Effective use of sleep states with context-aware selective resume

Eric J. Wright, Nilton Bila, Eyal de Lara, Ashvin Goel

University of Toronto

Abstract

In response to increasing power costs and the social importance of improving energy efficiency, there has been significant work on taking advantage of a computer's ability to enter low-power sleep states. However, current approaches do not allow fine-grained control of the transition process, so times between the power states are long. This limits the usefulness of these states, especially when computers are accessed sporadically.

We introduce a context-aware selective resume framework that selectively resumes a system, with only the minimal set of devices needed for the waking task, to save power and shorten transition times. In this paper, we discuss the classes of applications that would benefit from selective resume, describe the design and implementation of CAESAR, our selective resume framework, and evaluate two applications, a temperature monitoring system and a fast memory server. Our evaluation demonstrates that context-aware selective resume can improve resume latency by 3.9X and energy savings over 5X compared to traditional approaches.

Categories and Subject Descriptors D.4.0 [*Software*]: OPERATING SYSTEMS—General; C.0 [*Computer Systems Organization*]: GENERAL—System architectures; Hardware/software interfaces

General Terms Design, Experimentation, Measurement, Performance

Keywords Power Management, Selective Resume, Suspend, CAESAR

1. Introduction

Low-power states in PCs are designed to handle traditional PC usage, where the PC is on when the user is actively engaging it in an interactive task, and asleep when the user is away. In this context, low power ACPI modes [1], such as S3 (Suspend-to-RAM), provide a good tradeoff between low energy consumption when sleeping and moderate resume latencies that are within human tolerance.

The problem is that these low power states are not a good match for usage scenarios that involve sporadic computation without user involvement. Consider, for example, a desktop with network presence. The system may be configured to sleep when idle while remaining network accessible for remote administration, file access, or to maintain existing connections. The result is a system that needs to resume from sleep often and inspect packets destined to it. A large portion of this traffic constitutes network noise, and because resume and suspend cycles are slow, it prevents the desktop from sleeping for much of the time [15]. Similarly, a memory server that supports partial consolidation of idle desktop VMs to save energy, as proposed in [9], requires the desktop to resume quickly to satisfy page faults from the consolidated VM and quickly return to sleep. Finally, remote sensing applications that periodically aggregate sensor readings from a cluster of hosts also require these hosts to wake up to provide a sensor reading and quickly return to sleep. For these sporadic usage scenarios, the current approach of waking up the full system is too slow and energy inefficient.

There are fundamental differences between the interactive and sporadic usage scenarios. Interactive resume assumes that the PC will stay on for minutes or hours; sporadic resume assumes the PC will stay on for seconds. Interactive resume assumes the user will require all the PC's resources; sporadic scenarios needs only specific resources, such as the NIC and a sensor. Interactive resume cares about minimizing the time before the system becomes responsive to user IO; sporadic resume cares about minimizing the time it takes to complete the resume→perform task→suspend cycle and energy consumption during that cycle. This paper introduces the concept of *context-aware selective resume* which addresses these issues and improves the usability of low power states for sporadic access scenarios.

Currently, when a system enters (suspends) or exits (resumes) from a low power state, both the BIOS and the OS must iterate through all devices on the system for the suspend and resume operations. A suspend places all devices in low-power state, a resume returns all devices to their pre-sleep state. Additionally, the systems do not know why re-

sume requests were made, i.e., they are not aware of their resume context. As such, they are unable to adapt the transition process to its intended use. This approach is inflexible and causes long resume latencies. For example, it can still take up to 30 seconds to fully resume from S3 sleep [5]. While some vendors are claiming improved resume transition times [7], the approach is still "all or nothing".

This paper introduces CAESAR[1] a framework for context-aware selective resume. The CAESAR framework reduces resume and suspend latencies, increases energy efficiency, and improves throughput by enabling systems to only resume hardware that is specifically needed to complete a task. Additionally, we have built two prototype applications using CAESAR, a temperature monitoring system and a fast memory server. The temperature monitoring application enables a sleeping system to provide remote access to the ambient temperature sensor on a machine. The fast memory server enables a sleeping system to provide access to system memory for on-demand memory operations useful for applications utilizing virtual machine migration [9, 13]. In both cases, these applications were implemented at the OS level to demonstrate their value. However, we believe further gains are possible through firmware level implementations.

To evaluate the benefits of the context-aware selective resume functionality provided by CAESAR, we conducted experiments to compare the resume latencies, energy savings and throughput of our two applications in comparison to non-selective resume versions of the application. Our results demonstrate that compared to current resume approaches, a selective resume can improve resume latencies by over 3.9X. Additionally, our results demonstrate that selective resume can improve energy savings by over 5X. We also show the ability of a selective resume approach to realize energy savings in cases where they are not attainable with current approaches. Finally we review related work and discuss how selective resume can improve upon their effectiveness.

2. Motivation

Selective resume provides benefits to applications that want sporadic access to only a part of a system, such as just CPU and memory, quickly and efficiently. Below, we briefly describe a normal and selective resume operation, and then present various scenarios that would benefit from selective resume.

2.1 Resume Operations

The normal resume process begins when the system receives a power management event (PME) triggered by a device, such as the NIC or the CMOS timer. In the normal resume process, the hardware initialization powers on the devices, then the BIOS code performs low-level initialization of the CPU, chipsets and other devices on the system. It then passes control to the OS, which in turn iterates through all devices to reinitialize them to their pre-sleep state, and starts threads that were previously suspended. The BIOS and OS follow a predetermined order of events during every resume.

In contrast, with selective resume, after the hardware initialization, the BIOS initializes only the CPU, then determines its resume context by reading the source of the PME. For example, if the resume was caused by a network packet, the firmware reads the resume context embedded in the packet from the NIC. Based on the resume context, which specifies the required application, the system selectively initializes any required devices and then jumps to one of several selective resume vectors. For example, a selective resume vector could be a memory address that transfers control back to 1) the OS, which then continues the selective resume process, or 2) a special in-memory user-code segment designed to operate without the OS, or 3) to a firmware code segment.

2.2 Selective Resume Scenarios

The ability to provide context to a sleeping system upon resume allows fine-grained control over device use, enabling the various scenarios that we describe below.

Remote Desktop Access: A common scenario is a user wishing to conserve power while still having their system available for remote access and maintaining existing connections. In this case, users can configure the system to either wake up whenever it receives any network packet destined to it (wake on directed packet) or to wake up only on reception of special Wake-on-LAN packets. To use Wake-on-LAN packets, all applications that communicate with the system need to be modified so they precede normal traffic with Wake-on-LAN requests, which limits the number of applications that will work. As a result, allowing the system to wake up on directed packet is the most readily deployable solution. Unfortunately, this leads to very little actual system sleep as the machine is woken up by a steady stream of network traffic arriving in intervals that are much smaller than the time it takes for the machine to complete a resume and suspend cycle [15]. While many solutions have been proposed to address this issue [4, 5, 8, 11, 15], the fundamental limitation is that the system lacks an awareness of its resume context and the cycle time is too long, which limits the benefits of sleep. With selective resume, the resume context would be specified in the packet triggering the Wake-On-LAN. Upon receiving this packet the firmware would wake the system but initialize only the CPU, skipping the memory controller initialization (since it could use the CPU's L2 cache as working memory[14]) and all other devices, and then transfer control to the selective resume vector. The system would then read the packet from the Wake-Up Packet Memory, a set of registers in the NIC that store the Wake-on-LAN packet, and examine the resume context to determine if a full resume is really needed and if not, return to sleep. However, if a resume was needed, it would continue and complete the previously skipped BIOS initialization, such

[1] The name CAeSaR is a play on context-aware selective resume.

as the memory controller, then transfer control back to the OS by calling the normal resume vector. As shown in Section 3.3, selective resume and sleep could be performed in sub-second time, and so the system would be able to spend the majority of its time asleep, eliminating the need for network proxies [15], additional servers [5] or device augmentation [4].

Partial Desktop Migration: A partial desktop VM migration scheme would also benefit from selective resume [8]. In this approach, a minimal working set of memory from an idle virtualized desktop is migrated to a consolidation server to enable it to remain active while the host machine is put to sleep. However, if an activity occurs on the migrated VM that triggers a page fault, the system must wake up the original system and fetch a page of memory. This incurs a significant delay as the system completes a full resume, enabling all devices and loading the OS, only to fetch a 4KB page. With selective resume, a Wake-On-LAN packet with a memory page address would be sent, and it would selectively resume only the CPU, memory, and NIC, skipping the disk, graphics cards, and other devices, and transfer control to the selective resume vector. This operation would not require resuming the entire OS as the firmware would have direct access to memory, be able to read the page, and return the page by constructing and sending the packet directly via the NIC. This approach allows the migrated VM to quickly fetch the page and respond within application timeout limits.

Computational Offloading: In cloudlet scenarios [16] mobile systems take advantage of low-latency, high-bandwidth connections to a nearby cloud environment running virtual machines to offload computationally complex tasks.

In this approach, a mobile device sends a VM overlay (the state difference between a base VM stored on the server and the instance on the device) to a cloudlet server that merges it with the base image, performs the computation, and returns the residual VM difference back to the device when complete. A selective resume approach would enable the size of a VM overlay to be reduced as certain components, for example data files, could be left on the device but would still be accessible using a pull-based approach if needed. This smaller VM overlay means that only the minimal set of data is transferred to a cloudlet, saving both transmission energy usage and device energy usage, as the device can be put to sleep sooner. Additionally, the ability to rapidly resume the device means more opportunities for putting it to sleep can be exploited since we can take advantage of using the cloudlet for shorter tasks that would not be worth offloading with longer resume times.

Power Efficient Data Transfers: A selective resume system would also enable energy savings and quick response times in scenarios that wish to transfer data to or from a sleeping machine. For example in CatNap [12], the authors present a system that exploits the bandwidth differential between LAN and WAN links. Transfer data is buffered at a proxy device connected to the system by a fast LAN link (e.g. a 100Mb Ethernet). The system is put to sleep while the data is sent over a slow WAN link (e.g. a 1Mb ADSL). The data received from the WAN is buffered on the proxy device, and the system is only woken up once the proxy cache is full. With selective resume, transition times would be much shorter and this would maximize the sleep time and reduce the power consumption while awake, lowering the overall energy usage of the system.

Remote Sensing: Applications that remotely access sensor devices on idle systems can also benefit from selective resume. One such application is a data center temperature monitor which periodically aggregates temperature readings from hosts throughout the data center. Because the readings are only periodic, hosts that are idle can go to sleep and wake up quickly only when readings are required.

In Section 3 we demonstrate the potential for improvements in the suspend/resume cycles times of modern hardware, and in Section 4, we show how our implementation of selective resume meets the requirements of the applications we described.

3. Profiling Sleep Transitions

In this section, we describe a set of experiments we conducted to better understand sleep transition times in existing systems, including the asleep→awake→asleep cycle times.

Component	Component System	Integrated System
CPU	Dual-core 2.4GHz Intel, 1066Mhz FSB, 4MB L2	Dual-core 2.6GHz Intel, 800Mhz FSB, 2MB L2
Memory	4GB	2GB
Hard Drive	74G SATA	74G SATA
Graphics	EVGA 8800 GTX	Intel X4500, integrated
Sound	Sound Blaster Extreme	Realtek, integrated
NIC	NVIDIA, integrated	Realtek, integrated
Optical	DVD-R/W	<none>
Power	650 Watt PSU	200 Watt PSU

Table 1. Experimental Systems

3.1 Profiling Setup

Since power consumption, BIOS and hardware latency times are hardware specific, we conducted experiments on two different systems, a component system, representing a higher-end machine, and an integrated system, representing an office PC, as shown in Table 1. We used the same hard drive in both systems, configured with Debian Linux and the 2.6.18.8 kernel. Our experiments utilize the ACPI S3 sleep state.

3.2 System Transition Times

The S0→S3 sleep transition shows the OS time from issuing a pm-suspend command from the shell until the final call to suspend the hardware. The BIOS is not involved in this transition. The S3→S0 (user) resume transition shows both the BIOS and OS time until control was returned to the shell. The S3→S0 (net) resume transition shows both the BIOS and OS time until a connection was established to a remote

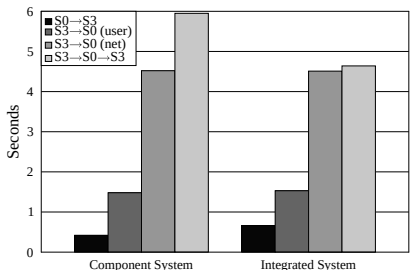

Figure 1. Suspend & Resume Times (w/o Hardware)

server. In all these cases, the systems were given at least 30 seconds to stabilize before another transition was initiated. The S3→S0→S3 case shows a complete cycle, consisting of resuming the system, then suspending it, but without any time to stabilize between transitions. These results do not include hardware latencies, which we show later. We conducted multiple runs and the average values shown have a standard deviation of less than 0.01.

Our findings, shown in Figure 1, demonstrate that the resume operations typically take considerably longer than the suspend operation, 1.48s vs. 0.42s, for the component system, and 1.53s vs. 0.66s, for the integrated system. More interesting was the impact of calling a suspend directly after a resume. The cycle times were 5.95s, for the component system, and 4.64s, for the integrated system, which are significantly larger than, 1.9s and 2.19s, the respective sums of the S3→S0 and S0→S3 times. This large increase was a result of the systems having to complete the background device resume operations, prior to beginning the subsequent suspend operation.

3.3 System Resume Profiling

To evaluate the potential benefits of selective resume, we profiled the systems by analyzing where and how time was spent during a resume operation. The test systems were connected to a server on the same subnet. The server issued a Wake-On-LAN packet to initiate a resume, and we measured how long it took before a response was received from the test systems by the server. We instrumented the kernel to log key events, then conducted multiple runs and used average times based on the timestamps found in the system log (/var/log/syslog). Hardware initialization time was inferred by subtracting the BIOS and OS time accounted for in the system logs from the total time taken to open a connection to the remote server (measured on the server).

As shown in Figure 2, the activities and time spent initializing the hardware and executing the BIOS code is specific to each system, but the majority of the resume time is spent in software, either the BIOS or the OS. Comparing the two systems, the time spent in the OS is roughly 3.7s and 3.8s respectively. We can also see how quickly the hardware responds, indicating that sub-second response times are achievable. For example, our integrated system completes its

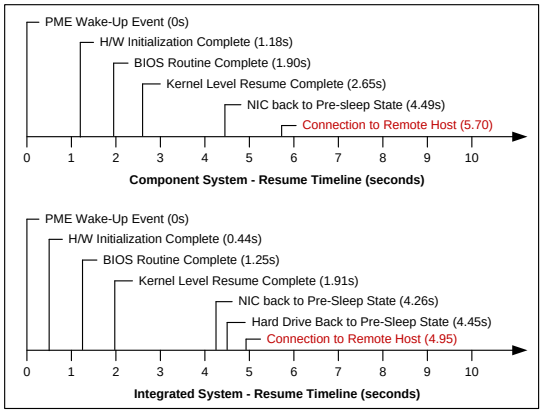

Figure 2. System Resume Timelines

hardware initialization in 0.44s, so a selective resume that does not waste any time initializing unused devices (e.g., with a BIOS implementation) could complete a memory page fetch or packet inspection operation without running the full BIOS code or starting the OS and return to sleep in under one second.

Our resume times are comparable to previous work, although our suspend times are lower [4, 5]. However, it is hard to compare these times because the hardware is different and their methodology for collecting timing was not fully specified. We plan to run these experiments on other hardware to get a more comprehensive set of results.

3.4 Cycle Time

Our next experiment demonstrates the potential cycle time improvement possible with selective resume. Cycle time improvement will benefit applications that require sporadic access to system resources, such as the remote access scenario, described in Section 2. In this scenario, if we find that the incoming packet is not worth waking the system, then we put the system to sleep as quickly as possible without initializing additional devices in the BIOS or the OS.

This experiment uses a modified version of the Linux kernel. After the first suspend, the subsequent resumes and suspends take a highly optimized kernel path designed to resume and suspend the system as quickly as possible. To measure the total cycle times, including hardware initialization times accurately, we used a digital oscilloscope and current probe connected to the common ground of the motherboard power supply. We conducted multiple runs and used average times based on the oscilloscope readings. In these experiments, the unmodified BIOS transfers control back to the kernel via the normal ACPI resume vector. We do not return control to the user space, and instead put the system to sleep as quickly as possible in the kernel.

Figure 3 shows the complete cycle times, including hardware suspend and resume latencies. Compared to the results in Figure 1 that show only BIOS and OS time, we see that the cycle time increases from 5.95s to 6.7s (0.75s) for the

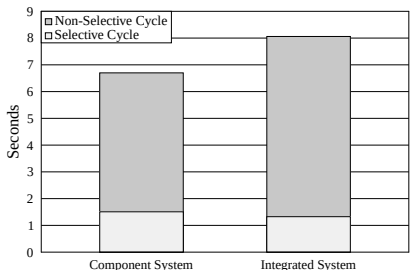

Figure 3. System Cycle Times (w/ Hardware)

component system, which is reasonable based on observed hardware initialization times, but for the integrated system the increase from 4.64s to 8.06s (3.42s) is higher than expected. Examining the oscilloscope readings, we found that for roughly 3s of the 3.42s, the hardware is not powered up. We believe that the hardware is in some transient shutdown state and does not respond to Wake-On-LAN packets during the 3s.

Our results also demonstrate that the opportunity for reducing cycle times is high, with our streamlined kernel being able to reduce the overall cycle time from 6.7s to 1.51s for the component system, a 77% reduction, and from 8.06s to 1.33s for our integrated system, an 83% reduction. Based on the oscilloscope readings, the 3s delay described above does not occur with selective resume, but unfortunately, we cannot pinpoint the reason for this difference, without more control over the BIOS. We believe that BIOS programming will also allow cycle times to be reduced much closer to the hardware times.

4. Prototype Implementation

Ideally, a context-aware selective resume implementation would enable both the firmware and OS to work together and ensure that the required devices are available and no other devices are activated. In most cases, firmware already has NIC support for the network boot (BOOTP) protocol so it could resume in response to Wake-On-LAN packets and read in the resume context information embedded within them. The firmware could determine if it should call the OS resume vector at all, as there may be cases where the context specifies a firmware application (e.g., a hard reboot or full power down) where all the processing happens without the OS being required. The firmware would also need to pass context information to the OS to inform the OS of which devices it has enabled, so that the OS knows which devices are available for its use. Otherwise, the OS may attempt to use uninitialized devices or mistakenly believe that a device has been removed from the system.

System firmware code has historically been closed source, but recently open-source alternatives such as the Coreboot BIOS [2] and Intel's TianoCore UEFI [3] will make it easier and more practical to modify firmware. Unfortunately, both of these initiatives are still in the developmental state, with limited support for modern hardware platforms and/or limited documentation. Instead, we have chosen to develop CAESAR, our context-aware selective resume prototype, in Xen. Xen provides us with a stable development environment and allows us to develop our applications, as described in Section 5, without modifying the OS. For CAESAR, the normal firmware routines run during resume before transferring control back to Xen and CAESAR. We hope to explore a firmware based selective resume implementation in the future and expect it to yield faster transition times and lower energy use.

4.1 Initiating the Resume

Our system responds to resume requests in the form of UDP packets that implement the AMD Magic Packet specification [6]. These packets are particularly useful as we can embed our resume context within the packet as described below. Upon receiving a Magic Packet the NIC triggers a hardware interrupt that causes the firmware to power up the system. The firmware then executes its normal resume code that transfers control back to Xen and our CAESAR selective resume framework.

It is possible to extend CAESAR to support other methods of initiating the resume, such as the keyboard or the CMOS timer event, as new scenarios arise. In the case of the keyboard, users could map a specific resume context to a specific keyboard scan code. This would enable a user to trigger a specific application upon resume based on the keyboard scan code from the wake-up event. In the case of the CMOS timer, a user would specify the resume context when setting the CMOS timer alarm prior to suspending.

4.2 Specifying the Resume Context

A fundamental aspect of the selective resume approach is that a system must have knowledge of the context under which it is being resumed. CAESAR makes use of a feature found on many Intel NICs which store the first 128 bytes of the Wake-On-LAN packet in a series of registers called the Wake-Up Packet Memory (WUPM) registers. These 128 bytes includes the packet header, giving us up to 108 bytes after the IP/UDP header data to store the resume context.

The Wake-On-LAN support can be configured to react to various types of incoming network traffic. We configured the system to resume upon receipt of a Magic Packet [6]. The Magic Packet is a normal UDP packet with special header information at the start of the data segment. To embed the resume context we place the data after the Magic Packet header. At a minimum a resume context contains 8 bytes. The first 4 bytes are a magic number, 0x53524d50, that identify the Wake-On-LAN packet as context-aware selective resume request. The second 4 bytes are an application specific context identifier (ID) that specify to CAESAR which application is being requested upon resume. Additionally, application specific data, such as the page frame number or sensor ID used in our applications, can also be appended to

the context after the ID. The embedding of context information within a Wake-On-LAN packet enables us to resume the system and immediately know what type of functionality is requested.

A production implementation of selective resume should also include robust security support to prevent exploitation from unauthorized users. While CAESAR does not currently implement any security mechanisms, support could be included by enabling the encryption and authentication of the resume requests through the creation of a secure channel like SSH/SSL.

```
Begin Xen Suspend Function
  Freeze Domains()
  Power Down Devices()
  sr_presuspend()
  While ( context = sr_resume() )
    Do ACPI Suspend()
    Switch ( context.id )
      Case 'App 1':
        ret = app1_resume(context)
      Case 'App 2':
        ret = app2_resume(context)
    End Switch
    if (ret == resume)
      Exit While
    sr_resuspend()
  End While
  sr_postsuspend()
  Power Up Devices()
  Thaw Domains()
End
```

Figure 4. Xen Suspend Function Modifications

4.3 Xen Modifications

As shown by the pseudo-code in Figure 4, CAESAR modifies the control flow of Xen's enter_state() suspend function, located in the power.c file, to allow the evaluation of the resume context and the selection of the appropriate resume path. The acpi_suspend() function, which handles the low-level calls to the ACPI interface for sleep and resume operations, call the sr_resume() function upon return from sleep to extract the context information specified in the Wake-On-LAN packet. CAESAR then evaluates the context information to determine which applications it should invoke upon the resume. The applications are then responsible for enabling specific device functionality that they may require.

Applications that wish to take advantage of the selective resume functionality enabled by CAESAR must implement, at a minimum, an application specific resume() function that takes as its input a pointer to the context data and returns a result that indicates if the system should return to sleep or continue along its normal resume path. Additionally, CAESAR provides a number of function hooks (sr_presuspend(), sr_resuspend(), sr_postsuspend()) to the applications, which they can implement to do any required preparation or clean-up during the course of the resume and suspend operations.

To enable the use of the Intel NIC and access to the Wake-Up Packet Memory by CAESAR, we ported the Linux e1000e network driver to the Xen hypervisor. This represented the largest and most complex portion of the system's development as portions of the PCI device handling functions and other Linux data structures and support routines had to be ported or reimplemented within Xen. By default, the Intel 82574L NIC used in our test system, when put into a low power sleep state, changes the link speed from its normal 1 Gbps to 10 Mbps, which uses less energy. The NIC renegotiates the link upon resume to return to a 1 Gbps speed, a process that typically takes 4-6 seconds. To prevent this from happening, CAESAR blocks the NIC from resetting its link state through the use of a management control register prior to sleeping. Applications that have little data to transfer may always decide to keep the NIC at a 10 Mbps link speed or decide to force a transition to 1 Gbps only when needed.

5. Applications

In this section, we discuss our usage of the CAESAR framework with two applications, a temperature monitoring system and a fast memory server application, that benefit from selective resume.

5.1 Temperature Monitoring Application

The temperature monitoring application provides the ability to remotely read a system's temperature sensor. Temperature monitoring applications are useful in data center environment allowing an operator to remotely monitor the temperature in racks without the need for additional sensors.

As laptops, tablets and other PCs continue to include a variety of sensors (e.g., GPS, camera, microphone) the ability to sporadically access local sensors in a power efficient manner becomes increasing important for applications that want to take advantage of these in-place sensor networks.

5.1.1 Temperature Monitoring Client

Our monitoring client requests temperature readings from remote systems running a temperature monitoring agent. Requests are made at a user specified frequency by sending a UDP Magic Packet [6] to the target machine that contains context information specifying the that this is a temperature monitoring request and which sensor to read (e.g., CPU or on-board). The client continues to send requests until the next reading interval. If no responses is received from the agent before the next scheduled reading the client exits with an error. Since UDP does not guarantee delivery, in addition to returning the requested temperature, we the agent also includes a sensor ID in the response. This enables the client to validate that it has received the appropriate sensor reading. After receiving a valid result, the client displays the reading and then waits until it is time for the next reading.

5.1.2 Temperature Agent - Full Resume Version

The Full Resume temperature agent runs in Dom0, the administrative domain of Xen, on a target machine and responds to incoming temperature requests from a remote monitoring client. The target machine is configured to respond to Wake-On-LAN requests. If the system is in a suspended power state when the initial request is received from the client, the target system will conduct a full resume. The system will then respond to subsequent requests received while awake. Temperature readings are taken from the motherboard's hardware monitoring chip, a FinTek f71862fg, which supports up to three on-board sensors in addition to the CPU's temperature sensor. After sending the response the target then returns into a sleeping suspend-to-RAM (S3) state by calling the "pm-suspend" function.

5.1.3 Temperature Agent - CAESAR Version

Using CAESAR, we created an agent version in Xen to take advantage of our selective resume framework. In this version, when the initial request packet triggers a resume the BIOS automatically enables the CPU and memory which then passes control to CAESAR. CAESAR resumes the NIC and checks for context information embedded in the initial request. Since the context information specifies that the request is for the temperature agent, CAESAR passes control to the temperature agent code. This code then selectively resumes the FinTek chip, takes a reading of the required sensor (as specified in the context information), and sends a response to the monitoring client. After the response is sent the temperature monitoring agent tells CAESAR to put the system back into a suspend-to-RAM (S3) state. All of these operations happen without resuming any Xen functionality or unfreezing Dom0.

5.2 Fast Memory Server

We utilized the CAESAR framework to enhance the Jettison partial desktop migration system [9]. Jettison reduces energy use of idle desktop systems, while supporting user applications that maintain network presence, by encapsulating user desktop environments within VMs, consolidating minimal state of each VM in shared servers, and placing desktops in sleep mode, whenever the VMs are idle. It consolidates minimal VM state by migrating only the VM's execution state to the server and fetching additional memory pages on-demand, whenever the consolidated VM's execution causes a fault. Jettison runs a memory server on the desktop, wakes up the desktop when a fault occurs, and subsequently returns the desktop to sleep mode if no additional fault occurs within a short interval. Because its implementation of the memory server is in the desktop's administrative OS, its resume/suspend cycles are slow, limiting the energy savings it can get. We implemented the memory server using the CAESAR framework to show the benefits of selective resume in this class of applications. This approach is also applicable to systems that implement post-copy VM migration [13] and want to take advantage of sleep opportunities between page transfers.

5.2.1 Memory Server - Full Resume Version

Jettison runs the Full Resume version of the memory server and a sleep daemon in the Dom0 of the user's desktop, whenever the VM is migrated to the consolidation server. The memory server maps the VM's frames with read-only access, and responds to network requests from the consolidation server with the corresponding page. The sleep daemon schedules desktop transitions into sleep mode. It maintains a sleep timer, used to ensure that the memory server is able to service bursts of page requests received with low inter-arrivals, whenever the is desktop awake. The timer specifies the number of seconds in which the desktop remains awake after serving a page in anticipation of the next request, and is reset whenever a new page request is received. Its use reduces the number of energy draining transitions to and from sleep mode. If no requests arrive before the timeout is reached, then the daemon puts the system into a suspend-to-ram (S3) state. In the Dom0 of the consolidation server, Jettison runs a memtap process and a wake-up daemon. The memtap process, maps the memory frames of the consolidated VM with write permissions, and listens for events sent by the hypervisor whenever the VM faults on access to a missing page. Memtap notifies the wake-up daemon, which ensures that the remote desktop hosting the VM's pages is awake, before the memtap can issue a request to the memory server. When the desktop is asleep, the wake-up daemon sends a Wake-on-LAN packet, in order to wake it up. The daemon keeps a timestamp of the last page request sent to the desktop, in order to determine its power state without sending a network probe. It tests that this timestamp has not aged by more than the sleep timer of the desktop.

5.2.2 Memory Server - CAESAR Version

The CAESAR version of the memory server runs from within Xen and is only activated upon completing a selective resume. After consolidation of the VM to a desktop and prior to suspending for the first time, the memory server uses the sr_presuspend() hook to build a memory mapping table that resolves page references to memory addresses. Upon resume the server then reads the page reference from the context information passed to it by CAESAR, looks up corresponding memory page and sends that page back to the requestor. The server continues to listen for other memory requests until a timeout occurs. After the timeout the memory server exits with a code that tells CAESAR to put the system back into a suspend-to-ram (S3) state.

5.2.3 Page Request Emulator

For repeatable evaluations we built an emulator that plays back memory traces captured during real-world usage of the Jettison system. The memory traces capture the references

to the memory pages requested by the consolidated VM with timestamps. The emulator reads in these memory traces and sends requests to the desktop, replicating the interarrival times of requests in accordance with the timestamps in the traces.

Our emulator embeds the page references in UDP request packets that adhere to the Magic Packet specification [6]. With this approach, if the desktop is sleeping when a request is made, it triggers a Wake-On-LAN resume. When a response is received, the emulator processes the next request at the time interval specified by the previously captured request trace.

Because UDP does not guarantee delivery, we ensured that our emulator could handle requests being lost because they were sent while the desktop was shutting down. We used a timeout mechanism to resend the request if no response was received within a second and we validated the page reference on incoming responses to ensure they were consistent with our original request.

6. Evaluation

In this section we discuss and evaluate the performance of our two selective resume applications and demonstrate the additional benefits that a context-aware approach provides. First, we evaluate energy savings that are possible for both the temperature monitoring application and the fast memory server application, by comparing a version of the application using CAESAR to traditional non-selective resume versions. Secondly, we provide response time and network throughput results for our fast memory server application.

6.1 Evaluation Setup & Environment

We conducted experiments on the integrated system identified in Table 1 since it had the lowest hardware resume time, but we disabled the on-board Realtek NIC since it did not support the storing the Wake-On-LAN packet contents as described in Section 4.2 and used an Intel 82574L Gigabit PCI-E NIC which does. All of our experiments utilize the ACPI S3 sleep state.

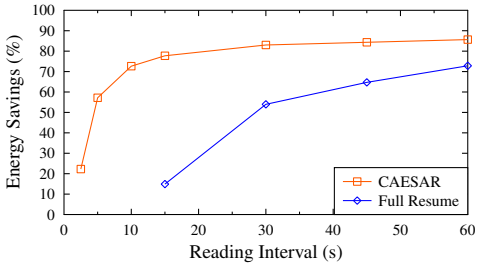

Figure 5. Temperature Monitoring Application - Energy Savings

6.1.1 Temperature Monitor Application Evaluation

We evaluated our temperature monitoring application by comparing the Full Resume and CAESAR versions described in Section 5.1. The monitoring agents were run in Dom0 using a Linux 2.16.18.8 kernel under Xen 3.4.

We evaluated the energy saving by measuring the amount of power saved by each version relative to power of running the agent on the server but never suspending it. We varied the frequency of temperature reading between 2.5 and 65 seconds as seen in Figure 5. We took measurements with intervals as low as 15 seconds for the Full Resume version, below which we were unable to get the system to reliably complete the resume→respond→suspend cycle within the 15 second interval.

At the lowest frequency schedule, once every 60 seconds, the CAESAR version demonstrated an energy savings of 85% vs. ~73% for the Full Resume version, a 16% improvement in energy savings. At the highest obtainable frequency for the Full Resume version, 15 seconds, the CAESAR version demonstrated an energy savings of 78% vs. 15%, an improvement of over 5X. In all cases, the energy saving of the CAESAR version were higher and grew as the sample frequency increased. Additionally, since the CAESAR version has quicker suspend and resume times, it can support sampling frequencies up to 2.5 seconds and realize energy savings where utilizing a Full Resume approach, with its long cycle times, could not.

6.2 Memory Server Evaluation

We evaluated our memory server application by comparing the Full Resume and CAESAR versions described in Section 5.2.

We evaluated energy savings by replaying three previously captured memory traces using the emulator described in Section 5.2.3. Figure 6 compares energy savings when desktops resume fully to serve pages, with savings obtained CAESAR invokes only the memory server. The figures show the energy savings of the desktop when its VM is consolidated over a period of one hour. In total we collected and analyzed over 90 traces then selected these three traces as representative of the volumes and page request times found in the other traces.

During the evaluation we tuned a server timeout setting, which controls how long the server waits for an additional request before it returns to sleep, to achieve the optimal energy savings for each version. To determine this value, we used a statistical analysis on our set of memory traces, in conjunction with the power profile of the system and the resume and suspend times [9]. This value ensures that the system doesn't return to sleep too early and enables optimal results for each version. We used a value of 10ms for the CAESAR version and 6s for the Full Resume version.

Figure 6(a), shows that for User 1, CAESAR increased energy savings from 17% to 28% in the first five minutes. At ten minutes, the savings increased from 28% to 44%. At twenty minutes, CAESAR increased the savings from 44% to 61%, and finally, at the end of the hour, CAESAR increased from 61% to 74%. Initially, during the first minute,

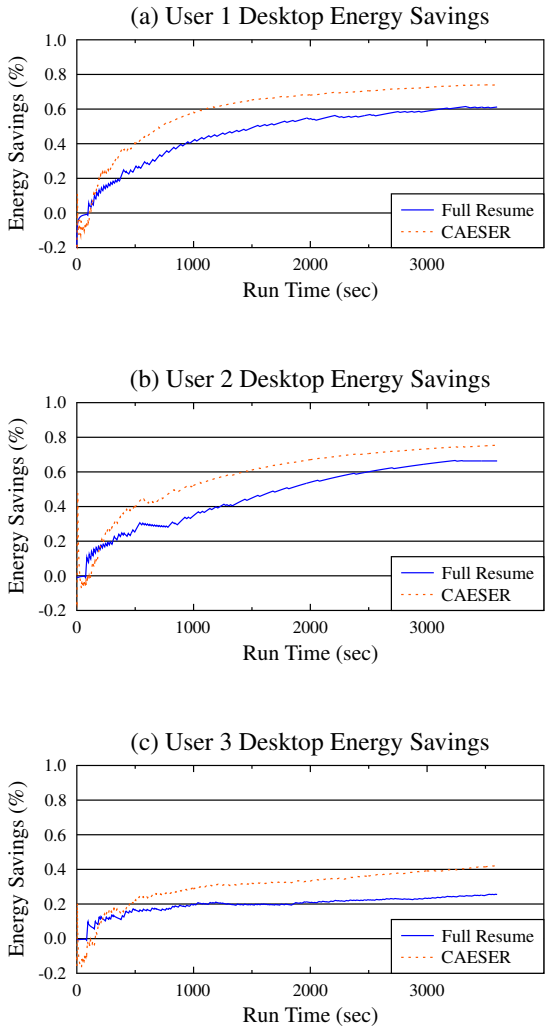

Figure 6. Memory Server Energy Savings

CAESAR leads to a loss of energy as our implementation first transitions the PC into low power, incurring the transition cost, before it can start serving pages. This initial penalty can be avoided by serving pages from the administrative domain (as done by Jettison) at the start, and transitioning to the CAESAR-based memory server only after the first sleeping opportunity.

Figure 6(b), shows that for User 2, CAESAR increased energy savings from 18% to 28% in the first five minutes. At ten minutes, the savings increased from 29% to 42%. At twenty minutes, CAESAR increased the savings from 39% to 57%, and finally at the end of the hour, CAESAR increased from 66% to 75%.

Figure 6(c), shows that for User 3, CAESAR increased energy savings from 14% to 16% in five minutes. In ten minutes it increased savings from 17% to 24%. In twenty minutes it increased energy savings from 21% to 31%. And finally over the course of an hour, savings increased from 26% to 42%. The figure shows that, while savings for User 3 are lower across the consolidation, CAESAR is still able to improve savings significantly. Across users, CAESAR delivers improvements of 13% to 66% over five minutes, 43% to 54% in ten minutes, 38% to 49% in twenty minutes, and 14% to 65% in an hour.

Comparing the Full Resume and CAESAR versions, as shown in Figure 6, we see that the improvements offered by CAESAR are largest early on in the consolidation run time. This results from a large amount of sporadic requests as the system faults on a few pages frequently. In all cases the CAESAR version continued to deliver improved energy savings as it is able to exploit more opportunities to sleep between page requests as result of its lower latency and higher throughput.

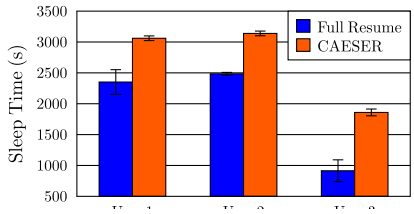

Figure 7. Jettison Sleep Times Results

6.2.1 Sleep Times

Shorter cycle times increase total sleep the desktop can perform during consolidation. Sleep time refers the time in which the desktop is in low power mode while its VM is consolidated. Figure 7 shows increases in desktop sleep times with the use of CAESAR, over an hour long consolidation of the VMs for the three users. Sleep times aggregate all intervals of at least one second in which the desktop used less than 8 W of power. The figure shows that, for User 1, CAESAR increases total sleep time from 39.19 minutes to 51.01 minutes. For User 2, it increases total sleep time from 41.41 minutes to 52.32 minutes. And finally, for User 3, CAESAR increases total sleep time from 15.24 minutes to 30.98 minutes. Across users, CAESAR increases sleep times by 26% to 103%, during the hour-long period of consolidation.

Version	Response Time	Max. Throughput
Full Resume	5,865 ms	22 MB/s
CAESAR	1,487 ms	34 MB/s
Always On	0.2 ms	22 MB/s

Table 2. Latency & Throughput Results

6.2.2 Resume Latency

We evaluated response time latency by measuring the time it takes for a system to successfully receive a requested page from our memory server. This represents the total time it takes to send the initial request to the server and receive the complete response from the server including all network,

system and resume time overhead. We conducted 45 runs and averaged the results across all the runs. The results are shown in Table 2. They demonstrate the effectiveness of our selective resume approach in reducing the overall resume latency. The response time for our CAESAR implementation is almost 4X the Full Resume version at resuming and responding to a request.

6.2.3 Throughput

We evaluated both overall throughput, which includes resume latency, and maximum throughput which shows the highest speed at which pages could be sent.

We measured the maximum throughput that each implementation could achieve by calculating the time it took to request ~256M worth of pages from an awake and active memory server. As shown in Table 2, our Selective Resume implementation was able to reach a maximum throughput speed of over 34 Megabytes/second, which was ~55% faster than the 22 Megabytes/second achieved by the other versions, including the Always On version. This increased performance is a result of our CAESAR version not having the operational overhead created by having the complete OS resumed.

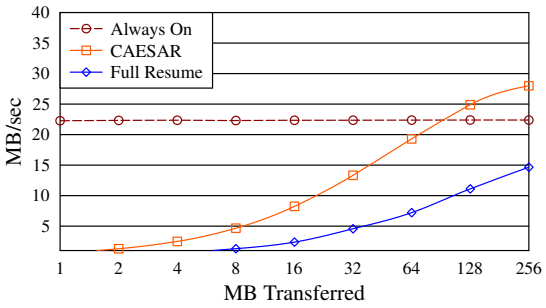

Figure 8. Overall Throughput Results

We evaluated the overall throughput by measuring the time it takes for a system to successfully receive a set of pages from a suspended memory server. This represents the total time it takes to send the initial request to the server and receive the complete response from the server including all network, system and resume time overhead. We conducted the tests by requesting a set of between 1 and 256 megabytes of pages and did five separate runs and averaged the results across all the runs for a given set size.

The results shown in Figure 8 demonstrate the effectiveness of selective resume in increasing overall throughput as a result of providing both increased maximum throughput and reduced first response latency. Interestingly, with request sizes of more than ~95 MB, our CAESAR implementation outperforms even the always on system because the higher maximum throughput our CAESAR version achieves outweighs the short resume latency it introduces during the resume phase.

7. Related Research

While ACPI sleep states have been available on systems for many years, prior research to take advantage of them has focused primarily on avoiding unnecessary resumes. These approaches have, in part, been driven by the lack of flexibility and high latency of resume transitions, and they all require system augmentation with external servers or embedded systems. In contrast, our research seeks to remove the underlying problem by making sleep states more flexible by having them designed into the system themselves.

Most similar to CAESAR is Turducken [18] which utilizes tiers of lower power hardware to respond to resume requests and only resumes higher tiers as required by the resume context. In contrast, our approach increases sleep times and provide the illusion of being online without requiring additional hardware or changes to applications.

Many systems have been designed to provide the illusion of on-line availability, by building a network proxy or specialized hardware that can filter and/or respond to incoming network requests [4, 5, 15, 17]. The proxy checks incoming requests and only resumes the system for those requests that meet specific criteria. These approaches all require an external proxy server or hardware to be effective and often require application modifications.

Other research has focused on taking advantage of sleep states by offloading processing. For example, virtualized desktops can be offloaded to a separate server [9, 11]. Brakmo et al. [10] propose opportunistic use of short sleeps, that are imperceptible to the user, to save energy of portable devices. Our work complements both these approaches by providing fast fine-grained access to resources as they are needed.

8. Conclusions

We have made the case that the current resume functionality does not have the flexibility to meet the needs of modern applications that require sporadic access to system resources. We have shown that resume latencies are mostly dominated by software and can be reduced significantly. We have introduced CAESAR, a framework for utilizing context-aware selective resume. We have implemented two applications and demonstrated that our selective resume approach can respond up to 3.9X faster and improve energy savings by over 5X. Additionally, we show that selective resume enables improved cycle time enabling us to realize energy savings in applications where non-selective approaches cannot.

Ultimately, we expect that context-aware selective resume functionality will enable new research that can take advantage of the greatly reduced cycle times and additional power savings. If we can reduce the cycle times below an application's or user's ability to perceive them then we have effectively created an always-on system that consumes almost no power.

References

[1] ACPI, advanced configuration and power interface specification, revision 4.0. http://www.acpi.info/DOWNLOADS/ACPIspec40a.pdf.

[2] Coreboot project - http://www.coreboot.org/.

[3] Tianocore project - http://sourceforge.net/projects/tianocore/.

[4] AGARWAL, Y., HODGES, S., CHANDRA, R., SCOTT, J., BAHL, P., AND GUPTA, R. Somniloquy: augmenting network interfaces to reduce PC energy usage. In *Proceedings of the USENIX symposium on Networked systems design and implementation (NSDI)* (2009), pp. 365–380.

[5] AGARWAL, Y., SAVAGE, S., AND GUPTA, R. SleepServer: a software-only approach for reducing the energy consumption of PCs within enterprise environments. In *Proceedings of the 2010 USENIX annual technical conference* (2010).

[6] AMD. Magic packet technology - http://support.amd.com/us/embedded_techdocs/20213.pdf.

[7] ASUS. Asus zenbook ux21e product website. http://asus.com/Notebooks/Superior_Mobility/.

[8] BILA, N., DE LARA, E., HILTUNEN, M., JOSHI, K., LAGAR-CAVILLA, H. A., AND SATYANARAYANAN, M. The case for energy-oriented partial desktop migration. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing* (Berkeley, CA, USA, 2010), HotCloud'10, USENIX Association, pp. 3–3.

[9] BILA, N., DE LARA, E., JOSHI, K., LAGAR-CAVILLA, H. A., HILTUNEN, M., AND SATYANARAYANAN, M. Jettison: Efficient idle desktop consolidation with partial vm migration. *To appear in Proceedings of the European Conference on Computer Systems (EuroSys2012)* (2012).

[10] BRAKMO, L. S., WALLACH, D. A., AND VIREDAZ, M. A. usleep: A technique for reducing energy consumption in handheld devices. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications and Services (Mobisys 2004)* (Boston, MA, USA, Jun 2004).

[11] DAS, T., PADALA, P., PADMANABHAN, V. N., RAMJEE, R., AND SHIN, K. G. LiteGreen: saving energy in networked desktops using virtualization. In *Proceedings of the USENIX annual technical conference* (2010).

[12] DOGAR, F. R., STEENKISTE, P., AND PAPAGIANNAKI, K. Catnap: exploiting high bandwidth wireless interfaces to save energy for mobile devices. In *Proceedings of the international conference on Mobile systems, applications, and services (MobiSys)* (2010), pp. 107–122.

[13] HINES, M. R., AND GOPALAN, K. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments* (New York, NY, USA, 2009), VEE '09, ACM, pp. 51–60.

[14] LU, Y., LO, L., WATSON, G. R., AND MINNICH, R. G. CAR: using cache as RAM in LinuxBIOS, 2006.

[15] 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 *Proceedings of the 6th USENIX symposium on Networked systems design and implementation (NSDI)* (2009), pp. 381–394.

[16] SATYANARAYANAN, M., BAHL, P., CACERES, R., AND DAVIES, N. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing 8* (October 2009), 14–23.

[17] SHIH, E., BAHL, P., AND SINCLAIR, M. J. Wake on wireless: An event driven energy saving strategy for battery operated devices. In *Proceedings of the 8th Annual International Conference on Mobile Computing and Networking (MOBICOM '02)* (Atlanta, GA, USA, Sep 2002).

[18] SORBER, J., BANERJEE, N., CORNER, M. D., AND ROLLINS, S. Turducken: Hierarchical power management for mobile devices. In *Proceedings of the 3rd International Conference on Mobile Systems, Applications and Services (Mobisys 2005)* (Seattle, WA, USA, Jun 2005).