Jettison: Efficient Idle Desktop Consolidation with Partial VM Migration

Nilton Bila[†], Eyal de Lara[†], Kaustubh Joshi[×], H. Andrés Lagar-Cavilla [*],
Matti Hiltunen[×] and Mahadev Satyanarayanan[‡]

[†]University of Toronto, [×]AT&T Labs Research, [*]GridCentric Inc., [‡]Carnegie Mellon University

Abstract

Idle desktop systems are frequently left powered, often because of applications that maintain network presence or to enable potential remote access. Unfortunately, an idle PC consumes up to 60% of its peak power. 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 bulk 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 for each VM. This paper introduces *Partial VM Migration,* a technique that transparently migrates only the working set of an idle VM. Jettison, our partial VM migration prototype, can deliver 85% to 104% of the energy savings of full VM migration, while using less than 10% as much network resources, and providing migration latencies that are two to three orders of magnitude smaller.

Categories and Subject Descriptors D.4.7 [*Operating Systems*]: Organization and Design —Distributed systems

General Terms Design, Experimentation, Measurement

Keywords Desktop Virtualization, Cloud Computing, Energy

1. Introduction

Modern offices are crowded with personal computers. Previous studies have shown that office computers are left continuously running, even when idle [8, 19, 25]. These idle times have been shown to add up to close to 12 hours per day, excluding off times [19]. Unfortunately an idle PC consumes close to 60% of the power of a fully utilized system. While modern computers support low power ACPI states [3], 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, VoIP clients, and remote desktop access and administration utilities, maintain network presence even when the PC is idle. Remotely waking up the PC on-demand via Wake-on-LAN [5] and similar mechanisms has been shown not to work, as frequent gratuitous network traffic in enterprise environments prevents the PC from sleeping [16, 19].

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 [13]. 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 paper introduces *Partial VM Migration*, a technique 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, typically less than 10% of memory 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

[*] Andrés participated in this work while employed at AT&T Labs Research.

EuroSys'12, April 10–13, 2012, Bern, Switzerland.

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 desktop based partial VM migration prototype. Our experience with a Jettison deployment shows that significant energy savings are achievable without negative impact on user experience. Within an hour of inactivity, desktops were able to save up to 78% of energy. 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 migrations 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 current prototype targets VMs with local storage on the desktop, 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) [6] that simplify desktop management by centralizing it, while supporting local execution. Whereas these approaches concentrate on managing and backing up the VM persistent state, partial VM migration is mainly concerned about migration of run state.

This paper makes five contributions: (i), it shows that the working set of an idle VM 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; and (v), it presents a complete architecture used to consolidate idle partial VMs and reintegrate them back to their desktop, when active.

The remainder of the paper is organized as follows. Section 2 motivates the need for partial VM migration, by demonstrating that migrating VMs in full is inadequate for energy oriented consolidation of idle desktop VMs. Section 3 introduces partial VM migration as a technique to reduce energy use of idle desktops. Section 4 describes Jettison, our implementation of partial VM migration. Section 5 presents results from a our deployment of Jettison and, by using simulations, Section 6 extends these results for large office environments and evaluates the scalability of our approach. In Section 7, we discuss the sensitivity of our results, the implications of using network accessible shared storage, the implications of our approach to hardware and software reliability, and how our approach fits within the context of virtualization solutions in the market place. Section 8 discusses related work, and Section 9 concludes the paper.

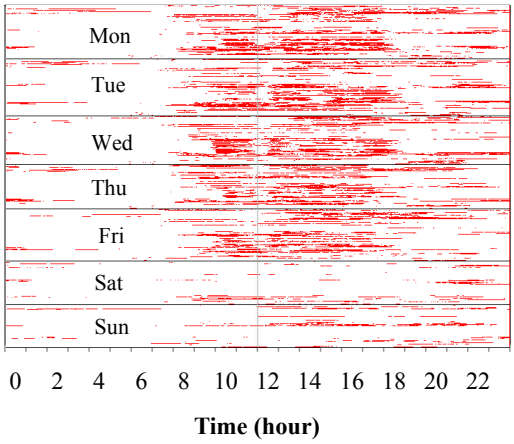


Figure 1. Desktop/laptop usage and idle periods

2. Motivation

Our research confirms prior studies (e.g., [19]) that desktops are powered up but idle significant portions of time. Figure 1 illustrates activity traces collected from desktop and laptop machines in a research lab for 500 person days. Each line represents one machine day, with a dot indicating that the machine is in use, and a white space that the machine is idle. The figure shows that, in addition to the overnight and lunch time idle periods, there are numerous significant idle periods that are opportunities for energy savings. However, to take advantage of these periods, the following requirements must be met:

1. Quick resume: To ensure user acceptance, the desktop must be restored in a few seconds when the user resumes their work.
2. Conservation of the network resources: With 100s or 1000s of users in one organization, frequent desktop migrations put significant strain on the network.
3. Cost effective: The extra capital cost incurred by the consolidation servers must be small enough that it does not exceed the savings from reduced energy use.

While prior work (e.g., [13]) proposes desktop consolidation using existing VM technology (live migration [12] and ballooning [23]), we find such techniques fall short in

meeting these requirements. We performed experiments to quantify the performance of existing VM technologies on enterprise class hardware[1]. In these experiments, the desktop VM was warmed up through a script that loaded a number of documents and web pages and then idled. After one minute of idleness, the VM was migrated to the consolidation server through a dedicated network switch. The VM used shared storage, provided through Redhat's network block device GNBD, which ensured disk availability upon migration. 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.

Live migration [12] provides an obvious baseline. In our experimental setup, the average latency of live migration is 38.59 seconds, and the average network bandwidth consumed is 4.27 GiB. However, the migration of one VM at a time does not give the full picture of the user experience. Specifically, a "boot storm" occurs when multiple users start work or resume work at the same time or in close proximity. As expected, live migrating multiple VMs out of a single consolidation server concurrently degrades linearly: 4 VMs takes an average of 137 seconds, while 8 take 253 seconds. Staggering the resume times helps, but even with 20 second pause between resumes of 8 VMs still results in average latency of 115.62 seconds. Not only is live migration unable to ensure quick resumes, but it will introduce significant strain on the network (number of migrations times the average VM size) as we will demonstrate in Section 6.

Ballooning is a technique that allows the memory footprint of a VM to be shrunk when desired and thus, using it before consolidations could alleviate some of the disadvantages of live migration. While ballooning is able to shrink the VM footprint, this happens at a considerable expense of time and I/O. In our experiments with Xen's ballooning implementation, our idle VM's footprint reached its saturation point at 423 MiB (swapping turned on to avoid killing any processes), but ballooning took an average of 328.44 seconds to reach this saturation point. In the process, it 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. While the ballooned VMs can be easily migrated—the VMs with 423 MiB footprints were migrated on the 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 network bandwidth. Even if we use local disks instead of shared storage in order to reduce network usage, the ballooning latency is still prohibitive for our goals 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.

[1] Xen VMs with 4 GiB of memory and 12 GiB of disk; Dell Poweredge R610, 24 GiB of RAM, 8 2.3 GHz Xeon cores, Fusion-MPT SAS drives, and a Broadcom NetXtreme II gigabit NIC.

3. Partial VM Migration

Partial VM migration allows user applications to maintain network presence while the desktop sleeps. It transfers the execution of an idle VM to a consolidation server, and fetches the VM's memory and disk state on-demand. Partial VM migration differs from post-copy VM migration [17] in that the migration is not executed to completion. Instead most of the VM state remains as a residual on the user's desktop in anticipation of a reverse migration. 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 performance to the user. When on the server, only the working set required for idle execution is available there.

By migrating only the limited working set that is accessed while the desktop remains idle, partial VM migration allows for 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 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 desktop microsleeps only when it will save energy. In the next sections, we determine strategies that answer the following questions:

1. When should a desktop microsleep?
2. How can prefetching be used to optimize microsleep opportunities?

We first describe how we migrate idle VM working set on-demand, and describe a deployment of our partial VM migration prototype from which we collected memory and disk access traces for the analysis.

3.1 Working Set Migration

When consolidating a VM from the desktop to the server, partial VM migration transfers memory state only as the VM

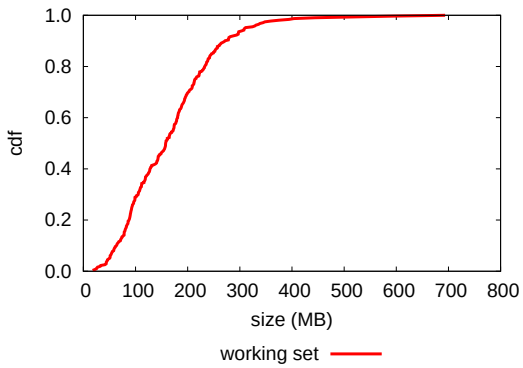


Figure 2. Distribution of working set sizes of idle Linux VMs with 4 GiB of memory.

requires it for its execution. During consolidation, partial VM migration transfers only a VM descriptor, which contains VM configuration (e.g. device listing, memory limit), virtual CPU register state, and page table pages set up by the VM's kernel. The descriptor is used to set up page tables and start the partial VM's execution. As the VM tries to access its pages, it causes faults that are handled by an external process. This process migrates each faulting page from the desktop, on-demand. We refer to these page faults accesses as *remote faults*. Once migrated, future accesses to a page do not result in remote faults. To improve performance, accesses caused by allocations that overwrite whole pages are handled differently. These faults are handled locally by the VM's kernel with no remote fault ensuing.

For VMs with local disk images rather than shared network storage, disk state is also migrated from the desktop on-demand. At consolidation time, only device configuration is transmitted as part of the descriptor. Each first time disk block access results in its migration from the desktop. First block accesses that result in whole block writes are also handled locally by the disk driver.

3.2 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 users over a seven week period. To identify idle periods, we monitored UI activity and if the user was found to be inactive for at least 15 seconds, a 5 second dialog was displayed, warning of an impending VM consolidation. If no response was given, we considered the user idle and the VM was consolidated. Each VM's 12 GiB disk image was stored locally on the desktop.

Figure 2 shows the distribution of working set sizes of the three user VMs over 313 idle periods. These desktop VMs were each allocated 4 GiB of memory and were used as general purpose Linux desktop systems with applications such as Web browsers, document processors and instant messengers. Idle periods occurred throughout the day.

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 allocated 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.

3.3 When to Microsleep

A desktop system experiences increased power use during transitions to sleep and wake-up. A microsleep will only save energy if it lasts long enough to compensate for the transient energy rise required to enter sleep and wakeup the system to serve a remote fault.

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

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

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

The energy use of an idle system that microsleeps is:

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

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

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

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

Plugging in our desktop profile from Table 1, we find that, for our systems, $t_b = 32.22$ seconds. Thus, our desktop should microsleep only when there is an expectation that no remote faults will arrive in at least 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 of 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.

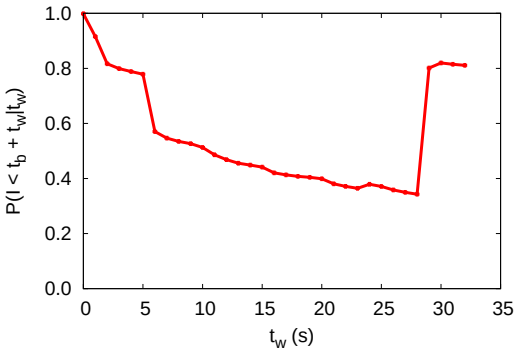


Figure 3. Conditional probability that the next remote fault will arrive in less than 32.22 s as a function of the wait time.

More formally, $p(I < t_b + t_w|t_w)$ is the probability of inter-arrival I being energy inefficient.

Figure 3 plots the conditional probability that the next remote fault will arrive in less than 32.22 s based on remote fault inter-arrival times for the prototype deployment of the previous section. The figure 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 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:

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

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 4, and it is clear that the energy waste minimizing t_w is 6 seconds.

3.4 State 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.

As discussed in Section 3.1, most state transfer, hence remote faults, is caused by memory accesses (more than 99%). 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.

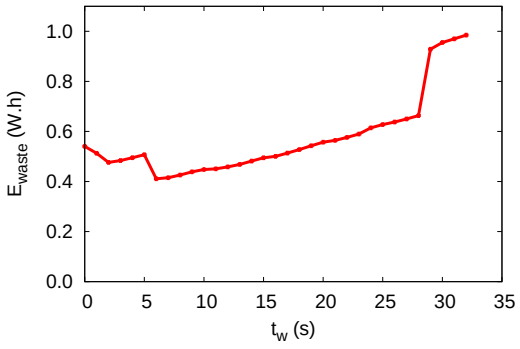


Figure 4. Expected energy waste as a function of sleep timeout (t_w).

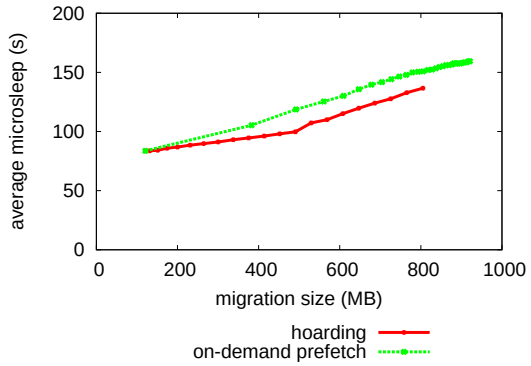


Figure 5. Effect of hoarding and on-demand prefetching on microsleep lengths.

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 unlikely to be used.

Figure 5 compares the performance of hoarding and on-demand prefetch per MiB of migrated state. The figure, uses simulation results based on page access traces from a user VM from our deployment that is consolidated 58 times. The primary performance metric is the average length of time of each microsleep. The objective of prefetch is to reduce the number remote faults that interrupt microsleeps. Figure 6 shows the energy savings for the same VM, normalized over the energy the desktop uses during those idle periods when left powered. We compute energy use of the

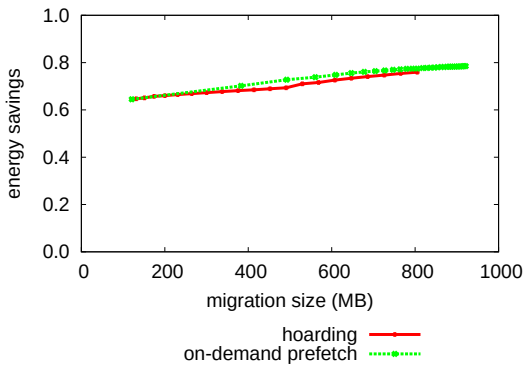


Figure 6. Effect of hoarding and on-demand prefetching on energy savings.

desktop by aggregating energy used over each interval the desktop would be idle, suspending, sleeping and resuming under each prefetch strategy, using Equation 2. For any given migration size, on-demand prefetch increases average microsleep durations faster than hoarding, which result in slightly better energy savings. Migration size is composed of state migrated by the prefetch strategy, and state fetched due to a prefetch cache miss. Our results indicate that on-demand prefetch is better at predicting contiguous sequence of requests. Hoarding can more often miss requests in the middle of a cluster, resulting in extra desktop wake-ups.

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 state migration over the duration of consolidation, as state is 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 it's effectiveness in reducing remote faults to be no better than with same VM residuals.

For the remainder of the paper 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.

4. Jettison

Our partial consolidation prototype is Jettison. It is implemented on top of the Xen 3.4 hypervisor [10]. Xen is a type 1 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 patches to the domU kernel. Our implementation currently supports paravirtualized guests, and we plan to extend it to also support fully virtualized VMs. In the discussion below, we explain where changes are needed for fully virtualized VMs.

Jettison runs the following components on the dom0 of the desktop system. An *activityMonitor* daemon, responsible for detecting user activity, initiating both consolidation and user-triggered reintegrations, and suspending the desktop system to memory sleep (S3). At present, activityMonitor monitors keyboard and mouse and, after an user configured period of inactivity, provides an on-screen warning for another pre-configured period, before consolidating the VM. Our defaults are 15 and 5 seconds for the inactivity and warning periods, respectively. Only while the VM is consolidated, the desktop runs a *memserv* and a *diskserv* processes, responsible for serving memory pages and disk blocks, respectively, to the server over TCP. memserv, maps all of the consolidated VM's frames with read-only access.

On the consolidation server's dom0, Jettison runs a *remoteWakeup* daemon and two additional processes for each domU. remoteWakeup wakes up a sleeping desktop via Wake-On-LAN [5] whenever remote state is required. A *memtap* process monitors VM page faults, notifies remoteWakeup and issues page requests to memserv running on the desktop and, on response, updates the VM's page frame. memtap maps its VM's frames with write access so it can perform direct updates.

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 [24] 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.

When to consolidate and reintegrate? The decision to consolidate rests on three conditions: (i) user idleness - the user is not actively engaging the VM, (ii) server capacity - the server has sufficient resources to accommodate the VM, (iii) VM idleness - the VM can execute on the server with sufficient autonomy from the desktop, such that the desktop can sleep and save energy.

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.

On the server we care primarily about memory availability. When deciding to consolidate an idle VM, Jettison checks that the server has enough memory to accommodate it. Because a partial VM requires only a fraction of its nominal memory, we must estimate the size of its working set before migration, and ensure that the server can accommodate it. Initially, we estimate the working set size from the observed sizes in our deployment. This estimate can then be adjusted based on the VM's previous history on the server. The median working set size of our VMs was 157.87 MiB. 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. If not, the server evicts the VM back to its desktop.

Consolidating a VM to the server is only feasible when it allows its desktop to sleep long enough to conserve energy. This requires that the VM is accessing minimal amount of disk and memory state resident on the desktop. While the VM runs on the desktop, we monitor it'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. Xen can initially make 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.

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 the user becomes active, the VM becomes active and requires a large amount of state from the desktop, or the server's capacity is exceeded.

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 diskserv and a memserv processes 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 small 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. 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 paravirtual-

ized 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 hosts free page heap by increasing the granularity of requested memory chunks to 2MiB at a time, while still replacing ghost MFNs one at a time.

While we have not implemented dynamic memory allocation and remote fault handling for fully virtualized guests, known as hardware assisted VMs (HVMs), we plan to use Xen's built-in populate-on-demand (PoD) mechanism to do both. PoD maps PFNs to MFNs on-demand for HVMs, by faulting on first access to each page. This mechanism is used to boot HVMs with lower memory commitments than their maximum reservation, $maximum$. 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.

5. Prototype 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?

5.1 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.

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 1. Power profile of Dell Studio XPS 7100 Desktop.

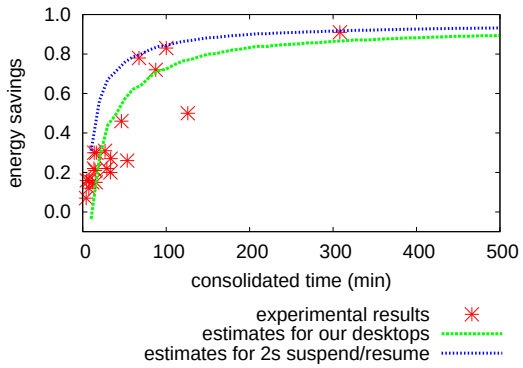


Figure 7. Desktop energy savings.

Each VM was configured with 4 GiB of memory and 12 GiB of disk. The VMs ran Linux with the GNOME desktop configured with Mozilla Firefox and Thunderbird, OpenOffice.org, Pidgin IM client, OpenSSH, among others. 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^{TM}II X4 945 processor and 6 GiB of RAM. Table 1 presents the desktop's power profile obtained with a GW Instek GPM-8212 power meter. These numbers are comparable to those of other published systems [8, 13]. The server was a Sun Fire X2250 system with two quad-core 3 GHz Intel Xeon® CPUs and 16 GiB of memory. It's 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.

5.2 Energy Savings

Figure 7 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

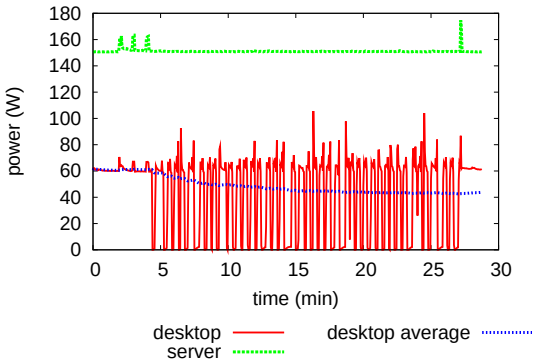


Figure 8. Power usage of a desktop and server during partial migration. The figure shows a reduction in the average energy use over time.

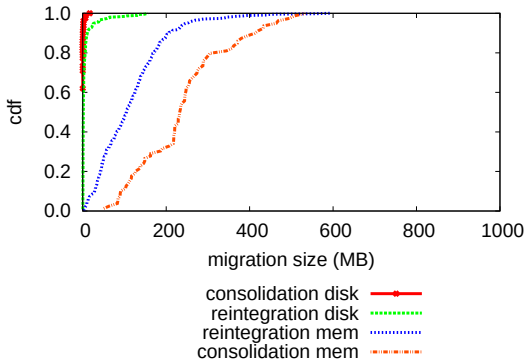


Figure 9. Distribution of migration sizes for VMs with 4 GiB of memory. Plot order matches legend top to bottom.

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 3.4. 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%.

In too short idle times, the energy expended by the desktop going to sleep and waking up is not outweighed 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). We note, however, that recent laptops have demonstrated short resume times, such as nearly 2 seconds in both the Macbook Air [2] and Acer Aspire S3 [1]. We argue that with approaches such as Context-Aware Selective Resume [26] that initialize only the necessary devices on wake-up, desktops can achieve fast suspends and resumes that are comparable to those of optimized laptops. The estimates for a desktop with 2-second suspend and resume in Figure 7 shows that, faster transitions lead to higher savings in short idle times. Intervals under 10 minutes would see savings that are closer to 30%.

Figure 8 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%. 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.

5.3 Network Load

Figure 9 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 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).

5.4 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 1.

5.5 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. 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 "boot 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 screen-saver timer. We deployed the tracker for 4 months at an industrial research lab on 22 researchers' primary work Macs including both desktops and laptops. The machines had user-controlled software environments - there were no corporate lockdowns in place. We collected 2086 person day traces from which a sample of 500 are shown in Figure 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 boot 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.

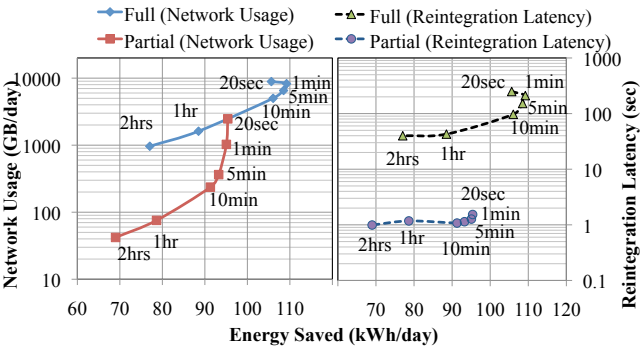


Figure 10. Energy savings in kW-h per day vs. reintegration latency and network utilization for 100 desktops and varying idle timeout values.

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 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 9. 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 7 that estimates the savings as a function of consolidation time for our desktops.

Is Partial VM Migration a Real Improvement? Section 5 suggests that when compared to full migration, partial migration significantly improves the network load and user-perceived reintegration latency at the expense of reduced energy savings. The question then arises whether full migration can be made competitive simply by increasing the

idle timeout to migrate less aggressively, thus reducing network load and improving reintegration latencies, but reducing sleep opportunities and energy savings. Figure 10 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 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 11(a) 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 boot 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 will be intolerable for users. Das et al. [13] propose using a remote desktop solution to provide immediate reintegration access to users to mask long reintegration latencies of full migration. However, remote desktop access has many limitations, such as the 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 11(b) 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 11(c) 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 [2]. 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: 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 5 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.[3] 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.

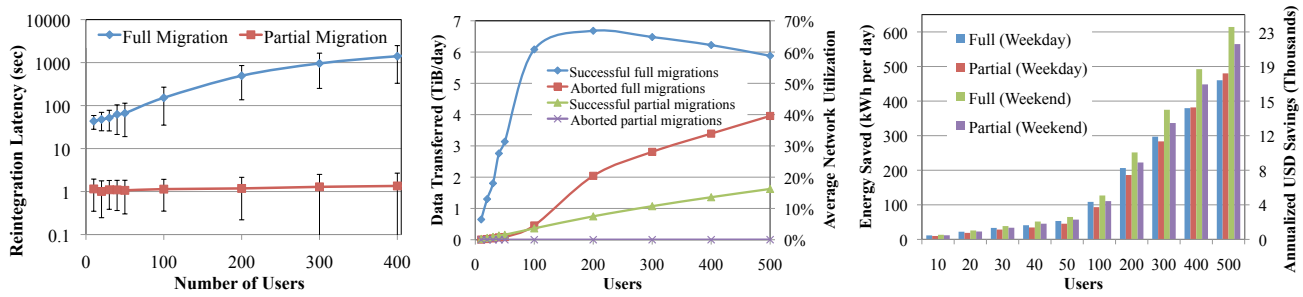
7. Discussion

We discuss next the sensitivity of our results, how our approach fits in the contexts of shared storage and virtualization infrastructures, and the implications of our approach to hardware reliability and software behaviour.

Sensitivity of Results. Our experimental use of Jettison was limited to a Linux desktop based research environment and our results demonstrate the benefits of partial VM migration in this type of environment. While we cannot speculate about the system's behaviour under a different OS or different office scenarios, we hope that the applications we use, many of them available across platforms, exhibit sim-

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

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



(a) Reintegration latency as desktops increase. The error bars show the standard deviations.

(b) Network load. Aborted migrations occur when a new migrations overrides an ongoing one.

(c) Total energy savings in kWh and USD.

Figure 11. Performance of partial migration and full migration as the number of desktops grow.

ilar behaviours in other OSes. We expect to support other guest OSes after implementing the HVM support discussed in Section 4. In our current prototype, the only function that required domU kernel modification was the optional fetch avoidance for whole page allocations described in Section 3.1. On average, this optimization reduced the fraction of the working set fetched remotely by 9.06 MiB from 165.63 MiB to 156.57 MiB. Similarly, our study did not quantify the energy savings in a laptop computer, however, given their similarity to desktop energy profiles, with more than an order of magnitude gap between idle and sleep power, we expect the savings to be comparable to our desktop results. For example, Agarwal et al. [8] found idle and sleep power of three laptop models to be 16.0 W and 0.74 W, 27.4 W and 1.15 W, and 29.7 W and 0.55 W, respectively.

Storage Placement. As described in Section 1, partial VM migration is data placement agnostic. Disk state can be placed either in network servers or locally on the desktops. The benefit of partial VM migration is in reducing migration of run state and, as we have shown in Section 3 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.

The semantics of failure of a consolidated VM on storage consistency is similar under both scenarios, 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 disk state that was written on the server and that may otherwise be useful can be lost. 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, the dirty slice may still be recovered, though this is something we have not implemented. Presently, we have implemented the server-side disk writes buffering, which enables desktop recovery from checkpoint, only for the local disk driver. Adding a similar function to shared storage drivers is left for future work.

Virtualization Context. Virtual Desktop Infrastructure (VDI) has emerged in the past decade as a means to simplify desktop management. It consolidates desktop VMs permanently on shared servers and provides remote access to desktop clients. Full time conversion to VDI can help reducing idle energy use, by powering off idle client devices. However, the pace of adoption of VDI remains slow [15], and full fledged desktops continue to outsell thin clients used in VDI. These thick clients will remain in use for years to come. Our approach provides the energy savings benefits of virtual machine consolidation without incurring costs of full VDI deployment. Our infrastructure requirements are modest because servers need only host idle VMs. In addition, partial VM migration provides support for clients that leverage local hardware resources, such as 3D acceleration.

Partial VM migration can be used with the Intelligent Desktop Virtualization (IDV) model [6] to simplify desktop management. IDV manages desktops centrally while executing them locally on desktops. Partial VM migration VMs can be based off a single golden OS image that resides on a shared server, with user state placed on separate image layers. Like VDI, this approach allows administrators to perform administrative tasks such as software upgrades on a single master image.

Reliability Implications. Our use of microsleeps leads to two concerns. First, frequent power state transitions may lead to reduced life span of hardware components. And second, slow wake up times reduce responsiveness of applications during remote faults, and for networked applications this may cause unintended side-effects on connections.

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 is not needed. Indeed, most desktop wake ups require only access to CPU, memory and network card.

This problem can be mitigated with the Context-Aware Selective Resume approach proposed by Wright et al. [26]. That paper demonstrates that the majority of resume time is spent on OS and BIOS re-initialization, and shows that an approach that bypasses most devices and, in fact, does not re-initialize the entire OS on wake-ups but only the components necessary to, for example read memory and access the network, may reduce resume and suspend times by up to 87%.

In terms of software reliability, the desktop applications we used were able to content 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.

8. Related Work

Although VM consolidation is one of the oldest ideas in cloud computing, previous work has focused on coarse-grain migration of VM memory state. Once the decision to migrate a VM has been made, the migration is executed to completion. Live migration [12] allows execution to begin at the destination before state is fully transferred from the source, but the transfer is completed in the background. After completion, no residual VM state is retained on the source machine in anticipation of a future reverse migration. This stateless model makes sense in the context of large public clouds, where all machines are viewed as a common pool. In our use case, however, a user is typically associated with a specific desktop. Preserving migrated state in anticipation of a reverse migration is likely to have high payoff and, indeed, our work shows that it does.

Enterprise energy savings through desktop consolidation in a private cloud has been previously explored in the context of LiteGreen [13]. However, the consolidation technique used there is the coarse-grain migration approach described in the previous paragraph. While LiteGreen shows significant energy savings, we expect many advantages with fine-grain migration. As discussed, these advantages include the ability to target shorter periods of idleness, a smaller private cloud, and greater tolerance of unpredictable user behavior.

More broadly, there has been extensive previous work on understanding the potential for energy savings from desktop computing infrastructure. Modern computers ship with built-in mechanisms to reduce the power usage of idle systems by entering low power sleep states [3] . However, recent studies have found that users are reluctant to put their idle systems in low power states. Nedevschi et al. [19] find that desktop systems remain powered but idle for an average of 12 hours daily. Webber et al. [25] find that 60% of corporate desktops remain powered overnight. It is conjectured that people refuse to put their system to sleep either for the off-chance they may require remote access, run background applications (IM, e-mail), among other uses [8], or because many idle periods are short, often interspersed with active periods [13]. While there have been approaches to support remote access [4, 5, 7, 21], they do not support always-on application semantics. Support for always-on applications have been proposed [8, 9, 16, 19, 22] either through remote proxies or specialized hardware. These approaches require developers to re-engineer most applications to support bimodal operation, transferring control and state between the full-fledged application and its proxied instance. Alternatively, thin clients [20] allow users to run low power clients, which waste little power when idle. However, thin clients remain unpopular due to poor interactive performance, lacking crispness in response and local hardware acceleration. Also, thin clients require fully-provisioned cloud infrastructure that is capable of supporting their peak collective load.

Exploiting short opportunities for sleep while a host is waiting for work is also explored in Catnap [14]. 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.

In the data center, SnowFlock [18] has demonstrated the benefits of on-demand state migration to instantiate stateful worker replicas of cloud VMs. Our work has shown that on-demand migration is suitable for reducing energy use of idle desktops with our support of host sleep, the working set of an idle desktop VM is small and consists mostly of memory, on-demand migration scales better than full VM migration in shared office networks, and provides a means to reintegrate a partial VM state back to its origin.

In previous workshop paper [11], we proposed Partial VM migration. That work, however, only estimated the migration size and used simulation to estimate energy savings. In contrast, this paper presents a working implementation and an evaluation with a real deployment.

9. Conclusion

This work introduces *fine-grain migration of VM state with long-term residues at endpoints.* 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 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. We show that a desktop can achieve energy savings of 78% in an hour of consolidation and up to 91% in longer periods, while maintaining latencies of about 4 s. We show 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.

Acknowledgments

We thank Vidur Taneja for his early assistance exploring on-demand prefetching of pages. We thank our shepherd Orran Krieger and the anonymous EuroSys'12 reviewers for helping improve the quality of the final version of this paper. This research was funded in part by the National Science and Engineering Research Council of Canada (NSERC) under grant number 261545-3. Satyanarayanan was supported in this research by the National Science Foundation (NSF) under grant number CNS-0833882.

References

[1] Acer Aspire S3. `http://us.acer.com/ac/en/US/content/series/aspiresseries`.

[2] Macbook Air. `http://www.apple.com/macbookair/performance.html`.

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

[4] Intel® Centrino® mobile technology wake on wireless LAN (WoWLAN) feature: Technical brief. `http://www.intel.com/network/connectivity/resources/doc_library/tech_brief/wowlan_tech_brief.pdf`, 2006.

[5] Wake on LAN technology. `http://www.liebsoft.com/pdfs/Wake_On_LAN.pdf`, Jun 2006.

[6] Vision paper: Intelligent desktop virtualization. `http://goo.gl/MNmEa`, Oct 2011.

[7] 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 *MobiSys '07*, Jun 2007.

[8] Y. Agarwal, S. Hodges, J. Scott, R. Chandra, P. Bahl, and R. Gupta. Somniloquy: Augmenting network interfaces to reduce pc energy usage. In *NSDI '09*, Apr 2009.

[9] Y. Agarwal, S. Savage, and R. Gupta. Sleepserver: A software-only approach for reducing the energy consumption of pcs within enterprise environments. In *USENIX ATC '10*, Jun 2010.

[10] 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 *SOSP '03*, Oct 2003.

[11] N. Bila, E. de Lara, M. Hiltunen, H. A. L.-C. Kaustubh Joshi, and M. Satyanarayanan. The Case for Energy-Oriented Partial Desktop Migration. In *HotCloud '10*, Jun 2010.

[12] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *NSDI'05*, May 2005.

[13] T. Das, P. Padala, V. N. Padmanabhan, R. Ramjee, and K. G. Shin. LiteGreen: Saving energy in networked desktops using virtualization. In *2010 USENIX ATC*, Jun 2010.

[14] F. R. Dogar, P. Steenkiste, and K. Papagiannaki. Catnap: Exploiting high bandwidth wireless interfaces to save energy for mobile devices. In *MobiSys 2010*, Jun 2010.

[15] 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.

[16] 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. *IJNM*, 15(5):297–310, Sep 2005.

[17] M. R. Hines, U. Deshpande, and K. Gopalan. Post-copy live migration of virtual machines. In *VEE 2009*, Mar 2009.

[18] 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 *EuroSys '09*, Mar 2009.

[19] 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 *NSDI '09*, Apr 2009.

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

[21] E. Shih, P. Bahl, and M. J. Sinclair. Wake on wireless: An event driven energy saving strategy for battery operated devices. In *MobiCom 2002*, Sep 2002.

[22] J. Sorber, N. Banerjee, M. D. Corner, and S. Rollins. Turducken: Hierarchical power management for mobile devices. In *MobiSys '05*, Jun 2005.

[23] C. A. Waldspurger. Memory Resource Management in VMWare ESX Server. In *OSDI '02*, Dec 2002.

[24] A. Warfield, S. Hand, K. Fraser, and T. Deegan. Facilitating the development of soft devices. In *USENIX ATC '05*, Jun 2005.

[25] 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.

[26] E. J. Wright, E. de Lara, and A. Goel. Vision: The case for context-aware selective resume. In *MCS 2011*, Jun 2011.