Agile, Efficient Virtualization Power Management with Low-latency Server Power States

Canturk Isci* Suzanne McIntosh* Jeffrey Kephart* Rajarshi Das* James Hanson*
Scott Piper† Robert Wolford† Thomas Brey† Robert Kantner† Allen Ng†
James Norris* Abdoulaye Traore* Michael Frissora*

*IBM T.J. Watson Research Center
Yorktown Heights, NY

†IBM Systems & Technology Group
Raleigh, NC & Kirkland, WA

ABSTRACT

One of the main driving forces of the growing adoption of virtualization is its dramatic simplification of the provisioning and dynamic management of IT resources. By decoupling running entities from the underlying physical resources, and by providing easy-to-use controls to allocate, deallocate and migrate virtual machines (VMs) across physical boundaries, virtualization opens up new opportunities for improving overall system resource use and power efficiency. While a range of techniques for dynamic, distributed resource management of virtualized systems have been proposed and have seen their widespread adoption in enterprise systems, similar techniques for dynamic power management have seen limited acceptance. The main barrier to dynamic, power-aware virtualization management stems not from the limitations of virtualization, but rather from the underlying physical systems; and in particular, the high latency and energy cost of power state change actions suited for virtualization power management.

In this work, we first explore the feasibility of low-latency power states for enterprise server systems and demonstrate, with real prototypes, their quantitative energy-performance trade offs compared to traditional server power states. Then, we demonstrate an end-to-end power-aware virtualization management solution leveraging these states, and evaluate the dramatically-favorable power-performance characteristics achievable with such systems. We present, via both real system implementations and scale-out simulations, that virtualization power management with low-latency server power states can achieve comparable overheads as base distributed resource management in virtualized systems, and thus can benefit from the same level of adoption, while delivering close to energy-proportional power efficiency.

1. INTRODUCTION

Over the last decade, reducing the energy consumption of computing devices in data centers has become a topic of great practical and academic interest [24, 41]. Myriad techniques for reducing energy consumption have been introduced at all levels of the stack, from circuits and architecture levels up through the firmware, operating system, and the middleware.

Among these techniques, those which employ virtualization have been shown to yield significant energy efficiency improvements. By consolidating multiple workloads onto a single physical server, virtualization drives up system utilization and enables a given amount of computational work to be performed on a smaller set of servers. While a more heavily utilized server uses more power than a lightly utilized one, the total power consumed by running a workload on a smaller number of more heavily-utilized server can be substantially less than that for a large number of lightly-utilized servers.

One important class of virtualization-based energy efficiency methods is *static* consolidation. The fundamental idea, which arose from the observation that a large fraction of dedicated physical servers are severely underutilized [3], is based upon capacity planning. Long-term statistics on resource consumption for a given set of workloads are used to estimate how many virtualized physical servers would be needed to support that computational load [8, 36]. This approach is commonly used for migrating workloads from dedicated physical servers to virtualized environments [30, 43, 44], although the original environment could also be virtual. VM migration may be employed from time to time to even out the workload across the physical servers, but no further changes are made to the power state of the servers. Thus, while static consolidation eliminates substantial amounts of overprovisioning, it does not take advantage of dynamic fluctuations in workload intensity [31, 39].

A second important class, *dynamic consolidation*, attempts to extract further efficiency by migrating VMs to vacate some physical servers during lulls in workload intensity, turning them off to save energy until the workload increases [16, 25, 42, 46, 40]. Dynamic consolidation techniques promise to yield substantial energy efficiency improvements on top of what is achievable by static methods. However, while they have gained some traction in certain desktop virtualization frameworks [10, 5], in practice they do not make good on that promise at the enterprise level. This is because mechanisms for turning servers on and off entail latencies that can extend to minutes. The longer the latency, the greater the risk that the workload will increase and the performance will suffer for the period during which the increase is detected and addressed by turning on enough servers.

Previously-explored methods for mitigating the risk caused by the latency of coarse-grained power management include workload scheduling, workload forecasting, and keeping extra servers on as a buffer to protect against unanticipated

ISCA'13 Tel-Aviv, Israel

workload surges. All of these approaches have drawbacks. Workload scheduling is feasible only when one has substantial control over the workload (for example in a compute grid scenario). Workload forecasting is difficult to apply because no workload is completely predictable, and the penalty for suffering an unanticipated surge—even if such errors are rare—can be substantial. Therefore, even when one uses workload forecasting, some amount of buffering is necessary, and this can largely wipe out the potential energy savings.

In this paper, we demonstrate a radically different approach that overcomes the traditional barriers to achieving substantial energy savings via dynamic consolidation. Rather than merely treating the symptoms, we attack the underlying root cause or the problem: the latency itself. Our method exploits low-power, low-latency power states in enterprise servers; specifically, the S3 state. We first present an across-the-stack implementation of S3 capability for enterprise servers with two actual prototypes. Then, we build an end-to-end virtualization management solution, leveraging this technology. Our experimental results with our real-system prototypes and scale-out simulation framework show substantial energy efficiency improvements over standard techniques. These further underline two key aspects of our approach with low-latency server power states: the ability to react to fine-grained load fluctuations that most existing solutions cannot handle with standard power states; and reduced impact from incorrect power management decisions with substantially faster reaction times. While the basic principle of our solution is simple, realizing this principle in practice entails substantial work all along the stack from the hardware level up through the firmware, the OS and the middleware used to control the VM migration and manage the power states of the physical servers.

The remainder of this paper is organized as follows. Section 2 provides more background on the fundamental idea underlying our solution and highlights its energy saving potential. Section 3 presents a power state characterization study, establishing that the S3 state offers an excellent tradeoff between latency and power consumption. Section 4 details how we modified a commercial server to support S3 at the hardware, firmware and OS levels, while Section 5 describes middleware that we developed for power-aware virtualization management. Sections 6 and 7 describe our experimental results. We describe related work in Section 8, and summarize our results in Section 9.

2. BACKGROUND AND MOTIVATION

How can reducing the latency of power management operations increase the energy savings realized by dynamic consolidation? To develop some intuitive understanding, we sketch two scenarios in Figure 1.

The top scenario (Figure 1(a)) represents present-day dynamic power management approaches for virtualized systems that employ standard high-latency server power-on and power-off actions triggered by Wake-on-LAN or IPMI. VMware Distributed Power Management (DPM) [46] is a prime example of such technology. In this scenario, the cluster demand (dashed line) starts high, dips low for a while, and then returns to its original high level. During the period of lower demand, the power management (PM) algorithm identifies a server that could be evacuated and turned off. As can be seen by comparing the power consumption profile for that server (solid line) to the cluster demand, there are three distinct problematic time periods (cross-hatched

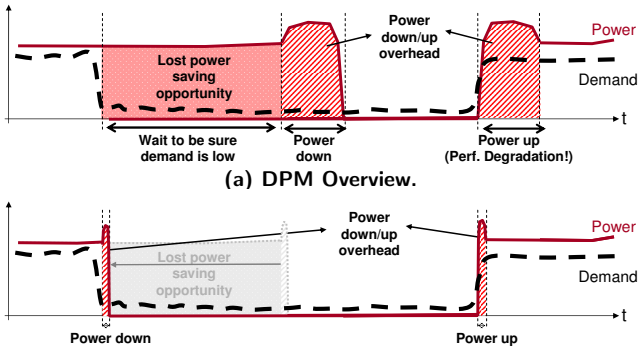


Figure 1: The effect of power state change overheads on energy-aware virtualization management.

areas) during which power is wasted and/or performance is degraded:

- **Power-on latency**. In the rightmost region, the PM algorithm detects that the cluster demand has risen, so it initiates a power-on action. As the server powers on, power consumption is high—potentially even higher than when the server is processing the workload. Worse, the performance of the cluster as a whole suffers until the boot-up is complete and work has migrated back to the server.
- **Power-off latency**. In the central region, the power consumed as the server is being turned off is used to place the server in a stable state rather than being used to process the workload.
- **Decision period**. A PM algorithm must be judicious in its response to lower demand. If it is too hasty, and the reduction in demand proves short-lived, the performance cost during power-on latency will outweigh any energy savings that may have accrued during the transient. To reduce the risk of regret, it is reasonable for the PM to adopt a more conservative strategy, according to which it waits for a specified time period before turning the server off. Yet this wait time (which may reasonably be set to the better part of an hour) can result in significant lost opportunity for power savings, and makes it impossible to save any power if the duration of the low demand is less than the wait time.

The bottom scenario (Figure 1(b)) represents the same situation, the only difference being that the power-up and power-down latencies are assumed to be very much smaller. (The actual technological basis for latency reduction will be discussed later.) One important effect is that performance penalty during the power-on period is reduced greatly from that of the top scenario. Even more importantly, to the extent that the performance penalty is reduced, the PM algorithm is now free to be much less risk averse, i.e. it can reduce the wait period to a much smaller value (zero in this idealized figure). The reduction in power-off latency is an extra bonus.

The key intuition is that, due to direct and indirect effects, substantial reductions in power-on latency allow the power profile to follow the demand profile much more closely, reducing both performance degradation and power consumption.

This said, it is important to acknowledge that there are two additional latency components missing from the simple story depicted in Figure 1:

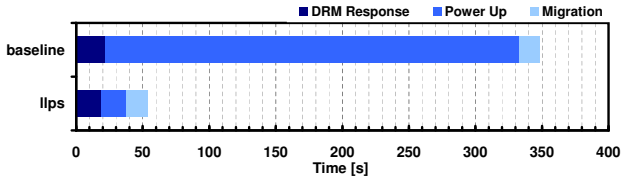


Figure 2: Decomposition of the demand response to the three orthogonal dimensions.

- **VM migration time** captures the time to migrate VMs across servers when a host is evacuated before a power down or is repopulated after a power up. Therefore the migration latency is added to both the power-down and power-up paths. Moreover, the associated extra impact on power and performance due to resources consumed by the act of migration must be considered in power-aware virtualization management.
- **DRM (Distributed Resource Management) response time** captures how quickly the virtualization manager detects a change in workload profile and decides on the reconfigurations needed in VM topology and host power states. In most cases, the resource managers operate at user-defined periods; for instance, both VMware DRS/DPM and IBM VMControl operate at periods of several minutes [45, 21].

If these two additional latencies were to outweigh the power-on latency, reducing the power-on latency would have little practical import, and the work described in this paper would be a mere academic footnote. Fortunately, in the real prototype that is described in detail later, our measurements (depicted in Figure 2) establish that the measured power-on latency for current dynamic consolidation techniques dominates the other two forms of latency ("baseline"). As we shall show, our low-latency power savings method, depicted as "llps", reduces this latency considerably, to the same order of magnitude as the other two. In both cases, the total size of the two bars represent the total time the virtualization resource manager takes to respond to a demand surge, and the different colored regions represent the amount of time consumed by each of the components (DRM response, power-up latency, and migration time). This example experiment underlines the two critical observations that drive our work: (i) in current virtualization power management techniques power latencies are the dominant bottleneck; and (ii) improving server power state latencies can improve virtualization power management substantially.

In the following section we present a detailed quantitative evaluation of the low-latency server power states, and discuss their specific characteristics based on real system evaluations.

3. LOW-LATENCY SERVER POWER STATES

In this section, we overview briefly the set of power states into which physical servers may be placed, and then present experimental measurements of the power consumption and transition latencies for these states on different platforms and under different load conditions. These results establish that the `S3` (suspend-to-RAM) power state provides a very attractive tradeoff between power consumption and latency.

The ACPI (Advanced Configuration and Power Interface) specification [17] defines a wide and growing range of power states for physical servers of all microarchitectures at various levels of the system hierarchy. These power states are categorized as *Global System States* (G-states), *Sleep States* (S-states), *Processor Power States* (C-states), *Processor Throttle States* (T-states), and *Processor Performance States* (P-states). In addition to these, ACPI also defines special power states for system devices as *Device States* (D-states). Each of these states can assume multiple levels, wherein a higher level translates to a lower power mode. G-states refer to entire platform-level states and range from `G3` (representing the mechanical off state) to `G0` (the working system state). `G2` (also equivalent to the `S5` halt state) is the soft off state, which consumes the bare minimum of power required to support base management components such as the service processor and the Wake-on-Lan path. `G1` is the global sleep state, which is further divided into different S-states: `S1` to `S4`. In `S4` the memory state of the system is persisted to disk, and the system is put into hibernation, turning off all components, similar to `G2`. `S3` is the suspend-to-RAM state, during which most of the system components are powered down, while memory is preserved via a lower-power self-refresh mechanism, thereby supporting very quick suspend-and-resume cycles. `G0` and `S0` together define a working platform state, at which a broad and continually evolving range of C-states are defined [33, 47]. P-states and T-states are sub states of `C0`: P-states define dynamic voltage and frequency scaling (DVFS) steps, while T-states define clock throttling rates, commonly used in response to thermal events [37, 7].

There is extensive literature on the comparative power-performance characteristics of C-, T- and P-states [11, 27, 26, 34] and their application to dynamic power management [1, 29, 18, 22]. They are attractive in that their latencies are negligible for most practical purposes (milliseconds in most cases), and they can save appreciable amounts of power. However, because they all occur when the system is in a state where all of the main subcomponents are turned on, the inherent power savings opportunities are not nearly as strong as they are for the S-states. On the other hand, much less is known about the comparative power-performance characteristics of S-states, and there are some misconceptions about the efficiency and reliability of these states and their suitability for dynamic, distributed power management in enterprise servers.

Accordingly, we set out to study the power and latency characteristics of S-states on two server platforms: IBM *HC10* and *HS22* servers. The *HC10* has a blade form factor with a workstation board supporting `S3` and `S4`. Experimental measurements on this first prototype demonstrated the surprisingly good latency-power characteristics of `S3`, which convinced us to invest substantial effort in developing a second prototype based on a true enterprise server—the *HS22* platform. We implemented `S3` and `S4` support on an experimental HS22 blade, as described more fully in the next section. There also exists a new set of *connected standby* (`S0ix`) states, recently introduced by Intel, driven by mobile platform requirements [32]. While not yet exercised on servers, their potential latency-power characteristics warrants further investigation of their adoption in server systems.

Figure 3 shows the power consumption for the base HC10 configuration and two HS22 systems with different memory sizes. The HS22 and HC10 were running RedHat Enterprise Linux (RHEL) and Fedora Linux respectively, and both platforms employed the KVM hypervisor. All power

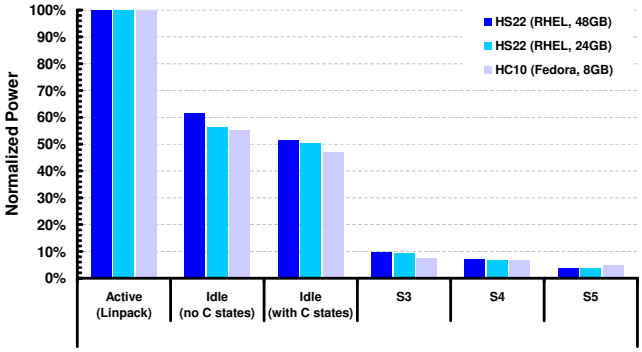


Figure 3: Power consumption of different server power states.

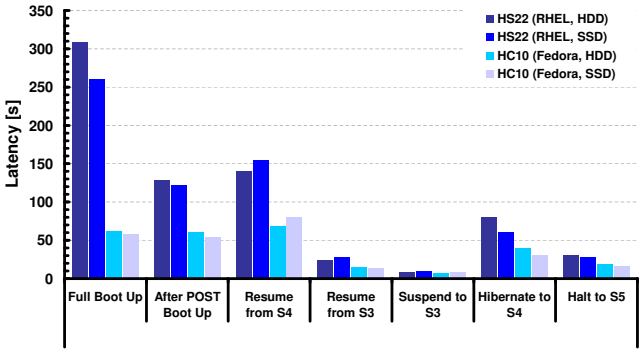


Figure 4: Latency characteristics of different server power states.

measurements were collected from the board service processor. First, we measured the peak power consumed while running the `Linpack` benchmark across all hardware threads of the systems, using this to normalize a set of measurements on an idle system, an idle system with C-states enabled, and a system placed in the `S3`, `S4` and `S5` states.

A consistent picture emerges across all configurations. In all cases, idle systems consume about 50% of the peak power. When enabled, C-states reduce the idle power consumption by about 10-15%, which is significant but pales in comparison to the S3, S4 and S5 states, which reduce the idle power consumption by 85% to over 90%. Importantly, there is a relatively small difference among the power savings achievable by different S-states. That is, `S3` saves almost the same level of power as a shut down (`S5`) system (recall that `S5` is a soft off state, so it still consumes some power).

Figure 4 depicts the other half of the tradeoff: the power-on and power-off latencies. We measured latency characteristics for both platforms, using both standard server disks (HDD) and solid-state drives (SSD), resulting in four different configurations that are reported in seven groups of results. The first four groups report the power-on latencies for `S5` (both full boot up and OS portion of the boot-up, shown as *after power-on self test (POST)* boot time), `S4`, and `S3`, while the last three groups report the power-off latencies for `S3`, `S4` and `S5`.

Again, a consistent story emerges across all configurations. Resume times from `S3` can be an order of magnitude better than those with `S4` or `S5`. Similarly, the power-off times for `S3` are significantly better than for `S4` and `S5`, which is plausible because entry to `S3` requires only a tiny amount of state saving into memory. Taken together, Figures 3 and 4 demonstrate that, relative to `S5` (which is the basis for DPM and other present-day solutions) and `S4`, `S3` provides dramatically lower latency at comparable power savings, and can therefore serve as a basis for heretofore unrealized energy savings.

Finally, we conducted further experiments to characterize how the power-on and power-off transition times for the various S-states depend upon system configuration and load. Three memory configurations were explored: 1GB, 2GB and 8GB. For each of these three memory configurations, we ran a server at idle, at high CPU load, and at high memory load. The top row of Figure 5 depicts the measured power consumption for the three idle runs. In each case, we repeatedly pinged the systems to detect when they were or were not operational; the operational time periods are indicated by the *ping response* markers.

When a resume request is received by a server, the power profile immediately switches from low to high. The gap between the point at which the power curve jumps up and when the ping response markers reappear on the curves provides a direct measure of the power-on latency of these systems. As shown in the figure, `S3` resume remains around 10s across all memory configurations, while the latency for `S4` grows with memory size, from 50s at 1GB to 220s at 8GB. Moreover, and not surprisingly, `S4` consumes more power during the transition than `S3`. (Note the similarity to the shaded *power-up overhead* region in Figure 1.)

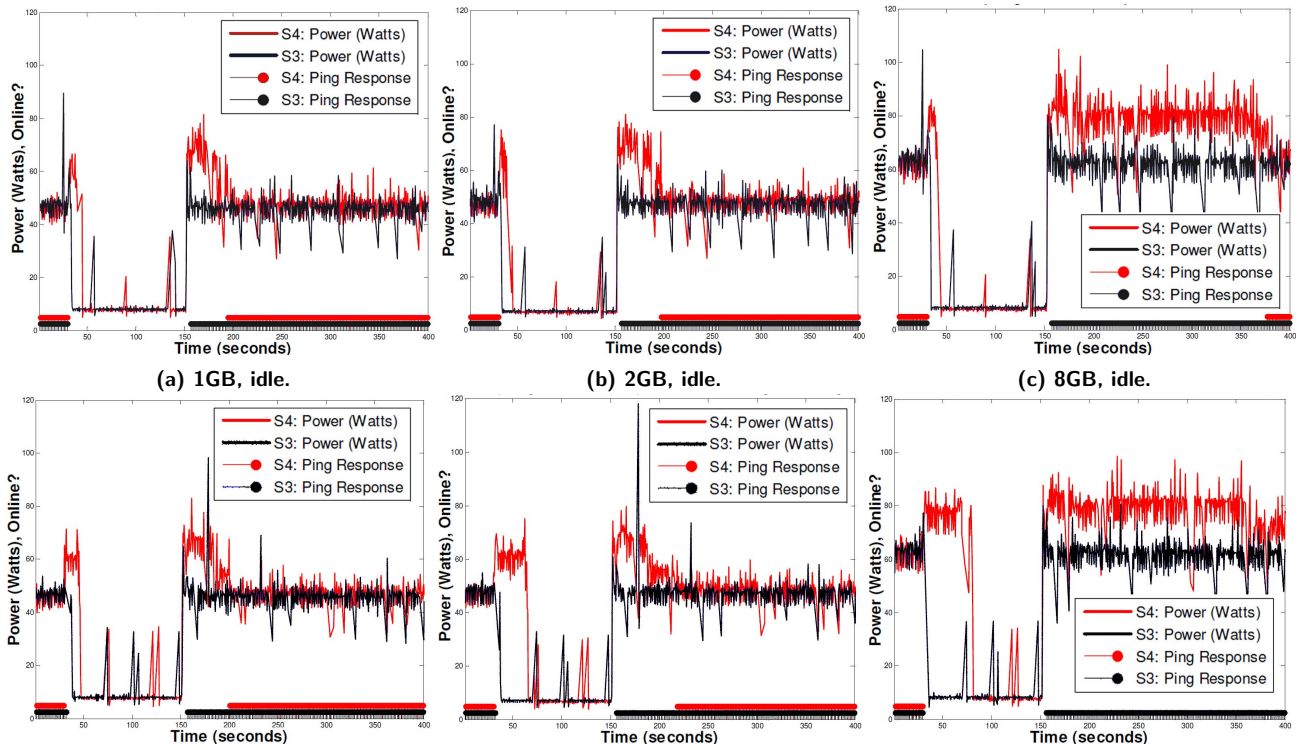
The experiments in which we ran the systems at 100% CPU load and small memory footprint (not shown for reasons of space) tell a similar story, although the overall power consumption is of course greater during the period when the system is being utilized. More interesting is the result when we run the system at small CPU load but high memory workload, shown in the bottom row of Figure 5. The power-on latency for `S3` remains at the same value as in the idle case, but for `S4` the power-on latency grows further beyond what was observed in an idle system. At 8GB, the power-on latency for `S4` was measured at greater than 250s.

These results further validate that `S3` is a superior power state for dynamic virtualization power management. Not only are the power savings nearly as high as for `S4` and `S5`, but the latency characteristics are strongly superior both in terms of being much lower and in terms of being independent of system CPU and memory load.

4. IMPLEMENTING SERVER S-STATES

Desktop workstations and laptops have supported `S3` and `S4` for many years. In fact, `S3` is the basis for the familiar "stand-by" power-down option on laptops. However, manufacturers had never thought it important to support these power modes on enterprise servers that run substantive workloads in data centers, as they never had to run on battery power. Thus the only power-down option on enterprise servers has been `S5`. In recent years, as interest in power management has increased, manufacturers have enabled a growing set of C- and P-states in servers, but as we have seen these only offer savings in the range of 10-15%.

To substantiate our claim that dynamic virtualization power management based on `S3` is practical in real systems, we set ourselves the task of enabling `S3` on a real enterprise system. As described in the previous section, we had already conducted encouraging measurements on an HC10 system, which is based on a workstation board in blade server form factor. In this section, we describe modifications that we had to make to the hardware, firmware, and OS to support



(a) 1GB, idle. (b) 2GB, idle. (c) 8GB, idle.

(d) 1GB, high memory load. (e) 2GB, high memory load. (f) 8GB, high memory load.

Figure 5: Power overhead and response time comparison for S3 and S4 with different memory sizes.

S3 on our experimental HS22 blade server.

Hardware: Three unique sets of hardware needed to be added to support S3:

- **Power sequencing circuitry**. When a server enters S3 state, the sequence to shut down the power rails is identical to that of powering down a server (entering S5 state), except that the power supply for memory has to be left in the on state. This is necessary as the server state is stored in memory during S3. This augmentation of function requires additional power rail switches.
- **System management hardware**. This piece of hardware processes the high-level requests from external entities to enter or exit S3 or S4 and reports the current state of the server.
- **Handshake signals.** The additional power rail switches require associated control signals. Handshake signals connecting the core motherboard chips to the system management subsystem are also necessary in order to determine when a request to enter or exit S3 is active and also to get confirmation that the server has actually entered the S3 state.

Firmware: One layer up from the hardware layer is the firmware layer. It must support S-states at three key times: when the server boots up, during an S3 suspend operation, and during S3 resume:

- **Boot-up sequence**. As part of the normal boot-up sequence, the firmware performs several tasks in support of S3 functionality. It initializes S3-related registers, it saves system, memory, and chipset configurations into two tables in non-volatile memory (NVM), and reserves a portion of memory for temporary use during the S3 resume process. The advantage of storing configuration state information during normal boot is that the time required to enter S3 sleep is minimized.
- **Suspend**. During the suspend process, S3 supporting devices are transitioned to the D3 state, and the waking vector address is saved.
- **Resume**. The firmware must retrieve configuration data from the tables in NVM and write them back to the registers and the chipset. Once the chipset state has been restored, control is passed to the operating system.

Operating System: At first glance, one may expect that only the software closest to the hardware, the firmware, must be S3/S4-aware. However, we have seen that device drivers, and some configurable OS functions must also cooperate in the state transition flow to support graceful suspend and resume during S3/S4 cycles.

- **Device drivers**. Device drivers run as part of the resident operating system. Some, such as fiber channel, did not fully implement the D-state specifications, causing the system to not resume properly. We added driver modifications to provide proper D-state support where possible, and unloaded some of the other non-essential drivers from the system.
- **Changes to device operation**. In other cases in which devices did not resume properly, we added workarounds to gracefully disable the service, and unload the device state. We added hooks on resume to reload the device and initialize the service.
- **Modifications to OS hooks**. We reconfigured some of the steps (*hooks*) of the OS suspend-and-resume flow

to enable proper server console startup on resume. As some of the network devices did not completely support S-states, they occasionally caused the entire resume process to fail, leaving the server in an inoperable state. We added stateless shutdown and cold reinitialization logic on the suspend and resume paths in the OS for these devices.

5. POWER-AWARE VIRTUALIZATION MANAGEMENT PROTOTYPE

Here we describe our end-to-end power-aware virtualization management prototypes that we designed to leverage low-latency server power states across the two generations of our S3-enabled experimental server hardware. The fundamentals of the two prototypes going up the OS and middleware stack are conceptually similar. Therefore, our discussion here focuses on the final HS22-based implementation. Figure 6 shows the overall architecture of our end-to-end system, consisting of three distributed components: hosts, management module(s) and the *global virtualization manager*.

The global controller is the center piece of our dynamic, distributed virtualization power management framework. It interfaces with all the physical and logical components to perform a wide range of tasks for virtualization management and sensing/actuation methods for managing the power states of physical hosts. The global controller interfaces with each hypervisor host to determine the VM inventory and topology in the cluster, the configurations, performance and resource requirements of each VM. The core algorithm of the controller is the *DRM policy*, which determines the dynamic VM placement decisions and the host power actions. After each policy execution, the controller decides on three sets of potential actions: host power up, VM migration, and host power down. These are carried out in a specific order, and across different paths. First, host power-up actions are carried out as the hosts must already be online for the VMs to migrate onto them. This is done through an out-of-band path, by talking to the server service processors. In our BladeCenter form factor, we accomplish this via programmatically exercising the BladeCenter Management Module APIs [6] to turn on specified servers. Second, the placement actions, i.e., VM migration decisions, are applied across the set of source and destination hypervisors with shared network-attached storage. The controller performs these by coordinating the two involved hypervisors for each migration to: (i) create the destination listening VM, (ii) to iteratively migrate live state, and (iii) to stun and terminate the source VM to finalize the migration. All migrations are completed before the host power down actions, because all VMs must be evacuated before hosts are powered down. Once the VM migrations are completed, the controller executes the third and final step of the management algorithm by powering down evacuated hosts. This is done in an in-band fashion, where the controller interfaces with the host agents to put hosts into lower power states. There have also been, however, recent efforts to introduce out-of-band paths to push hosts into lower S-states.

The hosts in our virtual infrastructure prototype are based on IBM HS22 blade server configuration with custom modifications for experimental S-state support. These modification include reworked motherboard hardware, custom non-production firmware and additional OS-level configurations to support repeatable `S3` and `S4` cycles. The host systems

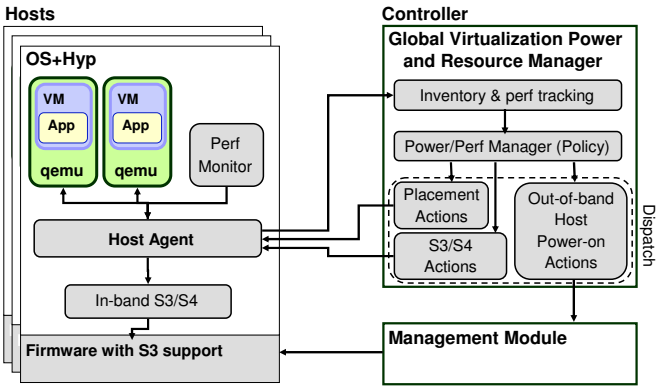


Figure 6: Prototype implementation.

run RedHat Enterprise Linux (RHEL) 6 with KVM/QEMU based virtualization support [23]. We developed a *host agent* and a *perf monitor* agent that are deployed in each hypervisor host to enable virtualization management and VM performance monitoring functions distributed across the hosts. The host agents provide a network interface, which the global controller uses to communicate and track each hypervisor system. The perf monitors are driven by the host agents to track performance and resource demand characteristics of the running VMs on the same system. The perf monitors track the dynamically-varying resource requirements of the running VMs and any resource contention they experience. We use raw QEMU and Linux interfaces to carry out virtualization-related tasks. These include VM inventory discovery, VM configuration, VM start/shutdown, live migration, creating listening VMs, configuring migration, checking migration progress. While `libvirt` abstractions are commonly employed in current KVM-based systems, we did not utilize libvirt as some of the QEMU migration configurations we used had not been yet exposed through libvirt. The host agents are also the gateways for triggering in-band suspend actions for pushing the systems into lower S-states. When the controller requests a system to be powered down, the host agent passes the request to the appropriate Linux handler, invoking the standard shutdown path for `S5` requests, and by using the Linux `pm-utils` for `S3` and `S4` requests [2].

In our experimental prototype, the hosts are primarily used as hypervisors, hosting VMs, where the VMs are the only significant source of workload in the cluster. For our VMs, we use various flavors of Linux distributions, including Fedora, DSL Linux and RHEL in our experiments.

5.1 Virtualization Management Policy

In our virtualization management framework, the distributed resource management (DRM) policy is at the core of our global controller. The DRM policy supports two optimization schemes, *capacity* and *capacity+power*. Capacity only management is, in principle, similar to VMware DRS [45], in that its primary objective is only to optimize resource allocation. In this case, no power action is taken and the cluster is always on. Only the VMs are migrated in case of contention or in some cases, for rebalancing actions. The capacity+power scheme introduces dynamic consolidation and power actions. Depending on the aggregate demand in the cluster, the physical infrastructure also expands and squeezes in tune with the overall demand. This scheme is similar to VMware DPM [46]. For both schemes we have

two modes of operation, *recommendation* and *auto*, where recommendation mode only recommends actions, but does not execute them, while the auto mode autonomically executes the actions. The actuation mechanisms are VM migration for capacity management, and migration, suspend and resume actions for capacity+power management. One configuration parameter used in the DRM policy is the *utilization target*. This is defined as a percentage of a host's capacity, and describes a threshold that the policy strives to keep each host's utilization under. For example, if the utilization target is defined as 90%, the policy aims to keep each host below 90% utilization.

In our current implementation, the DRM policy is triggered at fixed time periods that we refer to as *DRM period*, which we vary from 30s to 5mins in various experiments. At each invocation, the DRM policy starts with a cluster-level analysis of overall capacity vs. cluster-wide demand. This analysis uses host capacities, utilization target and each host's overall resource use and demand gathered from the perf monitors residing on the hosts. Based on this, the algorithm decides whether the overall cluster is overcommited or undercommited. Then, the algorithm drills down to each host and performs a similar overcommit analysis for each host, using host capacity and demand measures, utilization target and basic demand forecasting [19]. In the case of capacity-only scheme, if no hosts are overcommitted, no action is needed, and the algorithm terminates. In the case of capacity+power scheme, after the cluster- and host-level overcommit analysis, the power-aware policy then explores any host power-on requirements (if overcommitted) or power-off opportunities (if significantly undercommitted). This is done by matching the cluster capacity to the overall projected demand by adding hosts to or removing hosts from the active system pool. This first-round assignment assumes demand is infinitely divisible across hosts, ignoring VM-level demand fragmentation. Therefore, after VM placement, some hosts might still remain overcommitted. In this case, another host is added to the cluster—if available—and a new placement is computed based on the larger system pool. The hosts selection for power up and power down actions are done via two heuristics. For power down, hosts with the least number of VMs are selected to minimize the number of migrations. For power up, the hosts are chosen based on their capacity, to provide the highest capacity delta with similar power up overheads.

After the host provisioning step, the same VM placement algorithm is used for both capacity and capacity+power schemes. The placement algorithm starts with a hypothetical cluster composed of all the active hosts in the system, plus any new hosts identified for power on. It uses the current system topology (VM-host mappings) to guide the placement decisions. In the case of capacity+power scheme, if any hosts are selected for power down, they are assigned an infinite initial demand, to avoid any VM placement on them. All other active hosts are assigned zero demand. After this, the VMs are ordered and placed onto hosts using a variation of *best fit decreasing* packing, where placement is biased to place each VM back to its current host if it does not overcommit the host. The purpose of this bias is to minimize VM migrations. The hypothetical placement continues in this fashion until all VMs are hypothetically placed, or until one of the hosts get overcommitted, which might trigger a next round of host power provisioning and VM placement cycle. After all VMs are placed, the placement algorithm arrives at

Undercommitment → Power action:

```
Cluster Stats:
Total Cap.(%)    Demand(%)
       1600.0       638.50
Host Stats:
Host     State  Cap.(%)  Demand(%)
Host0    ON         800     567.00
Host1    ON         800      71.50

OVERCOMMIT ANALYSIS:
   Cluster Undercommitted
   Host0 Undercommitted
   Host1 Undercommitted

CAPACITY MOVE RECOMMENDATIONS:
   All Hosts undercommitted
   NO CAPACITY MOVE REQUIRED

CAPACITY+POWER MOVE RECOMMENDATIONS:
Host POWER ACTIONS:
   Host0: Keep Powered  ON
   Host1: POWER OFF
VM MOVES:
   VM fc11NHMccs: Keep on Host0
    :
   VM fc11NHM04: Keep on Host0
   VM dslNHM01: Migrate Host1 -> Host0
   VM dslNHM02: Migrate Host1 -> Host0
   VM dslNHM03: Migrate Host1 -> Host0
   VM dslNHM04: Migrate Host1 -> Host0
```

Overcommitment → Capacity move:

```
Cluster Stats:
Total Cap.(%)    Demand(%)
       1600.0       993.00
Host Stats:
Host     State  Cap.(%)  Demand(%)
Host0    ON         800     700.00
Host1    ON         800     293.00

OVERCOMMIT ANALYSIS:
   Cluster Undercommitted
   Host0 Overcommitted
   Host1 Undercommitted

CAPACITY MOVE RECOMMENDATIONS:
   VM fc11NHM01: Keep on Host0
    :
   VM dslNHM04:  Keep on H0
   VM fc11NHM04: Migrate Host0 -> Host1

CAPACITY+POWER MOVE RECOMMENDATIONS:
Host POWER ACTIONS:
   Host0: Keep Powered  ON
   Host1: Keep Powered  ON
VM MOVES:
   VM fc11NHM01: Keep on Host Host0
    :
   VM dslNHM04:  Keep on Host Host0
   VM fc11NHM04: Migrate Host0 -> Host1
```

Table 1: Distributed resource management policy execution examples.

a target host-VM map. The difference of this with the original host-VM map defines the required migration actions. These migration decisions, together with the identified host power actions, constitute the DRM recommendations. The transition from recommendations to actions is done through the *Go/Nogo Logic*. This logic can determine whether the chosen actions are worthwhile to execute, given how aggressive/conservative the system is configured to be. In our current prototype this logic is implemented as a simple all or none decision, where either all the actions or none of the actions are carried out.

Table 1 shows two examples of the application of the DRM policy in our prototype experiments. The first example shows a case for significant cluster-level undercommitment, which enables a host power-down action. The second example shows an overcommit scenario and the triggered capacity move action.

6. PROTOTYPE RESULTS

Our experimental results include both base real-system evaluations and detailed simulations using a data center simulator. We use our developed experimental platform to quantify the power-performance trade-offs of our virtualization power management approach, leveraging low-latency server power states. Here, we describe some of our real-system prototype results, followed by the scale-out simulations of the next section. Our power-aware virtualization manager implements `S3`, `S4`, and `S5` power modes. However, our evaluation mainly focuses on `S3` as the low-latency power state, and `S5` as the baseline power management approach.

Our real-system evaluation employs a distributed workload with a cyclic demand pattern. The workload is distributed unevenly to running VMs, with varying demand levels and duty cycles. We evaluate a one cycle period of this workload, starting with the high demand region, followed by successive ramp-down and ramp-up phases, as different VMs go through their demand cycles. Over one period, the average demand is around 60% of its peak value and the ratio of the highest to lowest demand is approximately 3. We use a distributed workload generator to distribute workload across VMs, which also provides a runtime, application-level performance feedback for each VM [19]. We use this to quantify the performance impact of power management. Figure 7 shows the overall flow of this evaluation, including the workload behavior, VM topology and demand, end-to-

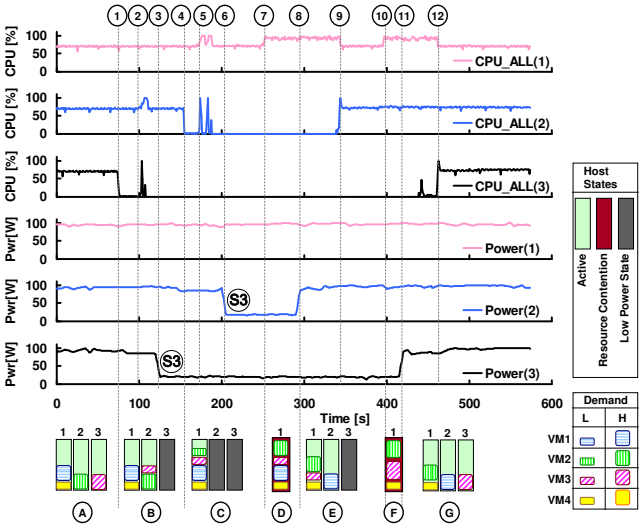


Figure 7: Power-aware virtualization management timeline from the prototype implementation.

end physical resource use, power consumption, virtualization management and power actions.

The bottom of the plot shows the evolution of cluster topology (labeled Ⓐ to Ⓖ) and the distribution of the VMs into three hosts. The size of the VMs show whether they have high or low demand. For each VM, high demand is around 60-80% and low demand is around 5%. We use `nmon` for our second-granularity resource use monitoring, including all resource components, CPU, memory, network and disk I/O. We use the service processor APIs for second-granularity power measurements. The plots show the CPU and power profile of the three hosts used in the experiment. The timeline is labeled ① to ⑫, pinpointing the key events in the power-aware virtualization management experiments. We use a 90% utilization target in our experiments. The highest cluster demand in our experiment reaches to around 240%, which requires distributing the VMs across all hosts, and the lowest is around 80% enabling all VMs to fit in the capacity of a single host.

At the beginning of the experiment, most VMs are at high demand (topology Ⓐ) and distributed across hosts. The cluster is at steady state with no capacity moves or power-up actions required and no power-down opportunities available. Then, at time ① VM demand on host3 drops. This is realized by the DRM policy the next time it is invoked, at ②. The elapsed time between ① and ② is the lost power saving opportunity due to DRM response. Once the policy is invoked, the change in demand creates a power saving opportunity. The VM is migrated from host3 to host2, the short-lived peaks in host utilizations here are indicative of the ongoing migration processes. Then, at ③ host3 is evacuated and is suspended. The elapsed time between ③ and ② is the additional lost power savings opportunity due to migration and suspend overheads. At this point, the cluster topology is as shown in Ⓑ, with host3 powered down. Similarly, steps ④-⑥ show a similar pattern for further decline in cluster demand, leading to multiple migrations from host2 to host1—as seen with multiple short-lived peaks—and consolidation of all VMs in host1, with host2 powered down (Ⓒ). This is followed by a ramp up phase after only a one minute low demand period. At time ⑦ the demand in some of the VMs rises in host1, creating contention in the host (Ⓓ). The DRM policy is invoked at ⑧, realizing the contention, identifies the optimal reconfiguration based on VM demands, and as a result starts resuming host2. After a short time period, host2 resumes and is readied for incoming migrations, which subsequently start to push some of the VM load into host2, from host1. The migrations complete at ⑨, resolving the contention in host1 and the cluster reaches a well-performing quiescent state again, as in Ⓔ. During the time between ⑦ and ⑨, the system remains under contention, and incurs performance degradation. The elapsed time between ⑧ and ⑦ is the source of the performance degradation due to DRM response; and the time between ⑨ and ⑧ identifies the performance degradation due to resume and migration latencies. Similar to the first ramp-up phase, steps ⑩-⑫ highlight a similar round of events for a second ramp-up phase, which completes the cycle, bringing the cluster back to its initial state.

This fairly straightforward experiment captures some of the key aspects of virtualization power management, and the dramatic difference low-latency power states make in management efficiency and agility. Overall, end-to-end, the whole experiment spans around 10 minutes, showing a relatively fast-moving workload—in virtualization management scales—with multiple ramp-up and ramp-down phases. Existing virtualization power management schemes cannot react to workloads at such granularity, without risking significant repercussions in performance. However, here we show that very simple virtualization management, with no sophisticated risk assessment, cost-benefit analysis can effectively respond to these fast workload dynamics by leveraging S3-based low-latency server power states. We also strive to minimize the other two dimensions of the overheads, DRM response and migration latency. We run the DRM policy at a relatively fine 30s interval, and employ additional migration latency and resource overhead optimizations [20]. As a result of all these, the entire cycle including the virtualization management and power actions is captured in this 10 minute window.

The experimental results clearly depict the three dimensions of overhead, and their substantially different implications for decreasing and increasing workload patterns. We quantitatively demonstrate the effects of these on the overall power-performance characteristics in Figures 8 and 9.

Figure 8 shows the aggregate power savings through the course of the experiment. The power savings are shown as a running time average from the beginning of the experiment and are aggregated across all hosts. At the end of the complete cycle, our prototype virtualization power manager achieves close to 20% power savings compared to a the baseline case with S5-based power management. We also include two additional hypothetical savings figures. The "Zero Overhead" case corresponds to the achievable power savings, when we delineate the opportunity lost and power overheads due to suspend, DRM response and migration overheads. This shows an additional 6% power savings opportunity. The third curve "Energy Proportional" shows the additional power savings achieved if the servers were closer to energy proportionality, with an additional 7% savings. The prototype curve shows the additional savings we have extracted from the cluster by leveraging the low-latency power states. The zero overhead case highlights the additional recoverable benefits as we improve the three overhead dimensions further, and the energy proportional case highlights further improvements possible by improving power proportionality

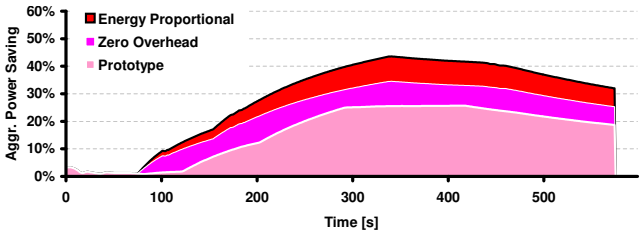


Figure 8: Power savings.

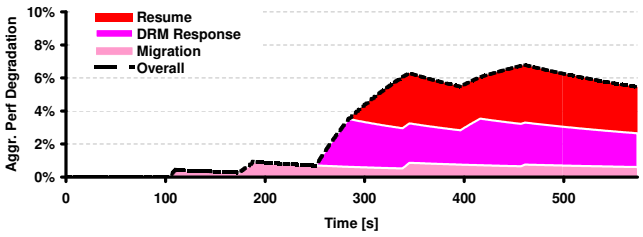


Figure 9: Performance degradation.

of the hardware itself.

Figure 9 shows the aggregate, time-averaged performance degradation through the workload cycle, in a similar fashion as Figure 8. Here, we show the overall end-to-end performance degradation is close to 5%. This degradation mainly stems from the DRM response, resume and migration delays on the ramp-up path, and is decomposed as such in the plot. As we have argued earlier in our characterization discussion, with S3 we bring down the power-state overhead on par with DRM response and migration overheads. As the decomposition shows, the overheads of the three dimensions have comparable impact.

In Figure 10 we show a higher-level comparison of our prototyped virtualization power management solution to two versions of existing, S5-based techniques. The "baseline DPM" imitates the current behavior of existing power management schemes, which waits for a breakeven period after a decrease in demand, before triggering power actions. The "Aggressive DPM" is based on the same base DPM technique, but with aggressiveness tuned to its maximum. With this change, it responds to dynamic demand changes very quickly, similar to our prototype, but cannot catch up to the efficiency of our solution. The heavy-handed power actions weigh down the entire virtualization management solution and it pays a serious performance price for being aggressive. We see the baseline case achieves no power savings, and hence, pays no performance penalty, as it cannot find enough room to trigger power actions. Aggressive DPM achieves a small amount of power savings; however, at a very high performance cost. Our S3 prototype shows the much more desirable power-performance trade-off point it operates at, with close to 20% power savings and less than 5% performance degradation.

This set of results highlights the key comparative value of our agile, efficient virtualization power management with low-latency power states. We show that we can achieve *agile* power management by being able to respond to quickly-varying workloads that today's management schemes cannot respond with their cost-benefit trade offs. Our prototype can achieve an impressive 20% power savings from minute-granularity dynamic workload patterns, which is not accessible via existing techniques. The results also show that existing management schemes do the correct thing by being cautious in their actions, as the price for aggressiveness can be quite steep with heavy-handed actions. Our results

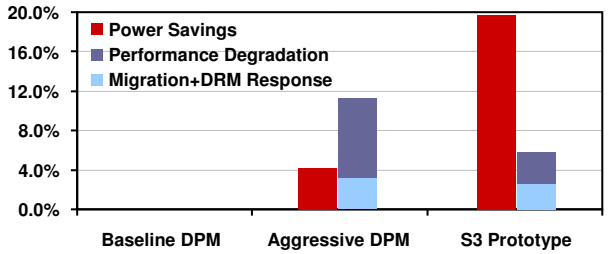


Figure 10: Power-performance trade offs in comparison to S5-based virtualization power management.

also show that our solution is very *efficient* in extracting the potential power saving opportunities, when compared to achievable savings with no virtualization management and power-state change overheads. The 5% performance impact fares significantly better than the possibilities with existing approaches. However, in case stricter performance constraints are of mandate, the power-performance trade-offs are tunable via varying the desired utilization target, a principle well established in existing virtualization management schemes.

Another interesting characteristic of our virtualization power management solution is shown in the performance graphs of Figure 10, which are further divided into the overhead components. These show that, in our prototype the cost of power actions are dialed down such that they are on par with migration and DRM response costs, the two costs that is also common to power-agnostic virtualization resource management, such as VMware DRS [45]. This observation has significant implications for the widespread adoption of dynamic power management in virtualized systems. As power management with low-latency power states has the same associated risk factors as base distributed resource management, in principle the same acceptance criteria applies for end users, where comparative cost of enabling power management is no different from simply enabling base resource management.

7. SCALE-OUT RESULTS

In order to understand the likely behavior of S3-enabled dynamic virtualization power management in larger-scale systems running realistic dynamic workloads, we augmented *DCSim*, a discrete-event simulation framework for enterprise systems that includes models for servers, data center topology, workloads and virtualization management middleware [16]. Using DCSim, we can configure the data center with different scales, server and VM configurations, and with different virtualization management parameters for resource management, migration and network characteristics.

We assign a fixed resource capacity to each host and use DCSim's scheduling logic to dynamically distribute resources among running VMs. We mirrored our prototype's DRM policy in DCSim and added the server power state latency and power characteristics from our characterization experiments. We extensively validated DCSim against our real-system results in both management actions and power/performance characteristics. Some key aspects of this validation are shown in Figure 11, which repeats our previous real-system experiment in DCSim. The management actions created by DCSim perfectly match our prototype results. As the figure shows, the power and performance profiles closely match reality not just for the final values, but throughout the sample-by-sample execution timeline. Throughout the

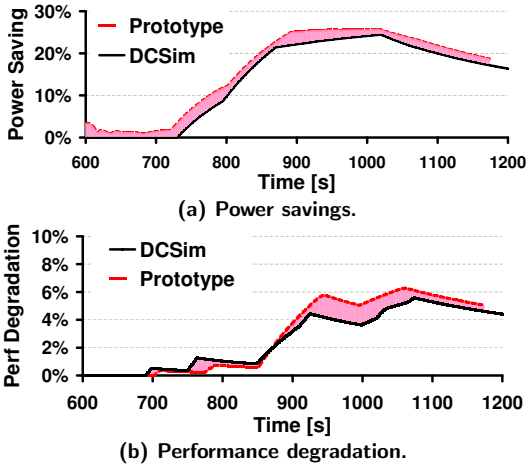


Figure 11: DCSim validation.

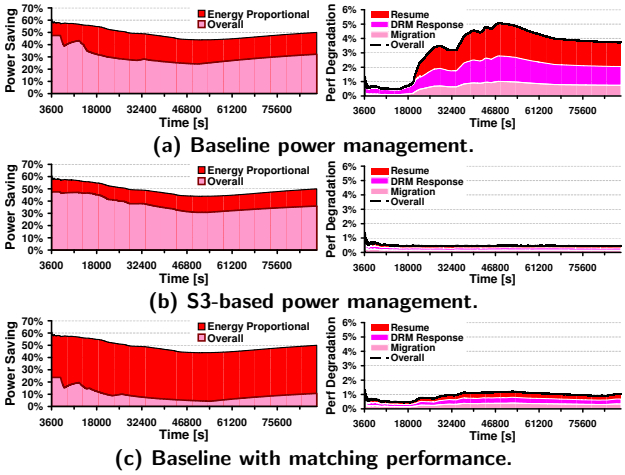


Figure 12: Australian Open workload results for power savings (left) and performance degradation (right).

timeline, DCSim power results are within 3% of real-system results, while performance is within 2%. The average error for power and performance is less than 1.5% and 0.5% respectively.

We use two additional real workloads for our scale-out evaluations. The first workload is based on a trace of web traffic directed to a set of servers that supported the 2009 Australian Open web site. This trace exhibits typical cyclic workload variations with daily periodicity. We also use SPECvirt virtualization benchmark, to evaluate the impact of scale-out with different number of tiles. A tile consists of six VMs configured as web, mail, application, database, infrastructure servers and an idle server. The scaling evaluation is achieved by adding more tiles to a virtualized system while meeting the same performance criteria.

Figure 12 shows our first set of evaluations using the Australian Open workload. Each of the simulated servers mimics our prototype servers in all aspects, while the VM CPU and memory usages are drawn from a random distribution based on the Australian Open workload. In Figure 12(a), we show the power-performance trade-offs with baseline distributed power management, similar to the prior aggressive DPM configuration. The aggressive DPM policy here achieves significant power savings across the cluster, averaging around 30%. However, the cost of aggressiveness is seen in the significant performance degradation. In contrast, Figure 12(b) shows that with our S3-based virtualization power management, we can achieve dramatic improvements in power-performance trade-offs. Our approach achieves 38% power savings, while the system experiences a minimal performance degradation impact of less than 1%. In Figure 12(c), we tune down the aggressiveness of baseline DPM by reducing utilization threshold and adding non-zero breakeven time for power actions, in order to reduce the performance impact to levels comparable to our approach. As the performance is pushed down to 1% the power savings of the baseline approach diminishes dramatically, to below 10%. S3-based power management is clearly superior to both aggressive and less-aggressive DPM-like approaches.

Figure 13 summarizes these observations across a broader set of configurations. Here, we configure our servers with varying assumptions about the three latency components combined together, and evaluate the power savings at a fixed performance target. For each assumed latency, we evaluate the system with different levels of aggressiveness and pick the configuration with the highest power savings that satisfies the performance target. As our baseline, we explore the best static consolidation configuration that meets the same performance target and normalize dynamic management power to the observed static consolidation power in each case. This figure outlines the key advantage, and necessity for our agile virtualization power management solution. Existing virtualization power management techniques fall on the right side of this plot, showing that, for a real workload, the benefits of power management with high-latency power states have limited returns over static consolidation, as the power management policies need to be dramatically throttled down to avoid their high performance impact. On the other hand, our S3-based approach pushes us closer to the left end of the graph, showing that very dramatic benefits can be realized while meeting the same performance goals. Thus, for any meaningful returns with power-aware virtualization, low-latency power states, together with any welcome improvements on the DRM response and live VM migration, play a crucial role.

The power results of Figures 12 and 13 also include the energy-proportional power profile of the evaluated systems as a reference point. This shows the potentially achievable power savings in an ideal system. Our virtualization power management solution exploits a sizable portion of this potential, but not all. The gap results, to a large extent, from granularity effects. The power savings are achieved at the granularity of hosts (as we turn on and off hosts) and workload migration is achieved at the granularity of VMs. This fragmentation effect can be mitigated across the entire system with increasing scale. As we scale-out virtualized systems, we expect them to further approach energy proportionality at the cloud scale.

Figure 14 explores this scale-out effect for the Australian Open and SPECvirt workloads. For each workload, we display the average performance degradation and the amount of power saved at each scale. The results for both workloads show an interesting trend. Applying the same DRM policy, the power savings approach energy proportionality with increasing scale. However, the performance results grow somewhat worse with scale, most likely because we introduce significantly more movement (migrations and power actions) into the virtualized system. One should keep in mind that our measure of performance degradation rep-

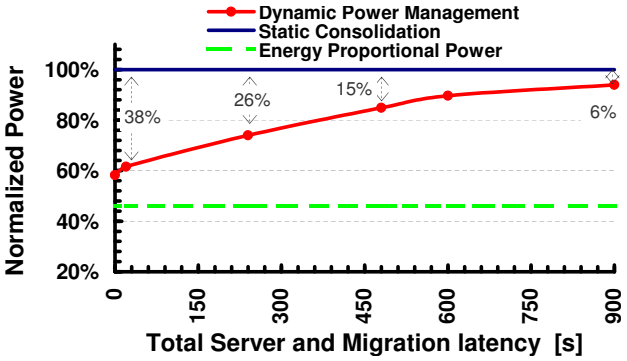


Figure 13: Power savings with different response latencies.

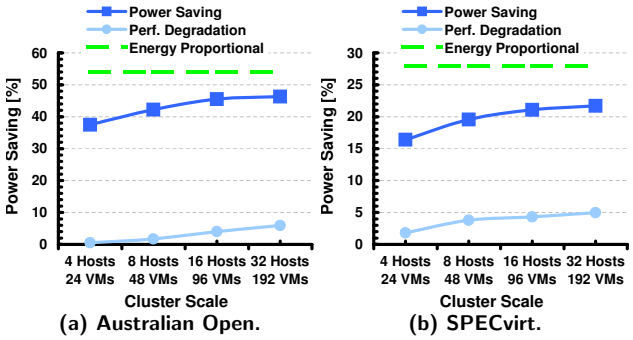


Figure 14: Scale-out results for Australian Open and SPECvirt workloads.

resents a low-level system view as opposed to a higher-level application view (e.g. based on a service level agreement) and may therefore provide a more pessimistic view of performance. Nonetheless, the growing performance degradation with system scale warrants further investigation.

8. RELATED WORK

A wide range of prior work explored data center power management with server consolidation [9, 35, 36, 38, 42, 46]. Most of these techniques use virtualization or job scheduling abstractions to move workloads across physical systems. These studies show the significant benefits of distributed power management in data centers. However, they commonly resort to detailed cost-benefit analysis, trend forecasting and other techniques to mitigate the performance impact of their power actions, particularly in response to a demand surge. In comparison to these, our work explores the use of low-latency power states in enterprise servers and demonstrates their application to agile, efficient power management in virtualized systems.

Other work has explored alternative techniques for improving energy efficiency at the systems and data center levels. PowerNap describes system-wide approach, with distributed sleep modes to dramatically improve power response characteristics of systems to fine-grain time variations in workload demand [28]. Parasol [4] and GreenSlot [15] leverage renewable energy and low-power server modes to improve data center energy efficiency in combination with workload scheduling.

The use of low-latency power states in data center servers has also attracted attention from recent research. Ghandi et al. explore the effectiveness of sleep states in data center servers via simulations modeling a range of effective power states with different latency and energy trade offs [12, 13, 14]. These studies show the potential of low-latency states for distributed power management and demonstrate the effect of varying conservativeness in management policies for different workload characteristics. There has also been prior work in desktop virtualization space for leveraging low-latency power states for power management. LiteGreen [10] and Jettison [5] both explore idle desktop migration using virtualization and pushing these systems into lower power modes once their state is migrated. LiteGreen uses full desktop migration for power management, while Jettison is a fine grained partial migration approach, where only the desktop working set is transferred to a server. Additional data is fetched from the system on demand. Our work, in comparison, demonstrates the use of low-latency states in servers for distributed power management in virtualized data centers.

9. CONCLUSION

In this paper we explore the use of low-latency power states in enterprise servers and demonstrate their dramatic benefits when applied in power-aware virtualization management. Compared to today's virtualization management solutions, these techniques fundamentally shift the power-performance trade-offs and enable agile and efficient power management in virtualized data centers.

We build multiple experimental prototypes to enable S-states in server platforms and describe our experiences with the challenges and requirements of supporting these states across the stack, including hardware, firmware and the operating system. Despite these challenges, our quantitative evaluations show the significant potential returns in making these power modes also ubiquitously available in servers. We present a characterization of the power-performance trade offs with different server power states, which underlines the dramatically better operating point with S3-based power management, compared to other commonly-used power states. We build a power-aware virtualization management framework, leveraging these low-latency power states, and show its superior power and performance efficiency compared to current approaches. Our real-system evaluations show that this agile, efficient virtualization power management approach can improve data center power efficiency by more than 30% by being able to respond to quickly-varying workload patterns with minimal performance impact, and with comparable power efficiency to deeper, more heavy-handed power states. Our scale-out evaluations further emphasize the benefits of this technique, approaching closer to energy proportionality with increasing system scale.

Our work demonstrates a promising path to substantial improvements in data center energy efficiency for virtualized systems. There remain, however, many interesting challenges in achieving energy efficiency at scale, which require more than a single technique isolation, but rather a concerted effort across the stack, from hardware and architectures to systems and management middleware.

Acknowledgments

We would like to thank Dipankar Sarma, Makoto Ono, Zeydy Ortiz, Freeman Rawson, Matthew Eckl, Max Asbock, Vaidyanathan Srinivasan, Takaaki Shiozawa, Irene Zubarev, Gaines McMillan, Thomas Pahel, Paul Bashor, Mike Scollard, Jim Hamilton, Karthick Rajamani and Jack Kouloheris for their help and for many useful discussions during various stages of this project. We also thank our anonymous reviewers for their very insightful comments that helped shape the structure and contributions of this work.

10. REFERENCES

[1] M. Annavaram, E. Grochowski, and J. Shen. Mitigating Amdahl's Law Through EPI Throttling. In *Proceedings of the 32nd International Symposium on Computer Architecture (ISCA-32)*, 2005.

[2] Arch Linux. Pm-utils. https://wiki.archlinux.org/index.php/Pm-utils, 2012.

[3] L. A. Barroso. The Price of Performance. *ACM Queue*, 3(7):48–53, Sept. 2005.

[4] R. Bianchini, I. Goiri, K. Le, and T. Nguyen. Parasol: A solar-powered datacenter. In *ACM European Conference on Computer Systems (Eurosys)*, 2012.

[5] N. Bila, E. de Lara, K. Joshi, H. A. Lagar-Cavilla, M. Hiltunen, and M. Satyanarayanan. Jettison: Efficient Idle Desktop Consolidation with Partial VM Migration. In *European Conference on Computer Systems (Eurosys)*, 2012.

[6] T. Brey, B. Bigelow, J. Bolan, H. Cheselka, Z. Dayar, J. Franke, D. Johnson, R. Kantesaria, E. Klodnicki, S. Kochar, et al. BladeCenter Chassis Management. *IBM Journal of Research and Development*, 49(6):941–961, 2005.

[7] L. Brown, A. Keshavamurthy, D. S. Li, R. Moore, V. Pallipadi, and L. Yu. ACPI in Linux. In *Linux Symposium (OLS)*, 2005.

[8] M. Cardosa, M. Korupolu, and A. Singh. Shares and Utilities based Power Consolidation in Virtualized Server Environments. In *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2009.

[9] J. S. Chase, D. C. Anderson, P. N. Thakar, A. N. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *Proc. 18th Symposium on Operating Systems Principles (SOSP)*, 2001.

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

[11] A. Gandhi, M. Harchol-Balter, R. Das, J. Kephart, and C. Lefurgy. Power Capping via Forced Idleness. In *Workshop on Energy-Efficient Design (WEED)*, 2009.

[12] A. Gandhi, M. Harchol-Balter, and M. Kozuch. The Case for Sleep States in Servers. In *SOSP 4th Workshop on Power-Aware Computing and Systems (HotPower 2011)*, 2011.

[13] A. Gandhi, M. Harchol-Balter, and M. Kozuch. Are sleep states effective in data centers? In *2012 International Green Computing Conference (IGCC)*, 2012.

[14] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. Kozuch. AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers. *Transactions on Computer Systems*, 30(4), 2012.

[15] I. Goiri, K. Le, M. Haque, R. Beauchea, T. Nguyen, J. Guitart, J. Torres, and R. Bianchini. GreenSlot: Scheduling Energy Consumption in Green Datacenters. In *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2011)*, 2011.

[16] J. Hanson, I. Whalley, M. Steinder, and J. Kephart. Multi-aspect Hardware Management in Enterprise Server Consolidation. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2010.

[17] Hewlett-Packard, Intel, Microsoft, Phoenix, and Toshiba. Advanced configuration and power interface specification. http://www.acpi.info, September 2004.

[18] C. Isci, G. Contreras, and M. Martonosi. Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management. In *Proceedings of the 39th ACM/IEEE International Symposium on Microarchitecture (MICRO-39)*, 2006.

[19] C. Isci, J. Hanson, I. Whalley, M. Steinder, and J. Kephart. Runtime Demand Estimation for Effective Dynamic Resource Management. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2010.

[20] C. Isci, J. Liu, B. Abali, J. Kephart, and J. Kouloheris. Improving Server Utilization Using Fast Virtual Machine Migration. *IBM Journal of Research and Development*, 55(6), 2011.

[21] Jeffrey Nowicki and Andy Arhelger. Optimizing Virtual Infrastructure with PowerVM and the IBM Systems Director VMControl. Whitepaper, IBM Systems and Technology Group, 2010.

[22] H. Jiang, M. Marek-Sadowska, and S. R. Nassif. Benefits and Costs of Power-Gating Technique. In *IEEE International Conference on Computer Design (ICCD)*, 2005.

[23] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. Kvm: the linux virtual machine monitor. In *Ottawa Linux Symposium (OLS)*, 2007.

[24] J. G. Koomey. Estimating total power consumption by servers in the U.S. and the world, 2007.

[25] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang. Power and Performance Management of Virtualized Computing Environments Via Lookahead Control. In *Proceedings of the International Conference on Autonomic Computing*, 2008.

[26] E. Le Sueur and G. Heiser. Slow Down or Sleep, That is the Question. In *USENIX Annual Technical Conference*, 2011.

[27] J. Lee and N. Kim. Optimizing Throughput of Power and thermal Constrained Multicore Processors Using DVFS and Per-core Power Gating. In *Design Automation Conference (DAC)*, 2009.

[28] D. Meisner, B. T. Gold, and T. F. Wenisch. PowerNap: Eliminating Server Idle Power. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2009.

[29] D. Meisner and T. Wenisch. DreamWeaver: Architectural Support for Deep Sleep. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012.

[30] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis. Efficient resource provisioning in compute clouds via vm multiplexing. In *IEEE International Conference on Autonomic Computing (ICAC)*, 2010.

[31] J. Moore, J. Chase, and P. Ranganathan. Making scheduling "cool": Temperature-aware workload placement in data centers. In *Proc. 2005 USENIX Annual Technical Conference (USENIX '05)*, 2005.

[32] R. Muralidhar, H. Seshadri, V. Bhimarao, V. Rudramuni, I. Mansoor, S. Thomas, B. Veera, Y. Singh, and S. Ramachandra. Experiences with Power Management Enabling on the Intel Medfield Phone. In *Linux Symposium (OLS)*, 2012.

[33] A. Naveh, E. Rotem, A. Mendelson, S. Gochman, R. Chabukswar, K. Krishnan, and A. Kumar. Power and thermal management in the intel core duo processor. *Intel Technology Journal*, 10(2):109–122, 2006.

[34] D. Snowdon, E. Le Sueur, S. Petters, and G. Heiser. Koala: A platform for OS-level Power Management. In *4th ACM European Conference on Computer Systems (Eurosys)*, 2009.

[35] S. Srikantaiah, A. Kansal, and F. Zhao. Energy Aware Consolidation for Cloud Computing. In *USENIX Conference on Power Aware Computing and Systems (PACS)*, 2008.

[36] M. Steinder, I. Whalley, J. Hanson, and J. Kephart. Coordinated management of power usage and runtime performance. In *Proceedings of the Network Operations and Management Symposium (NOMS)*, 2008.

[37] K. Tian, K. Yu, J. Nakajima, and W. Wang. How Virtualization Makes Power Management Different. In *Linux Symposium (OLS)*, 2007.

[38] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and X. Zhu. Delivering Energy Proportionality with Non Energy-proportional Systems–Optimizing the Ensemble. In *HotPower*, 2008.

[39] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. In *the 5th symposium on Operating systems design and implementation (OSDI)*, 2002.

[40] R. Urgaonkar, U. Kozat, K. Igarashi, and M. J. Neely. Dynamic Resource Allocation and Power Management in Virtualized Data Centers. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2010.

[41] U.S. Environmental Protection Agency ENERGY STAR Program. Report to congress on server and data center energy efficiency, 2007.

[42] A. Verma, P. Ahuja, and A. Neogi. pMapper: Power and Migration Cost Aware Placement of Applications in Virtualized Systems. In *Proceedings of the ACM Middleware Conference*, 2008.

[43] VMware Inc. VMware Capacity Planner, http://www.vmware.com/products/capacity-planner/.

[44] VMware Inc. VMWare vCenter CapacityIQ, http://www.vmware.com/products/vcenter-capacityiq/.

[45] VMware Inc. Resource Management with VMware DRS. Whitepaper, VMware Inc., 2006.

[46] VMware Inc. VMware Distributed Power Management: Concepts and Use. Whitepaper, VMware Inc., 2010.

[47] M. S. Ware, K. Rajamani, M. S. Floyd, B. Brock, J. C. Rubio, F. L. R. III, and J. B. Carter. Architecting for Power Management: The IBM POWER Approach. In *Proceedings of the 12th International Symposium on High-Performance Computer Architecture (HPCA-16)*, 2010.