LiteGreen: Saving Energy in Networked Desktops Using Virtualization

Tathagata Das
tathadas@microsoft.com
Microsoft Research India

Pradeep Padala[*]
ppadala@docomolabs-usa.com
DOCOMO USA Labs

Venkata N. Padmanabhan
padmanab@microsoft.com
Microsoft Research India

Ramachandran Ramjee
ramjee@microsoft.com
Microsoft Research India

Kang G. Shin
kgshin@eecs.umich.edu
The University of Michigan

Abstract

To reduce energy wastage by idle desktop computers in enterprise environments, the typical approach is to put a computer to sleep during long idle periods (e.g., overnight), with a proxy employed to reduce user disruption by maintaining the computer's network presence at some minimal level. However, the Achilles' heel of the proxy-based approach is the inherent trade-off between the functionality of maintaining network presence and the complexity of application-specific customization.

We present *LiteGreen*, a system to save desktop energy by virtualizing the user's desktop computing environment as a virtual machine (VM) and then migrating it between the user's physical desktop machine and a VM server, depending on whether the desktop computing environment is being actively used or is idle. Thus, the user's desktop environment is "always on", maintaining its network presence fully even when the user's physical desktop machine is switched off and thereby saving energy. This seamless operation allows LiteGreen to save energy during short idle periods as well (e.g., coffee breaks), which is shown to be significant according to our analysis of over 65,000 hours of data gathered from 120 desktop machines. We have prototyped LiteGreen on the Microsoft Hyper-V hypervisor. Our findings from a small-scale deployment comprising over 3200 user-hours of the system as well as from laboratory experiments and simulation analysis are very promising, with energy savings of 72-74% with LiteGreen compared to 32% with existing Windows and manual power management.

1 Introduction

The energy consumed by the burgeoning computing infrastructure worldwide has recently drawn significant attention. While the focus of energy management has been on the data-center setting [20, 29, 32], attention has also been directed recently to the significant amounts of energy consumed by desktop computers in homes and enterprises [17, 31]. A recent U.S. study [33] estimates that PCs and their monitors consume about 100 TWh/year, constituting 3% of the annual electricity consumed in the U.S. Of this, 65 TWh/year is consumed by PCs in enterprises, which constitutes 5% of the commercial building electricity consumption in the U.S. Moreover, market projections suggest that PCs will continue to be the dominant desktop computing platform, with over 125 million units shipping each year from 2009 through 2013 [15].

The usual approach to reducing PC energy wastage is to put computers to sleep when they are idle. However, the presence of the user makes this particularly challenging in a desktop computing environment. Users care about preserving long-running network connections (e.g., login sessions, IM presence, file sharing), background computation (e.g., syncing and automatic filing of new emails), and keeping their machine reachable even while it is idle. Putting a desktop PC to sleep is likely to cause disruption (e.g., broken connections), thereby having a negative impact on the user, who might then choose to disable the energy savings mechanism altogether.

To reduce user disruption while still allowing machines to sleep, one approach has been to have a *proxy* on the network for a machine that is asleep [33]. However, this approach suffers from an inherent tradeoff between functionality and complexity because of the need for application-specific customization.

In this paper, we present *LiteGreen*, a system to save desktop energy by employing a novel approach to minimizing user disruption and avoiding the complexity of application-specific customization. The basic idea is to virtualize the user's desktop computing environment, by encapsulating it in a virtual machine (VM), and then migrating it between the user's physical desktop machine and a VM server, depending on whether the desktop computing environment is actively used or idle. When the desktop becomes idle, say when the user steps away for several minutes (e.g., for a coffee break), the desktop VM is migrated to the VM server and the physical desktop machine is put to sleep. When the desktop becomes active again (e.g., when the user returns), the desktop VM is migrated back to the physical desktop machine. Thus, even when it has been migrated to the VM server, the user's desktop environment remains alive (i.e., it is "always on"), so ongoing network connections and other activity (e.g., background downloads) are *not* disturbed, regardless of the application involved.

[*]The author was an intern at MSR India during the course of this work.

The "always on" feature of LiteGreen allows energy savings whenever the opportunity arises, without having to worry about disrupting the user. Besides long idle periods (e.g., nights and weekends), energy can also be saved by putting the physical desktop computer to sleep even during short idle periods, such as when a user goes to a meeting or steps out for coffee. Indeed, our measurements indicate that the potential energy savings from exploiting short idle periods are significant (Section 3).

While the virtualization-based approach allows keeping the desktop environment "always on", two key challenges need to be addressed for it to be useful for saving energy on desktop computers. First, how do we provide a normal (undisrupted) desktop experience to users, masking the effect of VMs and their migration? Second, how do we decide when and which VMs to migrate to/from the server in order to maximize energy savings while minimizing disruption to users?

To address the first challenge, LiteGreen uses the *live migration* feature supported by modern hypervisors [21] coupled with the idea of always presenting the desktop environment through a level of indirection (Section 4). Thus, whether the VM is at the server or desktop, users always access their desktop VM through a remote desktop (RD) session. So, in a typical scenario, when a user returns to their machine that has been put to sleep, the machine is woken up from sleep and the user is able to immediately access their desktop environment (whose state is fully up-to-date, because it has been "always on") through an RD connection to the desktop VM running on the VM server. Subsequently, the desktop VM is migrated back to the user's physical desktop machine without the user even noticing.

To address the second challenge, LiteGreen uses an energy-saving algorithm that runs on the server and carefully balances migrations based on two continuously-updated lists: 1) VMs in the *mandatory to push* list must be migrated to the desktop machine to minimize user disruption, and 2) VMs in the *eligible to pull* list may be migrated to server for energy savings, subject to server capacity constraints (Section 5).

We have prototyped LiteGreen on the Microsoft Hyper-V hypervisor (Section 6). We have a small-scale deployment running on the desktop machines of ten users, comprising three administrative staff and seven researchers, including three authors of this paper. A demonstration video of LiteGreen is available at [4]. Separately, we have conducted laboratory experiments using both the Hyper-V and Xen hypervisors to evaluate various aspects of LiteGreen. We have also developed a simulator to analyze the data we gathered and to understand the finer aspects of our algorithms.

We have analyzed (a) over 65,000 user-hours of data gathered by us from 120 desktop computers at Microsoft Research India (MSRI), and (b) 3200 user-hours of data from a deployment of our prototype on ten user desktops over a span of 28 days. Based on this analysis, LiteGreen is able to put desktop machines to sleep for 86-88% of the time, resulting in an estimated energy savings of 72-74%. In comparison, through a combination of manual user action and the automatic Windows power management, desktop machines are put to sleep for 35% of time, delivering estimated energy savings of only 32%.

The main contributions of this paper are as follows:

1. A novel system that leverages virtualization to consolidate idle desktops on a VM server, thereby saving energy, while avoiding user disruption.
2. Automated mechanisms to drive the migration of the desktop computing environment between the physical desktop machines and the VM server.
3. A prototype implementation and the evaluation of LiteGreen through a small-scale deployment on the desktops of ten users, spaning 3200 user-hours over 28 days, yielding energy savings of 74%.
4. Trace-driven analysis of over 65,000 user-hours of resource usage data gathered from 120 desktops, yielding energy savings of 72%, with short idle periods (< 3 hours) contributing 20% or more.

2 Problem Background and Related Work

In this section, we provide some background on the problem setting and discuss related work.

2.1 PC Energy Consumption

Researchers have measured and characterized the energy consumed by desktop computers [17]. The typical desktop PC consumes 80-110 W when active and 60-80 W when idle, excluding the monitor, which adds another 35-80 W. The relatively small difference between active and idle modes is significant and arises because the processor itself only accounts for a small portion of the total energy. In view of this, multiple S ("sleep") states have been defined as part of the ACPI standard [13]. In particular, the S3 state ("standby") suspends the machine's state to RAM, thereby cutting energy consumption to 2-3 W. S3 has the advantage of being much quicker to transition in and out of than S4 ("hibernate"), which involves suspending the machine's state to disk.

2.2 Proxy-based Approach

As discussed above, the only way of cutting down the energy consumed by a PC is to put it to sleep. However, when a PC it put to sleep, it loses its network presence, resulting in disruption of ongoing connections (e.g., remote-login or file-download sessions) and the machine even becoming inaccessible over the network.

The resulting disruption has been recognized as a key reason why users are often reluctant to put their machines to sleep [17]. Researchers have found that roughly 60% of office desktop PCs are left on continuously [33].

The general approach to allowing a PC to sleep while maintaining some network presence is to have a network proxy operate on its behalf while it is asleep [33]. The functionality of the proxy could span a wide range:

- **WoL Proxy:** The simplest proxy allows the machine to be woken up using the *Wake-on-LAN* (WoL) mechanism [12] supported by most Ethernet NICs. To be able to send the "magic" WoL packet, the proxy must be on the same subnet as the target machine and needs to know the MAC address of the machine. Typically, machine wakeup is initiated manually.

- **Protocol Proxy:** A more sophisticated proxy performs automatic wakeup, triggered by a filtered subset of the incoming traffic [31, 34]. The filters could be configured based on user input and also the list of network ports that the target machine was listening on before it went to sleep. Other traffic is either responded to by the proxy itself without waking up the target machine (e.g., ARP for the target machine) or ignored (e.g., ARP for other hosts).

- **Application Proxy:** An even more sophisticated proxy incorporates application-specific stubs that allow it to engage in network communication on behalf of applications running on the machine that is now asleep [31]. Such a proxy could even be integrated into an augmented NIC [17].

Enhanced functionality of a proxy comes at the cost of greater complexity, for instance, the need to create stubs for each application that the user wishes to keep alive. LiteGreen sidesteps this complexity by keeping the entire desktop computing environment alive, by consolidating it on the server along with other idle desktops. On the flip side, however, LiteGreen is more heavyweight than the proxy approach, as we discuss in Section 9.2.

2.3 Saving Energy through Consolidation

Consolidation to save energy has been employed in other computing settings—data centers and thin clients.

In the data-center setting, server consolidation is used to approximate energy proportionality by migrating computation, typically using virtualization, from several lightly-loaded servers onto fewer servers, and then turning off the servers that are freed up [20, 37, 38]. Doing so saves not only the energy consumed directly by the servers but also the significant amount of energy consumed indirectly for cooling [29, 30].

Thin client based computing, an idea that is making a reappearance [23, 11] despite failures in the past, represents an extreme form of consolidation, with all of the computing resources being centralized. While the cost, management, and energy savings might make the model attractive in some environments, there remain questions regarding the up-front hardware investment needed to migrate to thin clients. Also, thin clients represent a trade-off and may not be suitable in settings where power users want the flexibility of a PC or insulation from even transient dips in performance due to consolidation. Indeed, market projections suggest that PCs will continue to be the dominant desktop computing platform, with over 125 million units shipping each year from 2009 through 2013 [15], and with thin clients replacing only 15% of PCs by 2014 [14]. Thus, there will continue to be a sizeable and growing installed base of PCs for the foreseeable future, possibly as part of mixed environments comprising both PCs and thin clients, so addressing the problem of energy consumed by desktop PCs remains important.

While LiteGreen's use of consolidation is inspired by the above work, a key difference arises from the presence of users in a desktop computing environment. Unlike in a data center setting, where machines tend to run server workloads and hence are substitutable to a large extent, a desktop machine is a user's *personal* computer. Users expect to have access to *their* computing environment. Furthermore, unlike in a thin client setting, users expect to have good interactive performance and the flexibility of attaching specialized hardware and peripherals (e.g., a high-end graphics card). Progress on virtualizing high-end hardware, such as GPUs [24, 28], facilitates LiteGreen's approach of running the desktop in a VM.

Central to the design of LiteGreen is preserving this PC model and minimizing both user disruption and new hardware cost, by only consolidating idle desktops.

2.4 Virtualization and Live Migration

A key enabler of consolidation is virtualization. Several hypervisors are available commercially [2, 5, 8]. These leverage the hardware support that modern processors include for virtualization [3, 1].

Virtualization has simplified the task of moving computation from one physical machine to another [40] compared to process migration [36]. Efficient live migration over a high-speed LAN is performed by iteratively copying memory pages while the VM continues execution, before finally pausing the VM briefly (for as short as 60 ms [21]) to copy the remaining pages and resume execution on the destination machine. Live migration has been extended to wide-area networks as well [27].

2.5 Page Sharing and Memory Ballooning

Consolidation of multiple VMs on the same physical server can put pressure on the server's memory resources. *Page sharing* is a technique to decrease the memory footprint of VMs by sharing pages that are in common across multiple VMs [39]. Recent work [26] has advanced the state of the art to also include sub-page level sharing, yielding memory savings of up to 90% with homogeneous VMs and up to 65% otherwise.

Even with page sharing, memory can become a bottleneck depending on the number of VMs that are consolidated on the server. *Memory ballooning* is a technique to dynamically shrink or grow the memory available to a VM with minimal overhead relative to statically provisioning the VM with the desired amount of memory [39].

2.6 Virtualization in LiteGreen Prototype

For our LiteGreen prototype, we use the Microsoft Hyper-V hypervisor. While this is a server hypervisor, the ten users in our deployment were able to use it without difficulty for desktop computing. Since Hyper-V currently does not support page sharing or memory ballooning, we conducted a separate set of experiments with the Xen hypervisor to evaluate memory ballooning. Finally, since Hyper-V only supports live migration with shared storage, we set up a shared storage server connected to the same GigE switch as the desktop machines and the server (see Section 9 for a discussion of shared storage).

3 Motivation Based on Measurement

To provide concrete motivation for our work beyond the prior work discussed above, we conducted a measurement study on the usage of PCs. Our study was set in the MSR India lab during the summer of 2009, at which time the lab's population peaked at around 130 users. Of these, 120 users at the peak volunteered to run our measurement tool, which gathered information on the PC resource usage (in terms of the CPU, network, disk, and memory) and also monitored user interaction (UI). In view of the sensitivity involved in monitoring keyboard activity on the volunteers' machines, we only monitored mouse activity to detect UI.

We have collected over 65,000 hours worth of data from these users. We placed the data gathered from each machine into 1-minute buckets, each of which was then annotated with the level of resource usage and whether there was UI activity. We classify a machine as being *idle* (as opposed to being *active*) during a 1-minute bucket using one of the two policies discussed later in Section 5.2: the *default* policy, which only looks for the absence of UI activity in the last 10 minutes, and a more *conservative* policy, which additionally checks whether the CPU usage was below 10%.

Based on this data, we seek to answer the following questions:

Q1. How (under)utilized are desktop PCs?

To help answer this question, Figure 1a plots the distribution of CPU usage and UI activity, binned into 1-minute buckets and aggregated across all of the PCs in our study. To allow plotting both CPU usage and UI activity in the same graph, we adopt the convention of treating the presence of UI activity in a bucket as 100% CPU usage. The "CPU only" curve in the figure shows that CPU usage is low, remaining under 10% for 90% of the time. The "CPU + UI" curve shows that UI activity is present, on average, only in 10% of the 1-minute buckets, or about 2.4 hours in a day. However, since even an active user might have 1-minute buckets with no UI activity (e.g., they might just be reading from the screen), the total UI activity is very likely larger than 10%.[1]

While both CPU usage and UI activity are low, it still does not mean that the PC can be simply put to sleep, as we discuss below.

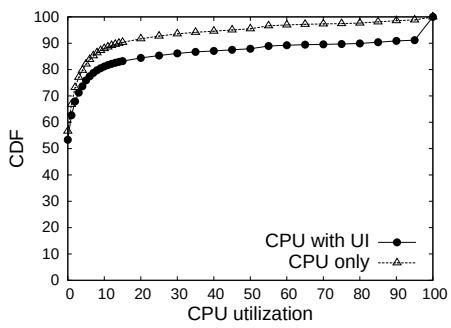
Q2. How are the idle periods distributed?

Given that there is much idleness in PCs, the next question is how the idle periods are distributed. We define an idle period as a contiguous sequence of 1-minute buckets, each of which is classified as being idle. The conventional wisdom is that idle periods are long, e.g., overnight periods and weekends. Figure 1c shows the distribution of idle periods based on the default (UI only) and conservative (UI and CPU usage) definitions of idleness noted above. Each data point shows the aggregate idle time (shown on the y axis on a log scale) spent in idle periods of the corresponding length (shown on the x axis). The x axis extends to 72 hours, or 3 days, which encompasses idle periods stretching over an entire weekend.

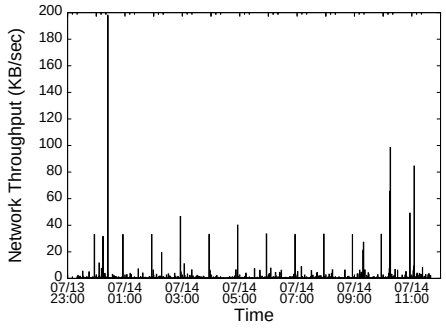
The default curve shows distinctive peaks at around 15 hours (overnight periods) and 63 hours (weekends). It also shows a peak for short idle periods, under about 3 hours in length. In the conservative curve, the peak at the short idle periods dominates by far. The overnight and weekend peaks are no longer distinctive since, based on the conservative definition of idleness, these long periods tend to be interrupted, and hence broken up, by intervening bursts of background CPU activity.

Figure 1d shows that with the default definition of idleness, idle periods shorter than 3 hours add up to about 20% of the total duration of idle periods longer than 3 hours. With the conservative policy, the short idle

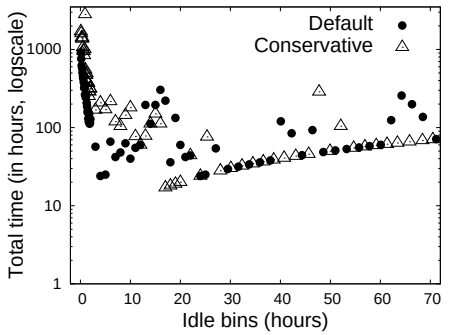
[1] It is possible that we may have missed periods when there was keyboard activity but no mouse activity. However, we ran a test with a small set of 3 volunteers, for whom we monitored keyboard activity as well as mouse activity, and found it rare to have instances, where there was keyboard activity but no mouse activity in the following 10 minutes.



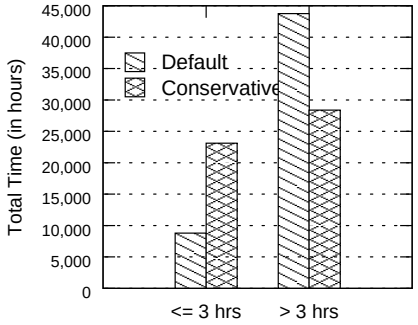
(a) Distribution of CPU utilization



(b) Network activity during the night on one idle desktop machine



(c) Distribution of idle periods



(d) Comparison of aggregate duration of short and long idle periods

Figure 1: Analysis of PC usage data at MSR India

Category	Example Applications	Sleep	Proxy-based On-demand Wakeup	LiteGreen
Incoming requests	incoming RDP	fails	works but with initial delay/timeout	works
	file share			works but requires disk
Idle connections	outgoing RDP	broken connection		works
	IM	user shown as offline		user shown as away
Background tasks	large file download	download stalled ⇒ delay		works
	software patching (e.g., Windows update)	patch download delay ⇒ larger window of vulnerability		patches downloaded but need disk for application

Table 1: Impact of various energy saving strategies on applications

periods add up to over 80% of the total duration of the long idle periods. Thus, the short idle periods, which correspond to lunch breaks, meetings, etc., during a work day, represent a significant opportunity for energy savings over and above the savings from the long idle periods considered in prior work.

Q3. Why not just sleep during idle periods?

Even when the machine is mostly idle (i.e., has low CPU utilization), it could be engaged in network activity, as depicted in Figure 1b. A closer look at this machine (with the owner's permission) revealed that the processes that showed sporadic activity were (a) `InoRT.exe`, a virus scanner, (b) `DfrgNtfs.exe`, a disk defragmenter, (c) `TrustedInstaller.exe`, which checks for Windows software updates, and (d) `Svchost.exe`, which encapsulates miscellaneous services. Putting the machine to sleep would delay or disrupt these tasks, possibly incoveniencing the user.

Privacy considerations prevented us, in general, from gathering detailed information such as process names, which would have revealed the identities of the applications running on a user's machine. Hence, we use indirect means to understand how sleep might be disruptive.

Through informal conversations at MSR India, we compiled a list of typical applications that users run. Table 1 categorizes these and reports on the impact of sleep on these applications. We find that the applications suffer disruption to varying degrees. In some cases, sleep causes a hard failure, e.g., a broken connection. In other cases, it causes a soft failure. For example, if a user steps out for a meeting and their (idle) machine goes to sleep, IM might show them, somewhat misleadingly, as being "offline" when "away" would be more appropriate.

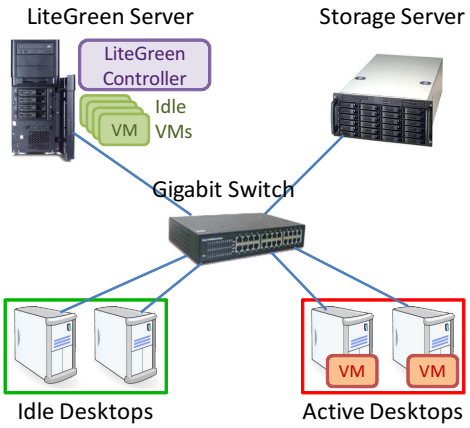


Figure 2: **LiteGreen architecture**: Desktops are in active (switched on) or idle (sleep) state. Server hosts idle desktops running in VMs

The ability to do on-demand wakeup, as provided by a proxy, helps when there is new inbound communication, e.g., an incoming remote desktop (RDP) connection. Such communication would work, although it might suffer from an initial delay or timeout owing to the time it takes to wake up from sleep. However, with applications where there is an ongoing connection, the proxy approach is unable to prevent disruption. In fact, the only way of avoiding disruption is to not go to sleep, which means giving up on energy savings.

Avoiding disruption requires that the applications continue to run and maintain their network presence even while the machine is (mostly) idle. Doing so while still saving energy motivates a solution such as LiteGreen. In some cases, however, LiteGreen would require access to the local disk, either immediately (e.g., file sharing) or eventually (e.g., software patching). While our current implementation does not migrate the disk, we believe that such migration is feasible, as discussed in Section 9.

In summary, we make two key observations from our analysis. First, desktop PCs are often idle, and there is significant opportunity to exploit short idle periods. Second, it is important to maintain network presence even during the idle periods to avoid user disruption.

4 System Architecture

Figure 2 shows the high-level architecture of LiteGreen. The desktop computing infrastructure is augmented with a VM server and a shared storage node. In general, there could be more than one VM server and/or shared storage node. All of these elements are connected via a high-speed LAN such as Gigabit Ethernet.

Each desktop machine as well as the server run a hypervisor. The hypervisor on the desktop machine hosts a VM in which the client OS runs. This VM is migrated to the server when the user is not active and the desktop is put to sleep. When the user returns, the desktop is woken up and the VM is "live migrated" back to the desktop. To insulate the user from such migrations, the desktop hypervisor also runs a remote desktop (RD) client [7], which is used by the user to connect to, and remain connected to, their VM, regardless of where it is running. Although our current prototype does not leverage it, the advent of GPU virtualization [24, 28] allows improving the user experience by bypassing remote desktop when the VM is running locally on the desktop machine.

The user's desktop VM uses, in lieu of a local disk, the shared storage node, which is also shared with the server. This aspect of the architecture arises from the limitations of live migration in hypervisors currently in production and can be done away with once live migration with local VHDs is supported (Section 9).

The hypervisor on the server hosts the guest VMs that have been migrated to it from (idle) desktop machines. The server also includes a *controller*, which is the brain of LiteGreen. The controller receives periodic updates from *stubs* on the desktop hypervisors on the level of user and computing activity on the desktops. The controller also tracks resource usage on the server. Using all of this information, the controller orchestrates the migration of VMs to the server and back to the desktop machines, and manages the allocation of resources on the server. We have chosen a centralized design for the controller because it is simple, efficient, and also enables optimal migration decisions to be made based on full knowledge (e.g., the bin-packing optimization noted in Section 5.3).

5 Design

Having provided an overview of the architecture, we now detail the design of LiteGreen. The design of LiteGreen has to deal with two somewhat conflicting goals: maximizing energy savings from putting machines to sleep while minimizing disruption to users. When faced with a choice, LiteGreen errs on the side of being conservative, i.e., avoiding user disruption even if it means reduced energy savings.

The operation of LiteGreen can be described in terms of a control loop effected by the controller based on local information at the server as well as information reported by the desktop stubs. We discuss the individual elements before putting together the whole control loop.

5.1 Which VMs to Migrate?

The controller maintains two lists of VMs:

- *Eligible for Pull:* list of (idle) VMs that currently reside on the desktop machines but could be migrated (i.e., "pulled") to the server, thereby saving energy without user disruption.

- *Mandatory to Push:* list of (now active) VMs that had previously been migrated to the server but must now be migrated (i.e., "pushed") back to the desktop machines at the earliest to minimize user disruption.

In general, the classification of a VM as active or idle is made based on both UI activity initiated by the user and computing activity, as discussed next.

5.2 Determining If Idle or Active

The presence of any UI activity initiated by the user, through the mouse or the keyboard (e.g., mouse movement, mouse clicks, key presses), in the recent past (*actvityWindow*, set to 10 minutes by default) is taken as an indicator that the machine is active. Even though the load imposed on the machine might be rather minimal, we make this conservative choice to reflect our emphasis on minimizing the impact on the interactive performance perceived by the user.

In the *default policy*, the presence of UI activity is taken as the *only* indicator of whether the machine is active. So, the absence of recent UI activity is taken as an indication that the machine is idle.

A more *conservative policy*, however, also considers the actual computational load on the machine. Specifically, if the CPU usage is above a threshold, the machine is deemed to be active. So, for the machine to be deemed idle, both the absence of recent UI activity and CPU usage being below the threshold are necessary conditions. To avoid too much bouncing between the active and idle states, we introduce hysteresis in the process by (a) measuring the CPU usage as the average over an interval (e.g., 1 minute) rather than instantaneously, and (b) having a higher threshold, c_{push}, for the push list (i.e., idle→active transition of a VM currently on the server) than the threshold, c_{pull}, for the pull list (i.e., for a VM currently on a desktop machine).

5.3 Server Capacity Constraint

A second factor that the controller considers while making migration decisions is the availability of resources on the server. If the server's resources are saturated or close to saturation, the controller migrates some VMs back to the desktop machines to relieve the pressure. Thus, an idle VM is merely *eligible* for being consolidated on the server and, in fact, might not be if the server does not have the capacity. On the other hand, an active VM must be migrated back to the desktop machine even if the server has the capacity. This design reflects the choice to err on the side of being conservative, as noted above.

There are two server resource constraints that we focus on. The first is *memory availability*. Given a total server memory, M, and the allocation, m, made to each VM, the number of VMs that can be hosted on the server is bounded by $n_{RAM} = \frac{M}{m}$. Note that m is the memory allocated to a VM after ballooning and would typically be some minimal value such as 384 MB that allows an idle VM to still function (Section 7.4).

The second resource constraint arises from *CPU usage*. Basically, the aggregate CPU usage of all the VMs on the server should be below a threshold. As with the conservative client-side policy discussed in Section 5.2, we introduce hysteresis by (a) measuring the CPU usage as the average over a time interval (e.g., 1 minute), and (b) having a higher threshold, s_{push}, for pushing out VMs, than the threshold, s_{pull}, for pulling in VMs. The server tries to pull in VMs (assuming the pull list is non-empty) so long as the aggregate CPU usage is under s_{pull}. Then, if the CPU usage rises above s_{push}, the server pushes back VMs. Thus, there is a bound, n_{CPU}, on the number of VMs that can be accommodated such that $\sum_{i=1}^{i=n_{CPU}} x_i \leq s_{push}$, where x_i is the CPU usage of the i^{th} VM.

The total number of VMs that can be consolidated on the server is bounded by $min(n_{RAM}, n_{CPU})$. While one could extend this mechanism to other resources such as network and disk, our evaluation in Section 8 indicates that enforcing CPU constraints also ends up limiting the usage of other resources.

Instead of simply pulling in VMs until the capacity limit is reached, more sophisticated optimizations are possible. In general, the problem of consolidating VMs within the constraints of the server's resources can be viewed as a bin-packing problem [25] since consolidating the multiple new VMs in place of the one that is evicted would likely help save energy. Details of our greedy bin packing algorithm for managing consolidation are described in [22].

5.4 Measuring & Normalizing CPU Usage

Given the heterogeneity of desktop and server physical machines, one question is how CPU usage is measured and how it is normalized across the machines. All measurement of CPU usage in LiteGreen, both on the server and on the desktop machines, is made at the hypervisor level, where the controller and stubs run, rather than within the guest VMs. Besides leaving the VMs untouched and also accounting for CPU usage by the hypervisor itself, measurement at the hypervisor level has the advantage of being unaffected by the configuration of the virtual processors. The hypervisor also provides uniform interface to interact with multiple operating systems.

Another issue is normalizing measurements made on the desktop machines with respect to those made on the server. For instance, when a decision to pull a VM is made based on its CPU usage while running on the desktop machine, the question is what its CPU usage would

be once it has been migrated to the server. In our current design, we only normalize at the level of cores, treating cores as equivalent regardless of the physical machine. So, for example, a CPU usage of $x\%$ on a 2-core desktop machine would translate to a CPU usage of $\frac{x}{4}\%$ on an 8-core server machine. One could consider refining this design by using the CPU benchmark numbers for each processor to perform normalization.

5.5 Putting It All Together: LiteGreen Control Loop

To summarize, LiteGreen's control loop operates as follows. Based on information gathered from the stubs, the controller determines which VMs, if any, have become idle, and adds them to the pull list. Furthermore, based both on information gathered from the stubs and from local monitoring on the server, the controller determines which VMs, if any, have become active again and adds these to the push list. If the push list is non-empty, the newly active VMs are migrated back to the desktop right away. If the pull list is non-empty and the server has the capacity, additional idle VMs are migrated to the server. If at any point, the server runs out of capacity, the controller looks for opportunities to push out the most expensive VMs in terms of CPU usage and pull in the least expensive VMs from the pull list. Pseudocode for the control loop employed by the LiteGreen controller is available at [22].

6 Implementation and Deployment

We have built a prototype of LiteGreen based on the Hyper-V hypervisor, which is available as part of the Microsoft Hyper-V Server 2008 R2 [5]. The Hyper-V server can host Windows, Linux, and other guest OSes and also supports live migration based on shared storage.

Our implementation comprises the controller, which runs on the server, and the stubs, which run on the individual desktop machines. The controller and stubs are written in C# and add up to 1600 and 600 lines of code, respectively. The stubs use WMI (Windows Management Instrumentation) [10] and Powershell to perform the monitoring and migration. The controller also includes a GUI, which shows the state of all of the VMs in the system.

In our implementation, we ran into a few issues from bugs in the BIOS to limitations of Hyper-V and had to work around them. Here we discuss a couple of these.

Lack of support for sleep in hypervisor: Since Hyper-V is intended for use on servers, it does not support sleep once the hypervisor service has been started. Also, once started, the hypervisor service cannot be turned off without a reboot. Other hypervisors such as Xen also lack support for sleep.

We worked around this as follows: when the desktop VM has been migrated to the server and the desktop machine is to be put to sleep, we set a registry key to disable the hypervisor and then reboot the machine. When the machine boots up again, the hypervisor is no longer running, so the desktop machine can be put to sleep. Later, when the user returns and the machine is woken up, the hypervisor service is restarted, without requiring a reboot. Since a reboot is needed only when the machine is put to sleep but *not* when it is woken up, the user does not perceive any delay or disruption due to the reboot.

BIOS bug: On one model of desktop (Dell Optiplex 755), we found that the latest version of BIOS available does not restore prior-enabled Intel VT-x support (needed by the hypervisor) after resuming from sleep. We are currently pursuing a fix to this issue with the manufacturer; until then, we are unable to use this model of desktop as a LiteGreen client.

6.1 Deployment

We have deployed LiteGreen to ten users at MSR India, comprising three administrative staff and seven researchers, three of whom are authors of this paper. As of this writing, the system has been in use for 28 days that includes 10 weekend days and holidays. Accounting for the ramp-up and ramp-down of users in the LiteGreen system, total usage was approximately 3200 user-hours.

Each user is given a separate LiteGreen desktop machine that is running a hypervisor (Hyper-V Server 2008) along with the LiteGreen client stub. The desktop environment runs in a Windows 7 VM that is allocated 2GB of memory. The users' existing desktop is left untouched in order to preserve the users' existing desktop configuration and local data. Different users use their LiteGreen desktop in different ways. Most users use the LiteGreen desktop as their primary access to computing, relying on remote desktop to connect to their existing desktop. A couple of users used it only for specific tasks, such as browsing or checking email, so that the LiteGreen desktop only sees a subset of their activity.

Our findings are reported in Section 7.3. While our deployment is very small in size and moreover, has not entirely replaced the users' existing desktop machines, we believe it is a valuable first step that we plan to build on in the coming months. A video clip of LiteGreen in action on one of the desktop machines is available at [4].

7 Experimental Evaluation

We begin by presenting experimental results based on our prototype. These results are drawn both from controlled experiments in the lab and from our deployment. The results are, however, limited by the small scale of our testbed and deployment, so in Section 8 we present a

Component	Make/Model	Hardware	Software
Desktops (10)	HP WS xw4600	Intel E8200 Core 2 Duo @2.66GHz	Hyper-V + Win7 guest
Server	HP Proliant ML350	Intel Xeon E5440 DualProc 4Core 2.83GHz, 32GB RAM	Hyper-V
Storage	Dell Optiplex 755	Intel E6850 Core 2 Duo 3.00 GHz	Win 2008 + iSCSI
Switch	DLink DGS-1016D	NA	NA

Table 2: Testbed details

larger scale trace-driven evaluation using the traces gathered from the 120 machines at our lab.

7.1 Testbed

Our testbed mirrors the architecture depicted in Figure 2. It comprises ten desktop machines, a server, and a storage node, all connected to a GigE switch. The hardware and software details are listed in Table 2.

We first used the testbed for controlled experiments in the lab (Section 7.2). We then used the same setup but with the desktop machines installed in the offices of the participating users, for our deployment (Section 7.3).

7.2 Results from Laboratory Experiments

We start by walking through a migration scenario similar to that shown in the LiteGreen video clip [4], before presenting detailed measurements.

7.2.1 Migration Timeline

The scenario, shown in Figure 3a, starts with the user stepping away from his/her machine (Event A), causing the machine to become idle. After *actvityWindow* amount of time elapses, the user's VM is marked as idle and the server initiates the VM pull (Event B). After the VM migration is complete (Event C), the physical desktop machine goes to sleep (Event D). Note that, if the user returns to their desktop between events B and C, the migration is simply canceled without any perceivable latency to the user. This is because, during live migration, the (desktop) VM continues to remain fully operational, except during the final switchover phase that typically lasts only tens of milliseconds.

Figure 3b shows the timeline for waking up. When the user returns to his/her desktop, the physical machine wakes up (Event A) and immediately establishes a remote desktop (RD) session to the user's VM (Event B). At this point, the user can start working even though his/her VM is still on the server. A VM push is initiated (Event C) and the VM is migrated back to the physical desktop machine (Event D), in the background using live migration feature.

Figures 3a and 3b also show the power consumed by the desktop machine and the server over time, measured using Wattsup power meters [9]. While the timeline shows the measurements from one run, we also made more detailed measurements of the individual components and operations, which we present next.

7.2.2 Timing of Individual Operations

We made measurements of the time taken for the individual steps involved in migration. In Table 3, we report the results derived from ten repetitions of each step.

Step	Sub-step	Time (sec) [mean (sd)]
Going to Sleep		840.5 (27)
	Pull Initiation	638.8 (20)
	Migration	68.5 (5)
	Sleep	133.2 (5)
Resuming from sleep		164.6 (16)
	Wakeup	5.5
	RD connection	14
	Push Initiation	85.1 (17)
	Migration	60 (6)

Table 3: Timing of individual steps in migration

7.2.3 Power Measurements

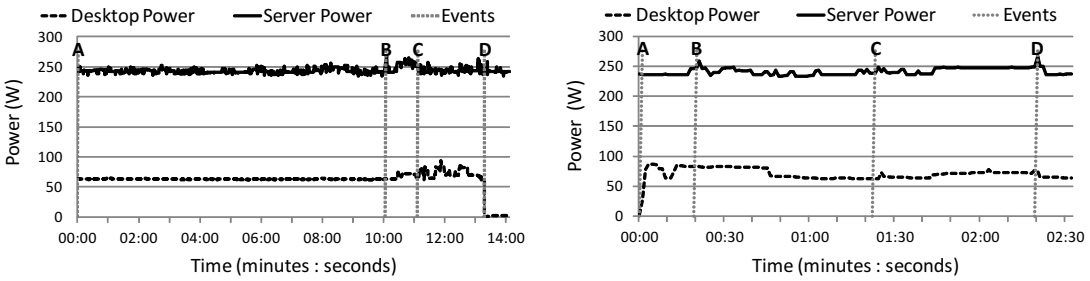
Table 4 shows the power consumption of a desktop machine, the server, and the switch in different modes, measured using a Wattsup power meter.

Component	Mode	Power (W)
Desktop	idle	60-65W
Desktop	100% CPU	95W
Desktop	sleep	2.3-2.5W
Server	idle	230-240W
Server	100% CPU	270W
Switch	idle	8.7 - 8.8W
Switch	during migration	8.7-8.8W

Table 4: Power measurements

The main observation is that power consumption of the desktop and the servers is largely unaffected by the amount of CPU load. It is only when the machine is put to sleep that the power drops significantly. We also see that the power consumption of the network switch is low and is unaffected by any active data transfers. Thus, the energy cost of the migration itself is negligible (the small bump between events B and C in Figure 3a), and can be ignored, as long as one accounts for the time/energy of the powered on desktop until the migration is completed.

Finally, the power consumption curves in Figures 3a and 3b show the marked difference in the impact of migration on the power consumed by the desktop machine



(a) Sleep timeline

(b) Wakeup timeline

Figure 3: Migration timelines

and the server. When a pull happens, the power consumed by the desktop machine goes down from about 60W in idle mode to 2.5W in sleep mode (with a transient surge to 75W during the migration process). On the other hand, the power consumption of the server barely changes. This difference underlies the significant net energy gain to be had from moving idle desktops to the server.

7.2.4 Compression to Reduce Migration Time

The time to migrate the VM — either push or pull — is determined by the memory size (2GB) of the VM and the network throughput. The transfer size can be greater than memory size when application activity during the time of migration results in dirty memory pages that are copied multiple times. We configured a desktop VM with typical enterprise applications, such as Microsoft Office, an email client and a browser with multiple open web pages. We then migrated this VM back and forth, between the desktop and the server. When the VM was on the desktop, we interacted with the applications as a regular desktop user. In this setup, we observed that different migrations resulted in transfer sizes between 2.2-2.7GB. Using a network transfer rate of 0.5Gbps (the effective TCP throughput of active migration on the GigE network), transfer takes about 35-43 seconds. Including the migration initiation overhead, the total migration time is about 60 seconds, which matches the numbers shown in the timeline and in Table 3.

We experimented with a simple compression optimization to reduce the migration time. We used EndRE [16], an end-system redundancy elimination service, with a 250MB packet cache to analyze the savings from performing redundancy elimination in the VM migration traffic between two nodes. EndRE works in a similar fashion to WAN optimizers [18], but on end hosts instead of middleboxes. After identifying and eliminating redundant bytes, as small as 32 bytes, with respect to the packet cache, GZIP is applied to further compress the data. For various transfers, we found that the compressor, operating at 0.4Gbps, was able to reduce the size of transfer by 64-69%. Note that EndRE is designed to be asymmetric. Thus, decompression is inexpensive and does not result in additional latency. This implies that *migration transfer time can be reduced from 35-43 seconds to about 15 seconds using redundancy elimination, thereby significantly speeding up the migration process.* This approach can also help support migration of VMs with larger memory sizes (e.g., 4GB) while limiting the transfer time to under a minute.

7.2.5 Further Optimizations

First, the time to put the machine to sleep is 133 seconds, much of it due to the reboot necessitated by the lack of support for sleep in Hyper-V (Section 6). With a *client* hypervisor that includes support for sleep, we expect the time to go to sleep to shrink to just about 10 seconds.

Second, the time from when the user returns till when they are able to start working is longer than we would like — about 19.5 seconds. Of this, resuming the desktop machine from sleep only constitutes 5.5 seconds. About 4 more seconds are taken by the user to key in their login credentials. The remaining 10 seconds are taken to launch the remote desktop application and make a connection to the user's VM, which resides on the server. This longer than expected duration is because Hyper-V freezes for several seconds after resuming from sleep. We believe that this happens because our unconventional use of Hyper-V, specifically putting it to sleep when it is not designed to support sleep, triggers some untested code paths. We expect that this issue would be resolved with a client hypervisor. However, resuming the desktop and connecting to the users' VM may still take on the order of a few seconds that may be disruptive. One approach to mask this disruption is to anticipate user returns, for example, through user mobile phone tracking, and resume the desktop before the user arrives at his disk. This aspect is discussed in [22]. Such tracking of the user' location could also be used to improve user experience in other days, for instance, by preventing a seemingly idle machine from being migrated to the server, say, if the user is still in their office.

7.3 Results from Deployment

As noted in Section 6.1, we have had a deployment of LiteGreen for a period of 28 days including 10 holidays

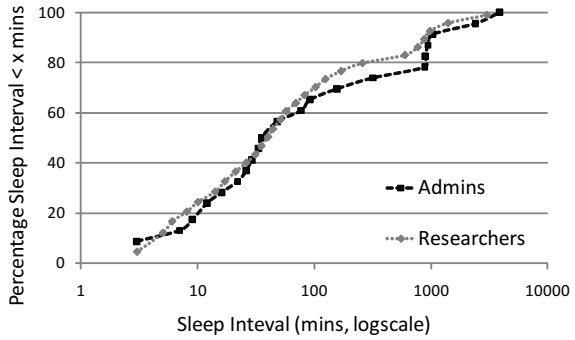


Figure 4: Distribution of desktop sleep durations

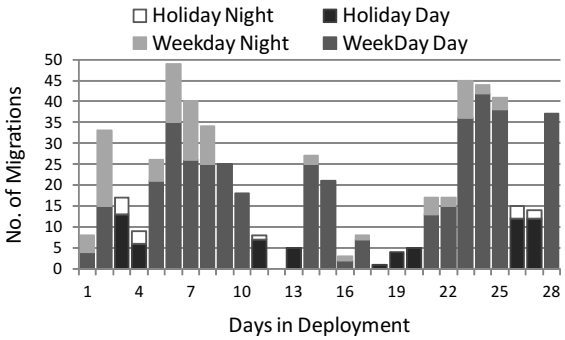


Figure 5: Number of migrations

and weekends, comprising about 3200 user-hours (maximum simultaneous usage by 10 users). While it is hard to draw general conclusions given the small scale and duration of the deployment thus far, it is nevertheless interesting to consider some of the initial results.

7.3.1 Desktop Sleep Time Distribution

Figure 4 shows the cumulative distribution function of the sleep durations for the seven researchers and the three administrative staff. The sleep times tend to be quite short, with a median of 40 minutes across the ten users, demonstrating the exploitation of short idle periods by the LiteGreen system. From the figure, one can see one distinct difference in behavior between the administrators and the researchers in our study. We notice that there is a sharper spike in the curve for the administrators around 900 minutes as compared to the smoother curve for researchers. This is explained by the fact that administrators are more likely than researchers to maintain regular workhours (e.g., 9AM to 6PM) which corresponds to 15 hours (900 minutes) of idle time.

7.3.2 Desktop Average Sleep Time

For our deployment, we used the default policy from Section 5.2 to determine whether a VM was idle or active. During the deployment period, the desktop machines were able to sleep for an average of 87.9% of the time. Even the machine of the most active user in our deployment, who used their LiteGreen desktop for all of their computing activity, slept for 76% of the time.

Note that, while 88% of desktop sleep time may appear unusually large, out of the 3200 user-hours, only about 960 user-hours corresponded to daytime weekdays (8AM – 8PM) in our deployment. Thus, 12% or 384 user-hours of desktop awake time corresponds to 40% of daytime weekday hours, representing a significant fraction of the workday.

7.3.3 Energy Savings

The conversion of desktop average sleep time to energy savings requires accounting of the energy costs of the server. While a LiteGreen server was necessary for this deployment, it was significantly under-utilized since it was dedicated to host only 10 idle VMs. If we amortize the cost of the server over a larger number of desktops (e.g., 60), the power cost of the server per desktop is 4.2W (see Section 8.4 for details). We use this amortized value of the server power cost below.

We use the power measurement numbers from Table 4 to estimate energy savings from LiteGreen. Let us assume power consumption of 62.5W for an idle desktop, 95W for a fully active desktop and 2.5W for a sleeping desktop. From Figure 1a, where CPU usage is less than 10% for 90% of the time, let us assume a desktop that never sleeps consumes 62.5W of power 90% of the time and 95W of power 10% of the time. Then, desktop power consumption, without any energy savings, is simply (0.9*62.5+0.1*95) = 65.75W per desktop.

In LiteGreen, since the average desktop sleep time is 88%, the power savings is (0.88*(62.5 - 2.5) - 4.2) = 48.6 W per desktop or 74% of total desktop energy consumption.

Finally, the above energy savings calculations are applicable for enterprises that already have a centralized storage deployment. Otherwise, we need to take into account the energy consumed by the centralized storage system as well. Consider a network attached storage box such as the QNAP SS-839 Pro Turbo [6] that can host up to 8 disks and consumes 34W in operation. Assuming two desktop users are multiplexed onto each disk, each of these storage devices can support up to 16 desktops. Thus, the amortized energy cost of centralized storage is 34/16 = 2.1W/desktop. Accounting for the storage overhead, the power savings in LiteGreen is 48.6 - 2.1 = 46.5W per desktop or 71%.

7.3.4 Number of Migrations

Finally, Figure 5 shows the number of migrations for the different days of deployment, segregated by daytime (8 am–8 pm) and nightime (8 pm–8 am), and further classified by weekdays and holidays (including weekends). There were a total of 571 migrations during the deployment period. The number of migrations are higher during

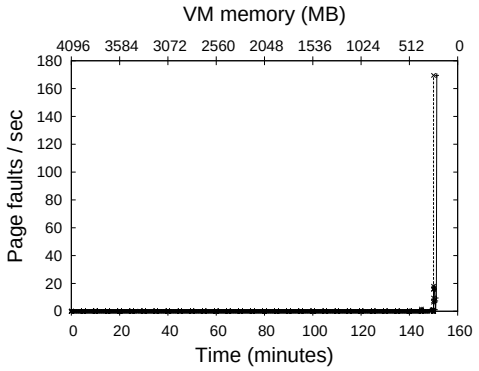


Figure 6: Memory ballooning experiment: every 5 minutes memory of a desktop VM is reduced by 128M. Initial memory size is 4096M

the day compared to night (470 versus 101) and higher in the weekdays compared to the holidays (493 versus 78). These numbers are again consistent with the LiteGreen approach of exploiting short idle intervals.

7.4 Experiments with Xen

We would like to evaluate the effectiveness of memory ballooning in relieving pressure on the server's memory resources due to consolidation. However, Hyper-V does not currently support memory ballooning, so we conducted experiments using the Xen hypervisor, which supports memory ballooning for the Linux guest OS using a balloon driver (we are not aware of any balloon driver for Windows). We used the Xen hypervisor (v3.4.2 built with 2.6.18 SMP kernel) with the Linux guest OS (CentOS 5.4) on a separate testbed comprising two HP C-class blades, each equipped with two quad-core 2.2 GHz 64-bit processors with 48GB memory, two Gigabit Ethernet cards, and two 146 GB disks. One blade was used as the desktop machine and the other as the server.

The desktop Linux VM was initially configured with 4096 MB of RAM. It ran an idle workload comprising the Gnome desktop environment, two separate Firefox browser windows, with a Gmail account and the CNN main page open (each of these windows auto-refreshed periodically without any user's involvement), and the user's home directly mounted through SMB (which also generated background network traffic). The desktop VM was migrated to the server. Then, memory ballooning was used to shrink the VM's memory all the way down to 128 MB, in steps of 128 MB every 5 minutes.

Figure 6 shows the impact of memory ballooning on the page fault rate. The page fault rate remains low even when the VM's memory is shrunk down to 384 MB. However, shrinking it down to 256 MB causes the page fault rate to spike, presumably because the working set no longer fits within memory.

We conclude that in our setup with the idle workload that we used, memory ballooning could be used to shrink the memory of an idle VM by over a factor of 10 (4096 MB down to 384 MB), without causing thrashing. Further savings in memory could be achieved through memory sharing. While we were not able to evaluate this in our testbed since neither Hyper-V nor Xen supports it, the findings from prior work [26] are encouraging, as discussed in Section 9.

8 Trace-driven Analysis

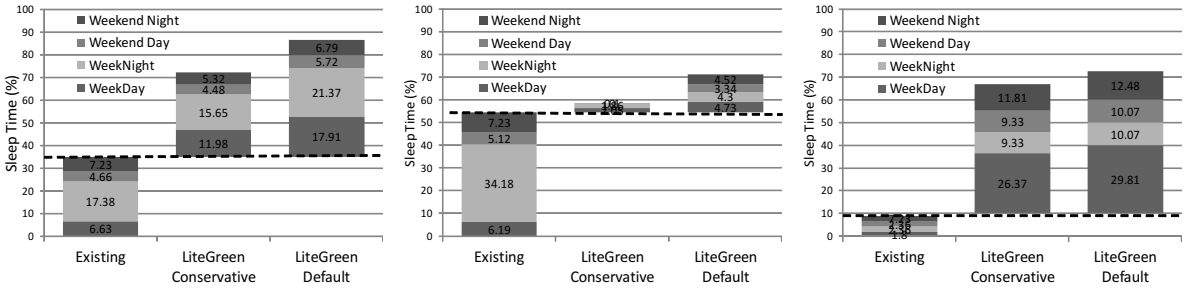
To evaluate our algorithms further, we have built a discrete event simulator written in Python using the SimPy package. The simulator runs through the desktop traces, and simulates the default and conservative policies based on various parameters including c_{pull} (client resource threshold below which client VMs are eligible to be pulled to server), c_{push} (client resource threshold above which client VMs are pushed to client) , s_{pull} (server resource thrshold above below client VMs are pulled to server), s_{push} (server resource threshold above which client VMs are pushed to clientdis) and $ServerCapacity$. In the rest of the section, we will report on energy savings achieved by LiteGreen and utilization of various resources (CPU, network, disk) at the server as a result of consolidation of the idle desktop VMs.

8.1 Desktop Sleep Time

Figure 7a shows the desktop sleep time for all the users with existing mechanisms and LiteGreen, default and conservative. For both the policies, we use $c_{pull} = 10$ (less than 10% desktop usage classified as idle), $c_{push} = 20$, $s_{pull} = 600$, $s_{push} = 700$ and $ServerCapacity = 800$ intended to represent a Server with 8 CPU cores.

As mentioned earlier, our desktop trace gathering tool records a number of parameters, including CPU, memory, UI activity, disk, network, etc., every minute after its installation. In order to estimate energy savings using existing mechanisms (either automatic windows power management or manual desktop sleep by the user), we attribute any unrecorded interval or "gaps" in our desktop trace to energy savings via existing mechanisms. Using this technique, we estimate that existing mechanisms would have put desktops to sleep 35.2% of the time.

We then simulate the migrations of desktop VMs to/from the server depending on the desktop trace events and the above mentioned thresholds for the conservative and default policy. Using the conservative policy, we find that LiteGreen puts desktop to sleep for 37.3% of the time. This is in addition to the existing savings, for total desktop sleep time of 72%. If we use the more aggressive default policy, where the desktop VM is migrated to the server unless there is UI activity, we find that LiteGreen puts desktop to sleep for 51.3% on time for a total



(a) Desktop sleep time (b) Desktop sleep time for user1 (c) Desktop sleep time for user2

Figure 7: Desktop sleep time from existing power management and LiteGreen's default and conservative policies

desktop sleep time of 86%.

The savings of the different approaches are also classified by day (8AM-8PM) and night (8PM-8AM) and also whether it was a weekday or a weekend. We note that substantial portion of LiteGreen desktop sleep time comes from weekdays, thereby highlighting the importance of exploting short idle intervals for energy savings.

8.2 Desktop Sleep Time for Selected Users

Based on the CPU utilization trace data, we found a user, say user1, who had bursts of significant activity separated by periods of no activity, likely because he/she manually switches off his/her machine when not in use. For this particular case, LiteGreen is unable to significantly improve on the energy savings of existing mechanisms (i.e., manual power management). This is reflected in the desktop sleep time for user1 in Figure 7b.

In contrast, for many of the users, say user2, the desktop exhibits low-levels of CPU activity with occasional spikes almost continuously, with only short gaps of inactivity. The few CPU utilization spikes can prevent Windows power management from putting the desktop to sleep, thereby wasting a lot of energy. However, LiteGreen is able to exploit this situation effectively, and puts the desktop to sleep for significantly longer time as shown in Figure 7c.

8.3 Server Resource Utilization during Consolidation

While the default policy provides higher savings than the conservative policy, it is clear that the default policy would stress the resources on the server more, due to hosting of more number of desktop VMs, than the conservative policy. We examine this issue next.

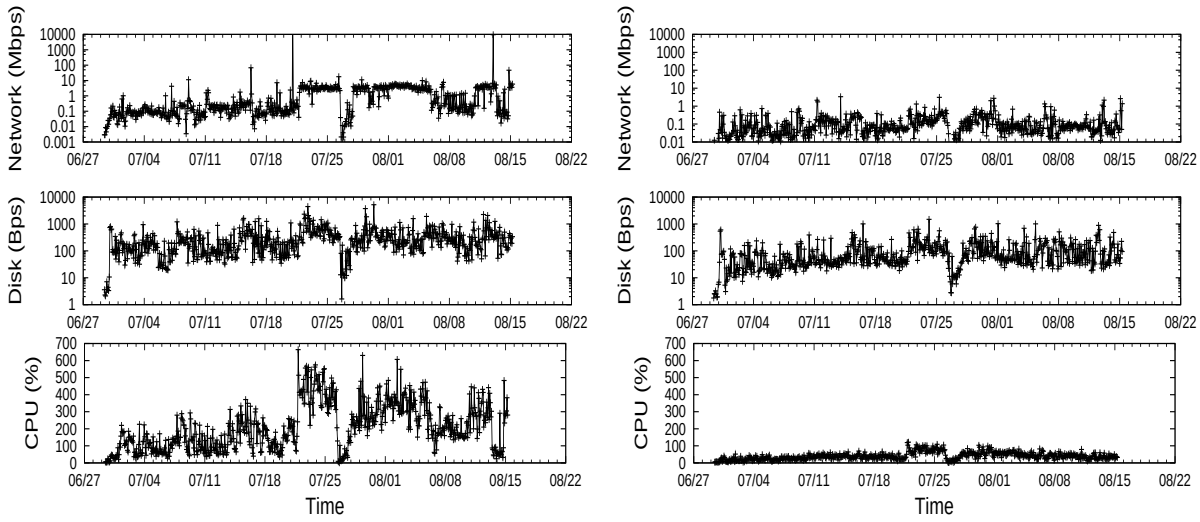
Figures 8a and 8b show the resource utilization due to idle desktop consolidation at the server for the default and conservative policies, respectively. The resources shown are CPU usage, bytes read/second from the disk, and network usage in Mbps.

First, consider CPU. Notice that the CPU usage at the server in the default policy spikes up to between $s_{pull} = 600$ an $s_{push} = 700$ but, as intended, never goes above s_{push}. In contrast, since the conservative policy pushes the VM back to the desktop as soon as $c_{push} = 20$ is exceeded, the CPU usage at the server hardly exceeds an utilization value of 100. Next consider disk reads. It varies between 10B-10KB/s for the default policy (average of 205 B/s) while it varies betwen 10B-1KB/s for the conservative policy (average of 41 B/s). While these numbers can be quite easily managed by the server, note that these are disk reads of idle, and not active, desktop VMs. Finally, let us consider network activity of the consolidated idle desktop VMs. For the default policy, the network traffic mostly varies between 0.01 to 10Mbps, but with occassional spikes all the way up to 10Gbps. In the case of conservative policy, the network traffic does not exceed 10Mbps and rarely goes above 1Mbps. While these network traffic numbers are manageable for a single server, these represent the workload of *idle* desktop machines. Scaling the server infrastructure to enable consolidation of *active* desktop VMs, as in the thin client model, will likely be expensive.

8.4 Energy Savings

We use calculations similar to the one performed in Section 7.3.3 for computing energy savings. Recall that power consumption of a desktop, without any energy savings mechanism, is simply (0.9*62.5+0.1*95) = 65.75 W per desktop.

Using existing energy saving mechanisms, where the desktop is put to sleep 35.2% of the time (Section 8.1), 0.352*(62.5-2.5) = 21.1 W per desktop or 32% of energy savings can be achieved. In the case of LiteGreen, our consolidation analysis (Section 8.3) suggests that one 8-core server is capable of hosting the idle desktops in the trace. Memory balooning results from Section 7.4 suggest that an idle VM could be packed in 384MB, implying that a 32GB server has enough memory capacity for up to to 96 idle VMs. Assuming some over-provisioning for capacity and redundancy, let us dedicate two servers



(a) Default policy

(b) Conservative policy

Figure 8: Resource utilization during idle desktop consolidation

of 250W each for the 120 desktops. The amortized cost of a server/desktop is then 500W/120 = 4.2W. Thus, power savings in LiteGreen (using the default policy with average desktop sleep time of 86%) is (0.86*(62.5 - 2.5) - 4.2) = 47.4 W per desktop or 72%, more than doubling the energy savings under existing mechanisms.

Finally, as in Section 7.3.3, if we were to include the amortized energy cost of centralized storage (2.1W/desktop), the energy savings in LiteGreen using the default policy is simply 47.4 - 2.1 = 45.3W per desktop or 69%.

9 Limitations and Future Work

We consider some limitations of LiteGreen, which also point to directions for future work.

9.1 Dependence on Shared Storage

Live migration assumes that the disk is shared between the source and destination machines, say in the form of network attached storage (NAS). This avoids the considerable cost of migrating disk content. However, this is a limitation of our current system since, in general, client machines would have a local disk, which applications (e.g., sharing of local files) need access to.

Recent work has demonstrated the migration of VMs with local virtual hard disks (VHDs) by using techniques such as pre-copying and mirroring of disk content [19] to keep the downtime to under 3 seconds in a LAN setting. Note that since the base OS image is likely to be already available at the destination node, the main cost is that of migrating the user data.

To quantify the costs involved in migrating the local disk, we performed detailed tracing of all file system operations on 3 actively used desktop machines using the `ProcessMonitor` tool [35]. Table 5 summarizes the volume of dirty disk blocks that is generated, which represents the amount of disk state that would need to be migrated. We consider two cases: dirty blocks being pre-copied every hour versus every 4 hours. The latter provides a greater opportunity for temporal consolidation (i.e., merging of multiple writes to a block).

1-hour window (MB per hour)	4-hour window (MB per hour)
80-240	40-100

Table 5: Volume of dirty disk blocks

Migrating 100 MB of disk content over a GigE network would take 1.6 seconds, assuming an effective throughput of 500 Mbps. This means that over 2000 disk migrations can be supported per hour, which suggests that these migrations will not be the bottleneck. Further optimizations are possible, for instance, by transferring dirty data at a sub-block level and filtering out writes to scratch space.

Note that enterprise enviroments often employ network storage to hold persistent user data, since this enables the data to be backed up. In such a setting, the amount of data to be migrated would be further reduced to only temporary files generated by applications.

9.2 Heavyweightness

LiteGreen is a more heavyweight solution than the alternative proxy-based approach. To deploy LiteGreen, we would need to have desktop machines run a client hypervisor and also provision the necessary network bandwidth and server resources.

We believe that technology trends make it likely that the enterprise IT infrastructure would move in this direction. Virtualized desktops simplify management for the IT administrators. Also, the growth in thin clients would

argue for server and network provisioning. Finally, the desire to support mobility and a "work from anywhere" capability would likely spur the development of a hybrid computing model wherein the desktop VM resides on the server when accessed from a thin client and migrates to the local machine at other times. Thus, we believe that that the LiteGreen approach fits in with these trends.

10 Conclusion

Recent work has recognized that desktop computers in enterprise environments consume a lot of energy in aggregate while still remaining idle much of the time. The question is how to save energy by letting these machines sleep while avoiding user disruption. LiteGreen uses virtualization to resolve this problem, by migrating idle desktops to a server where they can remain "always on" without incurring the energy cost of a desktop machine. The seamlessness offered by LiteGreen allows us to aggressively exploit short idle periods as well as long periods. Data-driven analysis of more than 65000 hours of desktop usage traces from 120 users as well as a small-scale deployment of LiteGreen on ten desktops, comprising 3200 user-hours over 28 days, shows that LiteGreen can help desktops sleep for 86-88% of the time. This translates to estimated desktop energy savings of 72-74% for LiteGreen as compared to 32% savings under existing power management mechanisms.

Acknowledgements

Rashmi KY helped with the desktop usage tracing effort at Microsoft Research India. Our shepherd, Katerina Argyraki, and the anonymous reviewers provided valuable feedback on the paper. We thank them all.

References

[1] AMD Virtualization Technology (AMD-V). http://www.amd.com/us/products/technologies/virtualization/Pages/amd-v.aspx.

[2] Citrix XenServer. http://www.citrix.com/xenserver/.

[3] Intel Virtualization Technology (VT-x). http://www.intel.com/technology/itj/2006/v10i3/1-hardware/5-architecture.htm.

[4] LiteGreen demo video. http://research.microsoft.com/en-us/projects/litegreen/default.aspx.

[5] Microsoft Hyper-V. http://www.microsoft.com/windowsserver2008/en/us/hyperv-main.aspx.

[6] QNAP SS-839 Pro Turbo Network Attached Storage. http://www.qnap.com/pro_detail_hardware.asp?p_id=124.

[7] Remote Desktop Protocol. http://msdn.microsoft.com/en-us/library/aa383015(VS.85).aspx.

[8] VMWare ESX Server. http://www.vmware.com/products/esx/.

[9] Wattsup Meter. http://www.wattsupmeters.com.

[10] Windows Management Instrumentation. http://msdn.microsoft.com/en-us/library/aa394582(VS.85).aspx.

[11] White Paper: Benefits and Savings of Using Thin Clients, 2X Software Ltd., 2005. http://www.2x.com/whitepapers/WPthinclient.pdf.

[12] White Paper: Wake on LAN Technology, June 2006. http://www.liebsoft.com/pdfs/Wake_On_LAN.pdf.

[13] Advanced Configuration and Power Interface (ACPI) Specification, June 2009. http://www.acpi.info/DOWNLOADS/ACPIspec40.pdf.

[14] Emerging Technology Analysis: Hosted Virtual Desktops, Gartner, Feb. 2009. http://www.gartner.com/DisplayDocument?id=887912.

[15] Worldwide PC 20092013 Forecast, IDC, Mar. 2009. http://idc.com/getdoc.jsp?containerId=217360.

[16] AGARWAL, B., AKELLA, A., ANAND, A., BALACHANDRAN, A., CHITNIS, P., MUTHUKRISHNAN, C., RAMJEE, R., AND VARGHESE, G. EndRE: An End-System Redundancy Elimination Service for Enterprises. In *USENIX NSDI* (Apr. 2010).

[17] AGARWAL, Y., HODGES, S., CHANDRA, R., SCOTT, J., BAHL, P., AND GUPTA, R. Somniloquy: Augmenting Network Interfaces to Reduce PC Energy Usage. In *NSDI* (Apr. 2009).

[18] ANAND, A., MUTHUKRISHNAN, C., AKELLA, A., AND RAMJEE, R. Redundant in Network Traffic: Findings and Implications. In *ACM SIGMETRICS* (Seattle, WA, June 2009).

[19] BRADFORD, R., KOTSOVINOS, E., FELDMANN, A., AND SCHIOEBERG, H. Live Wide-Area Migration of Virtual Machines Including Local Persistent State. In *ACM VEE* (2007).

[20] CHASE, J., ANDERSON, D., THAKAR, P., VAHDAT, A., AND DOYLE, R. Managing energy and server resources in hosting centers. In *SOSP* (October 2001).

[21] CLARK, C., FRASER, K., HAND, S., HANSEN, J. G., JUL, E., LIMPACH, C., PRATT, I., AND WARFIELD, A. Live Migration of Virtual Machines. In *NSDI* (May 2005).

[22] DAS ET AL., T. LiteGreen: Saving Energy in Networked Desktops using Virtualization, Extended Version,. http://research.microsoft.com/en-us/projects/litegreen/litegreen.pdf.

[23] DAVID, B. White Paper: Thin Client Benefits, Newburn Consulting, Mar. 2002. http://www.thinclient.net/technology/Thin_Client_Benefits_Paper.pdf.

[24] DOWTY, M., AND SUGERMAN, J. GPU Virtualization on VMware's Hosted I/O Architecture. In *USENIX WIOV* (2008).

[25] GAREY, M. R., AND JOHNSON, D. S. *Computers and intractability; a guide to the theory of NP-completeness*. W.H. Freeman, 1979.

[26] GUPTA, D., LEE, S., VRABLE, M., SAVAGE, S., SNOEREN, A. C., VARGHESE, G., VOELKER, G. M., AND VAHDAT, A. Difference Engine: Harnessing Memory Redundancy in Virtual Machines. In *OSDI* (Dec. 2008).

[27] KOZUCH, M., AND SATYANARAYANAN, M. Internet Suspend/Resume. In *IEEE WMCSA* (June 2002).

[28] MADDEN, B. Understanding the role of client and host CPUs, GPUs, and custom chips in RemoteFX. http://tinyurl.com/38wqson.

[29] MOORE, J., CHASE, J., RANGANATHAN, P., AND SHARMA, R. Making Scheduling Cool: Temperature-aware Workload Placement in Data Centers. In *Usenix ATC* (June 2005).

[30] NATHUJI, R., AND SCHWAN, K. VirtualPower: Coordinated Power Management in Virtualized Enterprise Systems. In *SOSP* (Oct. 2007).

[31] NEDEVSCHI, S., CHANDRASHEKAR, J., LIU, J., NORDMAN, B., RATNASAMY, S., AND TAFT, N. Skilled in the Art of Being Idle: Reducing Energy Waste in Networked Systems. In *NSDI* (Apr. 2009).

[32] NORDMAN, B. Networks, Energy, and Energy Efficiency. In *Cisco Green Research Symposium* (2008).

[33] NORDMAN, B., AND CHRISTENSEN, K. Greener PCs for the Enterprise. In *IEEE IT Professional* (2009), vol. 11, pp. 28–37.

[34] REICH, J., KANSAL, A., GORACKZO, M., AND PADHYE, J. Sleepless in Seattle No Longer. In *USENIX ATC* (2010).

[35] RUSSINOVICH, M., AND COGSWELL, B. Process Monitor v2.8, 2009. http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx.

[36] SMITH, J. M. A Survey of Process Migration Mechanisms. In *ACM SIGOPS Operating Systems Review* (July 1988), vol. 22, pp. 28–40.

[37] SRIKANTAIAH, S., KANSAL, A., AND ZHAO, F. Energy Aware Consolidation for Cloud Computing. In *HotPower* (Dec. 2008).

[38] TOLIA, N., WANG, Z., MARWAH, M., BASH, C., RANGANATHAN, P., AND ZHU, X. Delivering Energy Proportionality with Non Energy Proportional Systems Optimizations at the Ensemble Layer. In *HotPower* (Dec. 2008).

[39] WALDSPURGER, C. Memory Resource Management in VMware ESX Server. In *OSDI* (Dec. 2002).

[40] WOOD, T., SHENOY, P. J., VENKATARAMANI, A., AND YOUSIF, M. S. Black-box and Gray-box Strategies for Virtual Machine Migration. In *NSDI* (Apr. 2007).