial VM migration, allowing idle servers to be powered off or an overcommitment of server resources. Full VM migration may be used infrequently to colocate VMs that have idle periods in common, to ensure that servers become idle. When few VMs remain active in the server, they can be fully migrated out so that the remaining VMs can be consolidated with partial VM migration, and the server powered off. This approach can work as long as full VM migrations are not nearly as frequent as partial migrations (otherwise the network is flooded). To ensure that the server can sleep when its VMs have been partially migrated out, the server may require the presence a low power page cache, which services VM page faults while the server remains in low power (see Chapter 8).

An full exploration of the feasibility and performance of partial migration of VMs on VDI servers, desktops with colocated VMs, and other applications of desktop virtualization is left for future work.

Chapter 10

Conclusions and Future Work

This thesis introduces *fine-grain migration of VM state with long-term residues at end-points.* An important use of this capability is for energy savings through partial 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 1 MiB for disk. It migrates state on-demand and allows the desktop to microsleep when not serving requests. Partial VM migration 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. Our results show that migration latencies of partial VM migration are low, averaging 4 seconds.

Results with our prototypes 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 of 10 to 20 minutes to display management only,

because of their inability to maintain application network presence when the PC is in low power (e.g., default behaviours of Windows 7, Mac OS X 10.6 and guidelines of [16, 15]). Partial VM migration not only turns the monitor off, but also transitions the PC into low power, as 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 five hours. These results also showed that in small to medium 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), while providing 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 10 minutes remained under 16%.

We applied two solutions to address these challenges. The first, Context-Aware Selective Resume, is a software-only solution for legacy desktop PCs, which 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 ten minutes, it delivers energy savings of 24% to 44%, and in intervals of twenty minutes, it produces savings of 31% to 61%.

The second solution we developed, the low power page cache, is intended for future desktop hardware, and embeds a network accessible cache of VM memory pages on the PC hardware. The cache responds to page requests from the consolidation server while the PC remains in low power state. We presented Oasis, a prototype implementation of

the low power cache architecture, that uses the Gumstix computer-on-module platform as the cache. Oasis increases the energy savings of partial VM migration over the baseline, which wakes the PC up whenever a page is needed, two to twenty fold. In idle intervals of ten minutes, Oasis delivers savings between 45% and 50%, and in twenty minutes, between 67% and 73%.

This thesis has demonstrated that partial VM migration can save considerable energy in idle desktops and that both, Context-Aware Selective Resume and the low power page cache, make energy savings in short idle intervals attractive. Combined, our contributions can deliver energy savings of 45% to 91% in idle intervals of ten minutes or more.

10.1 Were our objectives met?

The objective of this thesis has been to provide an energy conservation solution for idle PCs that is practical, scalable and supports applications that maintain always-on network presence. Specifically, the solution needed to: (i), support application transparency; (ii), support always-on application semantics; (iii), provide a user experience similar to that of an always-on PC; and (iv), scale well with the size of enterprise offices. We now revisit these objectives and discuss the extent to which each has been met in this thesis.

Application transparency is necessary for adoption in enterprises, where changes to applications for support of energy conservation is not always possible. Legacy and third party applications are often the hardest to update to support an enterprise's deployment needs. Partial VM migration provides application transparency by operating at the virtual machine layer. Applications run unmodified, and consolidation ensures that all application state, including network connections, is preserved.

Always-on application semantics enable applications with persistent network presence to function while we conserve energy. Support for these applications is critical

because they have been found to cause idle desktops to remain powered. Partial VM migration is designed specifically to accommodate these applications. First, by migrating VMs, it ensures that all desktop applications continue to run during consolidation. And second, network migration ensures that all network flows remain operational, and the VM continues to receive packets destined to itself across hosts.

Low resume latency means that upon returning to their desktops users must not experience long delays when accessing their desktops or endure degraded performance. We believe that, to be practical, these latencies may at most be within the range of typical hardware resume delays from low power states, with which have been familiarized. Our goal has been to deliver resume latencies in the range of 10 seconds, typical of existing hardware. While our average total resume latencies, including both hardware resume and VM migration are close, we found worst case reintegration times, where the user requests the VM to resume immediately after the PC begins its transition to low power, to be high. For our custom desktop described in Section 7.4, average total resume times were 10.44 seconds. For the Dell Studio XPS 7100 desktops of Section 6.2, they were 12.69 seconds. Worst case latencies were 13.44 seconds and 21.07 seconds, for the custom desktop and Dell Studio XPS 7100 desktops, respectively. Worst case resume times can cause user frustration, particularly because they are indicative of an incorrect detection of user idleness. We detail in Section 10.2 several approaches we propose to address this shortcoming.

Scalability of the solution is needed to ensure that it works in offices with hundreds or thousands of PCs. Migration latencies must remain low and energy savings high, with only modest investments in hardware and network infrastructure. Our simulation results have shown that partial VM migration sustains migration latencies under 5 seconds, even as the number of PCs sharing the same GigE network increases to 500. Desktop VMs configured with 4 GiB of memory, have footprints of only 165.63 MiB on the consolidation

server, enabling high consolidation ratios. VM migration causes very little network traffic, averaging 242.23 MiB (including prefetch state) during consolidation and 114.68 MiB during reintegration. It is this low network traffic that ensures that networks are not congested, and migration latencies remain low.

10.2 Future Work

Additional improvements are needed before full time deployment of partial VM migration on 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 trade-offs between the goals of energy conservation and minimal impact on user, as well as any long term effects of the use of our approach on user behaviour; and finally, studies of page level OS behaviour may further improve the performance of the approach.

10.2.1 Resume Latencies

While partial VM migration delivers low migration latencies, as we have already observed, worst case total resume times remain high. For our desktops, total resume times, including hardware resume and VM migration, were 13.44 seconds and 21.07 seconds for the custom PC and the Dell Studio XPS 7100 PC, respectively. The problem remains the slow suspend and resume times of desktop PCs. While Context-Aware Selective Resume has a great effect in reducing suspend and resume cycle times when serving pages, these improvements do not apply to transitions to and from full desktop mode. These are still governed by the OS of the administrative domain which, at suspend time, iterates through all devices to save their pre-suspend state, and at resume time, scans buses to detect hardware changes and re-initializes devices. As we noted previously, however, recent laptops PCs are now able to achieve resume times of around 2 seconds (e.g., the

MacBook Air [11], and the Acer Aspire S3 [6]). These systems can deliver low resume times because the OEMs are able to simplify suspend and resume processes by limiting the customizability of the hardware. Non-customizable hardware makes boot up and resume sequences predictable, reducing the number of detection tests that must be performed by the OS. And second, the host OS or hypervisor may delay initialization of devices that are not immediately visible to the user until after initial UI screens have been displayed (e.g., delay initialization of external storage device until the login screen is presented.)

Additional techniques may be used to reduce reintegration latencies that are visible to the user. First, migration of dirty pages from the server can combine on-demand page migration with batch transfer, in a manner that is similar to post-copy live VM migration [60]. Initially, VM's vCPU context is copied to the desktop, the dirty pages invalidated on the desktop, and the VM allowed to run. Then, dirty pages are transferred in batch mode from the server. When the VM faults on a dirty page that has not been transferred, it preempts the batch transfer to allow the VM to quickly resume execution.

The second approach to reduce reintegration latencies experienced by the user involves implementing an emulated display. The hypervisor can implement the login screen which enables users to enter their authentication credentials while the VM is migrated, to mask reintegration time. The login screen can buffer the user input until the VM is reintegrated, at which point it forwards the input to the VM's OS. Such a strategy can be implemented on the device driver backends that run in the host OS or the hypervisor. The screen is generated in the virtual frame buffer, and can even be a bitmap copy of the screen on the server, and the user input is recorded in the virtual keyboard driver.

10.2.2 Conservation and Usability

While we sought to limit the impact of VM migration and desktop transition on the workflow of the user by ensuring we provide low migration latencies, we are yet to identify

the right balance between the frequency of VM consolidations and the impact on user activity for our system which, unlike widespread commercial systems, is able to engage low power modes while applications run to substantial energy savings, even in short idle periods. More frequent consolidations enable us to take advantage of shorter idle times, potentially even shorter than the 10 minute baseline. However, consolidations that are too frequent are likely to cause dissatisfaction with users, even if each time they wait for only a few seconds. The result is that users and administrators may opt to disable VM consolidation on their systems. Additional research is needed to understand well the trade-offs and identify the right idle periods when consolidation is appropriate. At present, we provide users with a configurable timeout setting, which dictates the length of time that must elapse without user activity before the user's VM is consolidated.

We also expect improved methods to detect user idleness that go beyond UI activity timeouts may the decrease the likelihood of user frustration. We limit false positives in detection of user idleness by allowing the user to cancel any consolidation before it begins. When we consider the user to be inactive, we display a dialog with a timer indicating the number of seconds remaining until the VM is consolidated. Any mouse or keyboard activity during this time aborts the consolidation. Other research has proposed the use of sensors, such as Web cameras to detect user presence [46] or even the user's eye gaze [75], as better mechanisms to gage user engagement with the system. We leave the study of the benefits of such approaches to partial VM migration to future work.

10.2.3 OS Interactions with On-demand Migration

This work has treated VMs and their OSes largely as black-boxes. This approach ensures the generality of our solution, so that it may be used with different operating systems. The downside of treating VMs as black-boxes is that we have no visibility into the OS. Our experiments have revealed results that warrant an investigation of page accesses at the OS level. First, the results presented in Chapter 4 show that even though most

faults occur early in the consolidation, they continue to occur sporadically throughout the consolidation session. What pages cause these faults? Could changes to the OS suppress them? Second, results from Section 5.5 show that while hoarding of previously accessed pages on the consolidation server increases the average microsleep duration of the desktop, its improvements are lower than those for on-demand prefetching of pages with spatial proximity. Why does hoarding not perform as well? What is the nature of pages being missed? Could OS provided hints improve the performance of prefetching? Finally, consolidated execution dirties only a portion of accessed pages. Which pages are these? These are all questions that motivate future studies of page level OS and application behaviour under partial VM migration.

10.3 The Future of the Desktop

As computing form factors continue to evolve, so does the definition of the desktop. Users are gradually doing more of their computing on mobile devices such as tablets and smartphones. These form factors may one day supplant the PC as the predominant desktop form factor. While the devices have characteristics that differ from today's desktop, a confluence of trends make partial VM migration relevant in this new context.

First, the difference in power use between full power and low power modes remains appreciable. As a result, by entering low power, there is an opportunity to save energy. Moreover, because mobile devices rely on batteries that have limited energy storage capacity, inadequate power management can have catastrophic results.

Second, these devices maintain runtime state on the device itself, and the size of that state continues to grow. Consider the Apple iPad tablet [13]. Its initial version released in 2010 contained 256 MiB of DRAM. Subsequent versions released in 2011 and 2012, each doubled DRAM capacity to 512 MiB and 1 GiB, respectively. Similarly, the Samsung Galaxy S [20] line of smartphones started shipping in 2010 with 512 MiB of DRAM. In

subsequent years, they increased DRAM capacity to 1 GiB, and 2 GiB in 2011 and 2012, respectively. All of this DRAM may be used for runtime state. If that runtime state is migrated to allow an array of applications to continue to run when the device enters low power, users will find themselves with the same kinds of problems found with desktop PCs. Migration is slow and can saturate networks, and the results are much worse in cellular networks.

The third trend we are seeing is the adoption of virtualization in mobile computing platforms [32, 95, 37, 24, 14, 22]. Virtualization is the key enabler of partial VM migration and its widespread availability contributes to the success of this approach.

There are challenges in employing partial VM migration on devices that are mobile, however. By their nature, mobile devices can move between networks, and can indeed lose network connectivity and become unreachable. Partial VMs cannot run when unable to reach the home device to service page faults. While IP mobility [2, 93] may help with devices that move between networks, it does little for devices that lose network connectivity. In such cases, it may be necessary to buffer runtime state at infrastructure "middleboxes" that are near the mobile device, in the same manner that middleboxes are used by Dogar et al. [49] to take advantage of bandwidth differences between upstream and downstream components of a link. Middleboxes can be implemented within the basestation of the mobile network to which the device is connected. Faults occurring on the consolidation server can then be serviced from the middlebox.

Bibliography

[1] Advanced configuration and power interface specification. http://www.acpi.info/DOWNLOADS/ACPIspec10b.pdf, Feb 1999.

[2] Mobility support in IPv6. http://www.ietf.org/rfc/rfc3775.txt, Jun 2004.

[3] Product brief: Intel® PRO/1000 GT desktop adapter. http://www.intel.com/content/dam/doc/product-brief/intel-pro-1000-gt-desktop-adapter-datasheet.pdf, 2004.

[4] Broadcom programmer interface specification for the NetXtreme® family of highly integrated media access controllers. http://www.broadcom.com/collateral/pg/57XX-PG105-R.pdf, Jan 2008.

[5] Product brief: Intel® Gigabit CT desktop adapter. http://www.intel.com/content/dam/doc/product-brief/gigabit-ct-desktop-adapter-brief.pdf, 2008-2011.

[6] Acer Aspire S3. http://us.acer.com/ac/en/US/content/series/aspiresseries, Oct 2012.

[7] The Acid3 test. http://acid3.acidtests.org/, Jul 2012.

[8] Facebook chat. http://www.facebook.com/sitetour/chat.php, Jun 2012.

[9] Gmail. http://gmail.com, Jun 2012.

[10] Google docs. http://docs.google.com, Jun 2012.

[11] MacBook Air. http://www.apple.com/macbookair/performance.html, Oct 2012.

[12] SunSpider 0.9.1 JavaScript Benchmark. http://www.webkit.org/perf/sunspider-0.9.1/sunspider-0.9.1/driver.html, Jul 2012.

[13] Apple iPad. http://www.apple.com/ipad/, Apr 2013.

[14] Cellrox mobile virtualization. http://www.cellrox.com/, Apr 2013.

[15] Enegy-efficient computer use. http://energy.gov/energysaver/articles/energy-efficient-computer-use, Jan 2013.

[16] ENERGY STAR® program requirements for computers. http://www.energystar.gov/ia/partners/product_specs/program_reqs/Computers_Program_Requirements.pdf, Jan 2013.

[17] Gumstix® Overo® AirSTORM COM. https://www.gumstix.com/store/product_info.php?products_id=266, Feb 2013.

[18] Install and use Windows XP mode in Windows 7. http://windows.microsoft.com/en-US/windows7/install-and-use-windows-xp-mode-in-windows-7, Apr 2013.

[19] Parallels Desktop 8 For Mac. http://www.vmware.com/products/workstation/, Apr 2013.

[20] Samsung Galaxy. http://www.samsung.com/us/galaxy-s-3-smartphone/, Apr 2013.

[21] VirtualBox. https://www.virtualbox.org/, Apr 2013.

[22] vlogix mobile for mobile virtualization. http://www.redbend.com/en/products-solutions/mobile-virtualization/vlogix-mobile-for-mobile-vitrualization, Apr 2013.

[23] VMware Fusion. http://www.vmware.com/products/fusion/overview.html, Apr 2013.

[24] Vmware mobile virtualization platform. http://www.vmware.com/company/news/releases/mvp.html, Apr 2013.

[25] VMware Workstation. http://www.vmware.com/products/workstation/, Apr 2013.

[26] Windows Virtual PC. http://www.microsoft.com/en-us/download/details.aspx?id=3702, Apr 2013.

[27] Advanced Micro Devices, Inc. AMD I/O virtualization technology (IOMMU) specification. http://support.amd.com/us/Embedded_TechDocs/34434-IOMMU-Rev_1.26_2-11-09.pdf, Feb 2009.

[28] Y. Agarwal, R. Chandra, A. Wolman, P. Bahl, K. Chin, and R. Gupta. Wireless wakeups revisited: Energy management for voip over wi-fi smartphones. In *5th International Conference On Mobile Systems, Applications And Services (MobiSys '07)*, San Juan, Puerto Rico, Jun 2007.

[29] Y. Agarwal, S. Hodges, J. Scott, R. Chandra, P. Bahl, and R. Gupta. Somniloquy: Augmenting network interfaces to reduce PC energy usage. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI '09)*, Boston, MA, Apr 2009.

[30] Y. Agarwal, S. Savage, and R. Gupta. Sleepserver: A software-only approach for reducing the energy consumption of PCs within enterprise environments. In *USENIX Annual Technical Conference (USENIX ATC '10)*, Jun 2010.

[31] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, Nov 2002.

[32] J. Andrus, C. Dall, A. V. Hof, O. Laadan, and J. Nieh. Cells: A virtual mobile smartphone architecture. In *ACM Symposium on Operating System Principles (SOSP '11)*, Caiscais, Portugal, Oct 2011.

[33] M. Badaloo. An examination of server consolidation: trends that can drive efficiencies and help businesses gain a competitive edge. IBM.com, `http://www.ibm.com/systems/optimizeit/pdf/server_consolidation_whitepaper.pdf`, Dec 2006.

[34] R. A. Baratto, S. Potter, G. Su, and J. Nieh. Mobidesk: Mobile virtual desktop computing. In *International Conference on Mobile Computing and Networking (MobiCom '04)*, Philadelphia, PA, USA, Sep–Oct 2004.

[35] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *19th Symposium on Operating Systems Principles (SOSP '03)*, Bolton Landing, NY, USA, Oct 2003.

[36] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *IEEE Computer*, 40(12), Dec 2007.

[37] J. Bickford and R. Cáceres. Towards synchronization of live virtual machines among mobile devices. In *Fourteenth Workshop on Mobile Computing Systems and Applications (HotMobile '13)*, Jekyll Island, Georgia, USA, Feb 2013.

[38] N. Bila, E. de Lara, M. Hiltunen, K. Joshi, H. A. Lagar-Cavilla, and M. Satyanarayanan. The case for energy-oriented partial desktop migration. In *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '10)*, Boston, MA, USA, Jun 2010.

[39] N. Bila, E. de Lara, K. Joshi, H. A. Lagar-Cavilla, M. Hiltunen, and M. Satyanarayanan. Jettison: Efficient idle desktop consolidation with partial VM migration. In *ACM European Conference on Computer Systems (EuroSys '12)*, Bern, Switzerland, Apr 2012.

[40] R. Bryant, A. Tumanov, O. Irzak, A. Scannell, K. Joshi, M. Hiltunen, H. A. Lagar-Cavilla, and E. de Lara. Kaleidoscope: Cloud micro-elasticity via VM state coloring. In *ACM European Conference on Computer Systems (EuroSys '11)*, Salzburg, Austria, Apr 2011.

[41] California Energy Commission. California commercial end-use, cec-400-2006-005. Energy.ca.gov, `http://www.energy.ca.gov/2006publications/CEC-400-2006-005/CEC-400-2006-005.PDF`, Mar 2006.

[42] P. M. Chen and B. D. Noble. When virtual is better than real. In *8th IEEE Workshop on Hot Topics in Operating Systems (HotOS '01)*, Elmau/Upper Bavaria, Germany, May 2001.

[43] Citrix Systems, Inc. Citrix VDI-in-a-box. `http://www.citrix.com/site/resources/dynamic/salesdocs/VDIinaBox_Datasheet.pdf`, 2011.

[44] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *2nd Conference on Symposium on Networked Systems Design and Implementation (NSDI '05)*, Boston, MA, May 2005.

[45] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: high availability via asynchronous virtual machine replication. In *5th USENIX Symposium on Networked Systems Design and Implementation (NSDI '08)*, San Francisco, CA, USA, Apr 2008.

[46] A. B. Dalton and C. S. Ellis. Sensing user intention and context for energy management. In *9th Workshop on Hot Topics in Operating Systems (HotOS IX)*, Lihue, HI, USA, May 2003.

[47] T. Das, P. Padala, V. Padmanabhan, R. Ramjee, and K. G. Shin. LiteGreen: Saving energy in networked desktops using virtualization. In *USENIX Annual Technical Conference (USENIX ATC '10)*, Boston, MA, USA, Jun 2010.

[48] T. Dierks and E. Rescorla. The TLS protocol: Version 1.2. `https://tools.ietf.org/html/rfc5246`, Aug 2008.

[49] F. R. Dogar, P. Steenkiste, and K. Papagiannaki. Catnap: Exploiting high bandwidth wireless interfaces to save energy for mobile devices. In *International Conference on Mobile Systems, Applications and Services (MobiSys '10)*, Jun 2010.

[50] Y. Dong, Z. Yu, and G. Rose. Sr-iov networking in xen: Architecture, design and implementation. In *First Workshop on I/O Virtualization (WIOV '08)*, San Diego, CA, USA, Dec 2008.

[51] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ACM International Symposium on Computer Architecture (ISCA '07)*, San Diego, CA, USA, Jun 2007.

[52] J. Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *17th ACM Symposium on Operating System Principles (SOSP '99)*, Kiawah Island, SC, USA, Dec 1999.

[53] K. Fograrty. The year of the virtual desktop fails to materialize–again. http://www.cio.com/article/691303/The_Year_of_the_Virtual_Desktop_Fails_to_Materialize_Again, Oct 2011.

[54] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson. Safe hardware access with the Xen virtual machine monitor. In *1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS '04)*, Boston, MA, USA, Oct 2004.

[55] J. Glanz. Power pollution and the internet: Data centers waste vast amounts of energy, belying industry image. http://www.nytimes.com/2012/09/23/technology/data-centers-waste-vast-amounts-of-energy-belying-industry-image.html?pagewanted=all.

[56] R. P. Goldberg. Architectural principles for virtual computer systems. Technical report, Harvard University, Feb 1973.

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

[58] C. Gunaratne, K. Christensen, and B. Nordman. 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):297–310, Sep 2005.

[59] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd, and Toshiba Corporation. Advanced configuration and power interface specification. ACPI.info, http://www.acpi.info/DOWNLOADS/ACPIspec40.pdf.

[60] M. R. Hines and K. Gopalan. 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)*, Washington, DC, Mar 2009.

[61] Intel Corporation. Enhanced intel® SpeedStep® technology for intel® Pentium® m processor. Intel.com, `ftp://download.intel.com/design/network/papers/30117401.pdf`, Mar 2004.

[62] Intel Corporation. Intel® Centrino® mobile technology wake on wireless lan feature: Technical brief. Intel.com, `http://www.intel.com/network/connectivity/resources/doc_library/tech_brief/wowlan_tech_brief.pdf`, 2006.

[63] Intel Corporation. Intel® virtualization technology for directed I/O: Architecture specification. `http://goo.gl/VscXC`, Feb 2011.

[64] Intel Corporation. PCI-SIG SR-IOV primer: An introduction to SR-IOV technology. `http://www.intel.com/content/dam/doc/application-note/pci-sig-sr-iov-primer-sr-iov-technology-paper.pdf`, Jan 2011.

[65] Intel Corporation. Vision paper: Intelligent desktop virtualization. `http://www.intel.com/content/dam/doc/white-paper/intelligent-desktop-virtualization-overview-paper.pdf`, Oct 2011.

[66] M. Jimeno, K. Christensen, and B. Nordman. A network connection proxy to enable hosts to sleep and save energy. In *27th IEEE International Performance Computing and Communications Conference*, Austin, TX, USA, Dec 2008.

[67] A. Kadav and M. M. Swift. Live migration of direct-access devices. In *First Workshop on I/O Virtualization (WIOV '08)*, San Diego, CA, USA, Dec 2008.

[68] M. Kozuch and M. Satyanarayanan. Internet suspend/resume. In *4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '02)*, Callicoon, NY, USA, Jun 2002.

[69] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan. SnowFlock: rapid virtual machine cloning for cloud computing. In *4th ACM European Conference on Computer Systems (EuroSys '09)*, Nuremberg, Germany, Mar 2009.

[70] J. LeVasseur, V. Uhlig, J. Stoess, and S. Götz. Unmodied device driver reuse and improved system dependability via virtual machines. In *6th Symposium on Operating Systems Design and Implementation (OSDI '04)*, San Francisco, CA, USA, Dec 2004.

[71] Lieberman Software Corporation. White paper: Wake on LAN technology. Liebsoft.com, `http://www.liebsoft.com/pdfs/Wake_On_LAN.pdf`, Jun 2006.

[72] F. X. Lin, Z. Wang, R. LiKamWa, , and L. Zhong. Reflex: Using low-power processors in smartphones without knowing them. In *17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '12)*, London, UK, Mar 2012.

[73] M. R. Marty and M. D. Hill. Virtual hierarchies to support server consolidation. In *34th International Symposium on Computer Architecture (ISCA '07)*, San Diego, CA, USA, 2007.

[74] D. Meisner, B. T. Gold, and T. F. Wenisch. PowerNap: Eliminating server idle power. In *14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '09)*, Washington, DC, USA, Mar 2009.

[75] V. G. Moshnyaga and E. Morikawa. LCD display energy reduction by user monitoring. In *International Conference on Computer Design (ICCD '05)*, San Jose, CA, USA, Oct 2005.

[76] V. K. Naik, S. Sivasubramanian, D. Bantz, and S. Krishnan. Harmony: A desktop grid for delivering enterprise computations. In *4th International Workshop on Grid Computing (GRID '03)*, Phoenix, AZ, USA, Nov 2003.

[77] S. Nedevschi, J. Chandrashekar, J. Liu, B. Nordman, S. Ratnasamy, and N. Taf. 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, Apr 2009.

[78] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig. Intel® virtualization technology: Hardware support for efficient processor virtualization. *Intel® Technology Journal*, 10(3), Aug 2006.

[79] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Agile application-aware adaptation for mobility. In *16th ACM Symposium on Operating System Principles (SOSP '97)*, Saint-Malo, France, Oct 1997.

[80] M. F. Oberhumer. LZO real-time data compression library. `http://www.oberhumer.com/opensource/lzo/`, Aug 2011.

[81] Oracle Corporation. Oracle virtual desktop infrastructure. `http://www.oracle.com/us/virtual-desktop-infrastructure-ds-067844.pdf`, Oct 2012.

[82] PC Power & Cooling. Power supplies: How much power do you need? `http://www.pcpower.com/technology/power_usage/`, Oct 2012.

[83] S. Rajagopalan, B. Cully, R. OConnor, and A. Warfield. Secondsite: Disaster tolerance as a service. In *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '12)*, London, UK, Mar 2012.

[84] J. Reich, M. Goraczko, A. Kansal, and J. Padhye. Sleepless in seattle no longer. In *2010 USENIX Annual Technical Conference*, Boston, MA, USA, Jun 2010.

[85] T. Richardson, Q. Stafford-Fraser, K. Wood, and A. Hopper. Virtual Network Computing. *IEEE Internet Computing*, 2(1), Jan/Feb. 1998.

[86] M. Ronsse, K. D. Bosschere, M. Christiaens, J. C. de Kergommeaux, and D. Kranzlmüller. Record/replay for nondeterministic program executions. *Communications of the ACM*, 46(9):62–67, Sep 2003.

[87] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum. Optimizing the migration of virtual computers. In *5th Symposium on Operating Systems Design and Implementation (OSDI '02)*, Boston, MA, USA, Dec 2002.

[88] B. K. Schmidt, M. S. Lam, and J. D. Northcutt. The interactive performance of slim: a stateless, thin-client architecture. In *17th ACM Symposium on Operating Systems Principles (SOSP '99)*, Kiawah Island, SC, USA, Dec 1999.

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

[90] S. Sen, J. R. Lorch, R. Hughes, C. G. J. Suarez, B. Zill, W. Cordeiro, and J. Padhye. Dont lose sleep over availability: The GreenUp decentralized wakeup service. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI '12)*, San Jose, CA, USA, Apr 2012.

[91] E. Shih, P. Bahl, and M. J. Sinclair. Wake on wireless: An event driven energy saving strategy for battery operated devices. In *8th Annual International Confer-*

ence on Mobile Computing and Networking (MOBICOM '02), Atlanta, GA, USA, Sep 2002.

[92] M. Shirts and V. S. Pande. Screen savers of the world, unite! *Science*, 290(5498):1903–1904, Dec 2000.

[93] A. C. Snoeren and H. Balakrishnan. An end-to-end approach to host mobility. In *6th ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 00)*, Boston, MA, Aug 2000.

[94] J. Sorber, N. Banerjee, M. D. Corner, and S. Rollins. Turducken: Hierarchical power management for mobile devices. In *3rd International Conference on Mobile Systems, Applications and Services (Mobisys 2005)*, Seattle, WA, USA, Jun 2005.

[95] P. Stanley-Marbell and L. Iftode. Scylla: A smart virtual machine for mobile embedded systems. In *3rd IEEE Workshop on Mobile Computing Systems and Applications*, Monterey, CA, USA, Dec 2000.

[96] E. L. Sueur and G. Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Workshop on Power Aware Computing and Systems (HotPower '10)*, Vancouver, BC, Canada, Oct 2010.

[97] A. Surie, H. A. Lagar-Cavilla, E. de Lara, and M. Satyanarayanan. Low-bandwidth VM migration via opportunistic replay. In *9th Workshop on Mobile Computing Systems and Applications (HotMobile '08)*, Napa Valley, California, Feb 2008.

[98] P. Svärd, B. Hudzia, and J. Tordsson. 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)*, Newport Beach, CA, USA, Mar 2011.

[99] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and X. Zhu. Delivering energy proportionality with non energy-proportional systems – optimizing the ensemble. In *Workshop on Power Aware Computing and Systems (HotPower '08)*, San Diego, CA, USA, Dec 2008.

[100] A. Tucker and D. Comay. Solaris zones: Operating system support for server consolidation. In *3rd Virtual Machine Research and Technology Symposium (VM '04), Works-In-Progress*, San Jose, CA, USA, may 2004.

[101] VMware, Inc. Virtual desktop infrastructure. `http://www.vmware.com/pdf/virtual_desktop_infrastructure_wp.pdf`, Feb 2013.

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

[103] A. Warfield, S. Hand, K. Fraser, and T. Deegan. Facilitating the development of soft devices. In *USENIX Annual Technical Conference (USENIX ATC '05)*, Anaheim, CA, USA, Jun 2005.

[104] C. A. Webber, J. A. Robertson, M. C. McWhinney, R. E. Brown, M. J. Pinckard, and J. F. Busch. After-hours power status of office equipment in the USA. *Energy*, 31(14):2487–2502, Nov 2006.

[105] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Symposium on Operating Systems Design and Implementation (OSDI '94)*, Monterey, CA, USA, Nov 1994.

[106] A. Whitaker, R. S. Cox, M. Shaw, and S. D. Gribble. Constructing services with interposable virtual hardware. In *1st Symposium on Networked Systems Design and Implementation (NSDI '04)*, San Francisco, CA, USA, Mar 2004.

[107] T. Wood, P. Shenoy, K. Ramakrishnan, and J. V. der Merwe. 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)*, Newport Beach, CA, USA, Mar 2011.

[108] D. Wright. Cheap cycles from the desktop to the dedicated cluster: Combining opportunistic and dedicated scheduling with Condor. In *Conference on Linux Clusters: the HPC Revolution*, Champain Urbana, IL, USA, Jun 2001.

[109] E. Wright, N. Bila, E. de Lara, and A. Goel. Effective use of sleep states with context-aware selective resume. Unpublished manuscript, Mar 2013.

[110] E. J. Wright, E. de Lara, and A. Goel. Vision: The case for context-aware selective resume. In *International Workshop on Mobile Cloud Computing and Services (MCS '11)*, Washington, DC, USA, Jun 2011.

[111] L. Xia, J. Lange, and P. Dinda. Towards virtual passthrough I/O on commodity devices. In *First Workshop on I/O Virtualization (WIOV '08)*, San Diego, CA, USA, Dec 2008.

[112] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. Ecosystem: Managing energy as a first class operating system resource. In *10th nternational Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '02)*, San Jose, CA, USA, Oct 2002.

[113] E. Zhai, G. D. Cummings, and Y. Dong. Live migration with pass-through device for linux VM. In *First Workshop on I/O Virtualization (WIOV '08)*, San Diego, CA, USA, Dec 2008.