vManage: Loosely Coupled Platform and Virtualization Management in Data Centers

Sanjay Kumar, Intel Corp., Hillsboro, OR, sanjay.k.kumar@intel.com
Vanish Talwar, HP Labs, Palo Alto, CA, vanish.talwar@hp.com
Vibhore Kumar, IBM Research, Hawthorne, NY, vibhorek@us.ibm.com
Parthasarathy Ranganathan, HP Labs, Palo Alto, CA, partha.ranganathan@hp.com
Karsten Schwan, Georgia Tech, Atlanta, GA, schwan@cc.gatech.edu

ABSTRACT

Management is an important challenge for future enterprises. Previous work has addressed platform management (e.g., power and thermal management) separately from virtualization management (e.g., virtual machine (VM) provisioning, application performance). Coordinating the actions taken by these different management layers is important and beneficial, for reasons of performance, stability, and efficiency. Such coordination, in addition to working well with existing multi-vendor solutions, also needs to be extensible to support future new management solutions potentially operating on different sensors and actuators. In response to these requirements, this paper proposes vManage, a solution to loosely couple platform and virtualization management and facilitate coordination between them in data centers. Our solution is comprised of registry and proxy mechanisms that provide unified monitoring and actuation across platform and virtualization domains, and coordinators that provide policy execution for better VM placement and runtime management, including a formal approach to ensure system stability from inefficient management actions. The solution is instantiated in a Xen environment through a platform-aware virtualization manager at a cluster management node, and a virtualization-aware platform manager on each server. Experimental evaluations using enterprise benchmarks show that compared to traditional solutions, vManage can achieve additional power savings (10% lower power) with significantly improved service-level guarantees (71% less violations) and stability (54% fewer VM migrations), at low overhead.

Categories and Subject Descriptors

D.4.0 [Operating Systems], C.0 [Computer System Organization]

General Terms

Management, Algorithms, Performance, Standardization

1. INTRODUCTION

The effective use of IT infrastructure strongly depends on easily and efficiently managing server hardware, system resources, and applications. However, rising complexity and scale in today's enterprise data centers has led to increased costs for such management (in some cases, up to 60-70% of the total IT budget [1]). The adoption of virtualization and the consequent need to manage virtual machine (VM) migration and mappings between physical and virtual resources [2,3] can further exacerbate these challenges.

There has been significant industry investment and prior work to improve the ways in which systems are managed. These solutions

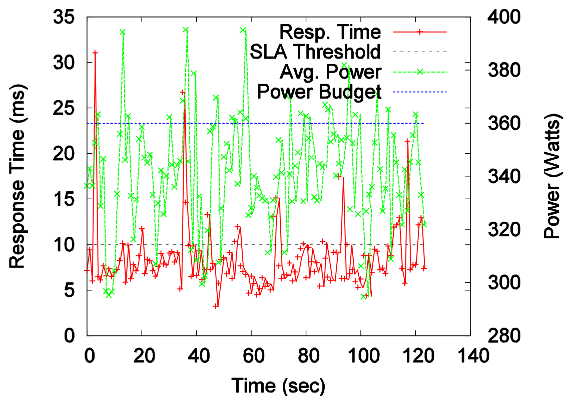


Figure 1: Illustration of problems with non-coordinated platform and virtualization management

can be classified as either platform management or virtualization management. The former is concerned with managing the hardware; examples include server configuration, hardware monitoring, and power and thermal management [4,5,6]. The latter is concerned with managing VM resources and application performance. Some examples include VM provisioning, runtime monitoring, SLA management, and data backup [7,8,9,10,11].

Traditionally, platform management and virtualization management solutions have been designed in isolation (in "silos"). The increased adoption of virtualization and the resulting blurring between the virtual and physical domains, however, argue for coordinating these solutions. This can avoid ineffective management with respect to stability and efficiency caused by the autonomous actions taken in current isolated designs. Figure 1 presents an illustrative excerpt from a 13-node experimental setup (discussed later in Section 4) when a power manager and a virtualization manager are deployed in isolation. The figure illustrates (1) the power consumption and the number of times the power budget (corresponding to the cooling provisioned in the system) is violated, as well as (2) the response time and the number of times the Service Level Agreement (SLA) is violated. As the figure illustrates, often the SLA and power violations happen approximately around the same time. This is because one violation triggers the other. Specifically, the power violations cause the power manager to reduce the CPU frequency; unfortunately, this leads to SLA violations. These SLA violations, then, trigger the power manager to increase the CPU frequency, which cause further power violations, and the cycle continues. Hence, this baseline "silo-ed" system not only violates both the power budget and response time guarantees significantly, but it also suffers from oscillatory instability between power and SLA violations due to the lack of coordination.

Similarly, in another scenario and again in the absence of coordination, a virtualization manager that migrates VMs in response to insufficient CPU cycles, and a power manager that increases CPU frequency for the same reason, may result in redundant actions causing wastage of system resources and hence system inefficiency. Extending this problem more generally within a data center, one can imagine uncoordinated VM migrations resulting in undesirable effects such as VM 'ping-ponging' (continuous migration of VMs among a set of hosts). All of these inefficiencies are due to isolated platform and virtualization management, ultimately resulting in higher data center costs.

Coupling the virtualization and platform management solutions avoids redundant and overlapping management functions, and results in greater efficiency and stability in the data center. However, effective coordination comes with several challenges. Manual or ad-hoc approaches can be time-consuming and potentially very costly in human costs. Similarly, while it may seem plausible at first glance to develop a well integrated platform-virtualization solution to achieve such coupling, a solution that is too tightly-coupled may not actually be viable. This is both for practical reasons, such as the need for concurrent development of platform and virtualization management solutions in multi-vendor scenarios and to preserve investment in existing products, and for theoretical reasons, such as the difficulty of finding effective general methods for controlling multi-layer systems and applications [12]. Additionally, any solution that is developed needs to be extensible to support different management solutions, different sensors, different actuators, while allowing coordination policies to be developed independent of low-level implementation issues.

In this paper, we present vManage, a practical coordination solution that loosely-couples platform and virtualization management in future data centers. The solution is comprised of *registry and proxy* mechanisms that provide unified monitoring and actuation across platform and virtualization domains, and of *coordinators* that provide policy execution for platform-aware virtualization management and virtualization-aware platform management. In addition, we provide for `plug-in' components, an example being one for stability management for coordinated VM placement in the data center, termed *stabilizer*. The vManage design has several useful features such as the ability for the coordinators to interface with existing management controllers, be easily portable across hardware platforms in an implementation independent manner, and flexibility and extensibility in allowing new management solutions to participate in a "plug-and-play" fashion.

We have implemented vManage in a Xen-based infrastructure with coordination policies for improved VM placement and better runtime management. We have evaluated the solution in a cluster with 28 VMs running a mix of business, web, internet services, and batch workloads. Our results show that the vManage coordinated approach achieves improved power savings (10% lower power) and significantly better QoS (71% less violations) with better stability (54% fewer VM migrations), all at low overheads and minimal interference with host workloads.

The remainder of the paper is organized as follows. Section 2 discusses in detail the vManage solution for coordinated platform and virtualization management. Section 3 describes the implementation for vManage. Evaluation results are presented in Section 4. Finally, Sections 5 and 6 discuss related work and conclude the paper respectively.

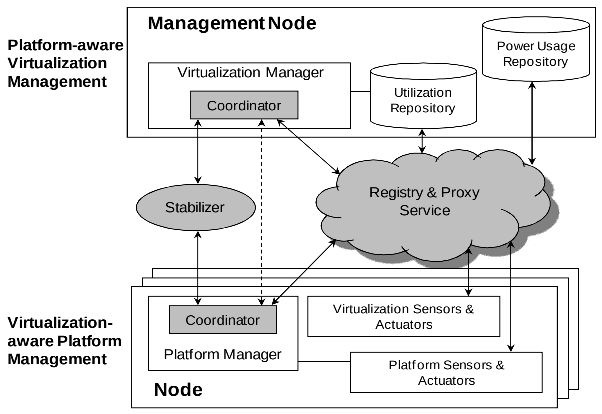


Figure 2: vManage: Loosely-coupled platform and virtualization management.

2. vMANAGE SOLUTION

2.1 System Model

Figure 2 shows the vManage solution for coordinating platform and virtualization management in the data center. Each node has multiple virtualization and platform sensors/actuators. Examples include power and thermal sensors, application SLA sensors, utilization monitoring daemons, ACPI p-state actuators (that vary performance for lower power), and VM migration actuators. Monitoring data from these sensors are aggregated across multiple nodes and stored in cluster-level repositories (e.g, see the system utilization and power usage repositories in Fig. 2). In addition, each node hosts a platform manager for hardware-related attributes such as power, and a cluster management node hosts a virtualization manager to manage VMs through their lifecycle. In such a system, we introduce coordination at two levels (as indicated in Fig. 2) -- platform-aware virtualization management at the cluster management node, and virtualization-aware platform management on each node.

In order to keep the system loosely-coupled and to be portable across multiple hardware platforms, we make two design choices. First, we introduce a *registry & proxy service* that is responsible for discovering and registering the individual sensors and actuators at a unified location using publish-subscribe mechanisms. Second, we introduce a *coordinator* that can be plugged into existing virtualization and platform managers. This coordinator interfaces with the registry & proxy service, and through it, becomes aware of and obtains data from available platform and virtualization management sensors. The coordinator then uses the unified data from these diverse sensors to implement control policies for better coupling. The policies are executed using actuators from both platform and virtualization management domains, similarly discovered and accessed via the registry and proxy service. In addition to the registry & proxy service and the coordinators, a key aspect of our solution includes the *stabilizer*, to which the coordinators interface. The stabilizer ensures system stability and prevents coordinators from taking redundant and unnecessary actions. We describe these elements below in the context of the operation of the coupled virtualization and platform managers as has been used in our prototype.

2.2 Platform-aware Virtualization Management

The coordinator plug-in to the virtualization manager implements policies for providing integrated VM placement considering both *VM* requirements such as CPU, memory, network bandwidth, VM priority, etc., as well as *platform* requirements such as power budget (or other platform attributes like reliability or physical location). It is invoked during initial provisioning of VMs and also subsequently when a suitable host is to be chosen during VM migration.

When invoked, the coordinator first obtains a list of nodes (provided by the virtualization manager) that satisfies the new (or migrating) VM's various requirements (CPU, memory, etc.). Thereafter, it queries the registry and proxy service to obtain information on the available power usage repositories in the cluster. When found, the power usage repository is queried to obtain the current power consumption for all nodes satisfying the VM's requirements. Power budget filtering is then applied on these nodes checking that the node will not violate its power budgets when the VM is deployed on it. From these filtered nodes, a final node is selected for VM placement using a pre-determined policy. For example, the coordinator can select the first "matching" node that satisfies VM and platform requirements, or perform a best match (e.g., the most idle host for a VM). Alternatively, and preferably, the coordinator chooses a node that ensures system stability, i.e., would satisfy both the VM and platform requirements for a sufficient period of time into the future with the stability checks provided through the *stabilizer* described in Section 2.4. Once the final node is selected by the coordinator, the virtualization manger deploys the VM on that chosen node based on the selection.

This approach for selecting a node considering both the VM and platform attributes avoids bad decisions that lead to early power budget violations or system downtimes. It also reduces the use of expensive runtime techniques like workload balancing to handle such violations or host failures. Furthermore, this approach also allows the coordination to be implemented as a plug-in to existing virtualization managers. More specific implementation details and quantitative benefits are presented in Sections 3 and 4.

2.3 Virtualization-aware Platform Management

We now explain the coordinator plug-in at the platform manager. This plug-in implements policies for providing better runtime management considering sensor data and actuators from both the platform and virtualization management domains. Below is the specific operation.

Periodically, the coordinator in the platform manager interfaces with the registry & proxy service and obtains access to application SLA sensors deployed in guest VMs (in addition to the power sensors). Two types of SLA notifications are used in our prototype: *SLA violation* and *SLA restore*. SLA *violation* is generated when a VM's or its application's SLA is violated. SLA *restore* on the other hand is generated when a VM's or its application's SLA metrics (e.g., application response time) are "well below" violation levels. In addition, the coordinator also queries the registry & proxy service to obtain access to the multiple actuators in the system across the platform and virtualization layers (ACPI p-states and VM migration). Given such an access to monitoring data and actuators from both the platform and virtualization layers, the coordinator in the platform manager implements *SLA-based* power management decisions, and executes a consolidation VM migration policy that deals with both SLA *and* power violations. More specifically, the coordinator executes the following policy -- if it receives a SLA *violation* notification from a VM, it increases the frequency of the CPUs running the VM using ACPI p-state actuators; on the other hand, if it receives a SLA *restore* notification or a power violation notification, it decreases the frequency of the CPUs also using ACPI p-state actuators; however if either the SLA or power violation has been persisting for long enough beyond an acceptable threshold, the coordinator triggers a different actuator - VM migration, and contacts the coordinator agent plug-in at the virtualization manager to choose an appropriate host to which to migrate the VM, considering both the VM and platform metrics.

With such a policy, we ensure that local scaling p-state knobs are first employed for runtime management decisions, and if they do not satisfy either SLA or platform requirements, we use global VM migration knobs and leverage other available nodes in the data center to achieve combined goals. Our approach improves on traditional platform management solutions (e.g., HP's PowerRegulator) that typically make decisions only based on lower level metrics obtained through hardware sensors and performance counters, with no direct feedback from higher virtualization layers. Similarly, our solution also improves on traditional virtualization management solutions (e.g., VMWare DRS, which manages VM migrations to meet performance goals) that make their decisions based on higher level metrics only obtained through virtualization and application sensors. These traditional solutions do not leverage local platform management knobs (ACPI p-states, reliability, etc.) in a combined manner with SLA goals to achieve their runtime objectives. With the policies executed in the coordinator, vManage avoids redundant actions (e.g, migrations and power management knobs enabled at the same time), improves data center efficiency (power usage, number of migrations) and restricts violations (SLAs and power).

2.4 Stability for Coordinated VM Placement

In typical settings, the resource and platform requirements associated with a hosted application, and therefore the container VM, may vary with time. Such variations, if not accounted for, can quickly render a seemingly sound initial placement or migration decision into one that causes further violations (SLA, power, etc.) on the new host and results in more migrations. Similarly, a failure to recognize the transient nature of variations, which manifest themselves as intermittent SLA and/or power violations, may lead to several unnecessary VM migrations. It is therefore important that the decision on when, which VM, and where to migrate should not only be based on the prevailing operational conditions but also on the validity of such a decision into some time into the future. We define the *stability* of a decision (placement or migration) as the probability with which such a decision continues to remain valid over certain duration into the future. The intuition is that more stable decisions will lead to a more stable system, and thereby prevent unnecessary VM migrations. The remainder of this section discusses how vManage formalizes the notion of stability outlined above.

Let $\mathbf{M}(\mathbf{m}_1, \ldots, \mathbf{m}_l)$ represent the set of l VMs deployed on a particular host, and $\mathbf{R}(\mathbf{r}_1, \ldots, \mathbf{r}_n)$ be the set of n resources (e.g. CPU, Memory, etc.) that affect the placement or migration decisions. We represent the resource requirements for a VM $\mathbf{m}_i \in \mathbf{M}$ using the set $\mathbf{V}^i$, where each element $\mathbf{v}_j^i(\mathbf{t})$ represents the time varying requirement for the resource $\mathbf{r}_j \in \mathbf{R}$. We also define the set **A**, where each element $\mathbf{a}_j(\mathbf{t})$ represents the amount of resource $\mathbf{r}_j \in \mathbf{R}$ available with a host with no VMs deployed on it[1].

For a resource requirement $\mathbf{v}_j^i(\mathbf{t})$, since its variation is not known exactly, we model it using a time varying mean $\boldsymbol{\mu}_{\mathrm{v}_j^i}(\mathbf{t})$, a standard deviation $\boldsymbol{\sigma}_{\mathrm{v}_j^i}(\mathbf{t})$, and a corresponding probability density function (PDF) expressed as $f_{\mathrm{v}_j^i}(\mathbf{x},\mathbf{t})$. We assume that $\boldsymbol{\mu}, \boldsymbol{\sigma}$ and f which correspond to a VM are known *a priori* or can be calculated by making use of methods like offline profiling or online calibration [36,37,10]. Using the PDF one can calculate the probabilistic resource requirement at time instant **t**. For instance, by plugging-in $\mathbf{x} = \mathrm{x}_0$ and $\mathbf{t} = \mathrm{t}_0$ one can calculate the probability with which the VM represented by $\mathbf{m}_i$ will require x_0 amount of resource $\mathbf{r}_j$ at time t_0.

To make further computations easier and more comprehensible, we now define a few additional variables that are derived from the ones defined above. In particular, we make use of the cumulative distribution function (CDF) $F_{\mathrm{v}_j^i}(\mathbf{x},\mathbf{t})$ which can be derived from a PDF [38]. Using the CDF with $\mathbf{x} = \mathrm{x}_0$ and $\mathbf{t} = \mathrm{t}_0$ one can find the probability with which the VM $\mathbf{m}_i$ will require x_0 or lesser amount of resource $\mathbf{r}_j$ at time t_0. We also define $\mathbf{v}_j^{\mathbf{M}}(\mathbf{t})$ which represents the net requirement for the resource $\mathbf{r}_j \in \mathbf{R}$ by all the VMs in **M**. Furthermore, the quantities $\boldsymbol{\mu}, \boldsymbol{\sigma}, f$ and F are also defined for the net resource requirement variable $\mathbf{v}_j^{\mathbf{M}}(\mathbf{t})$, an instance of specific derivation of these will be presented later in the section.

The stability of a host, which corresponds to the decision of assigning a set **M** of VMs to a particular host, can be calculated as the average probability with which the host can continue to provide sufficient resources to the VMs in **M** for a given time **T** into the future. Mathematically, this probability, for the time interval (t_0, t_0 + **T**), can be calculated using the following product equation.

$$\mathbf{P}(\mathrm{t}_0, \mathrm{t}_0 + \mathbf{T}) = \prod_{\mathbf{r}_j \in \mathbf{R}} (\mathbf{p}_{\mathbf{r}_j}(\mathrm{t}_0, \mathrm{t}_0 + \mathbf{T})) \tag{1}$$

Where the individual probability in turn is,

$$\mathbf{p}_{\mathbf{r}_j}(\mathrm{t}_0, \mathrm{t}_0 + \mathbf{T}) = \frac{\int_{\mathrm{t}_0}^{\mathrm{t}_0+\mathbf{T}} F_{\mathrm{v}_j^{\mathrm{M}}}(\mathbf{a}_j(\mathbf{t}), \mathbf{t})\mathbf{dt}}{\mathbf{T}} \tag{2}$$

Here, $F_{\mathrm{v}_j^{\mathrm{M}}}(\mathbf{x},\mathbf{t})$ represents the CDF corresponding to the net resource requirement $\mathbf{v}_j^{\mathbf{M}}(\mathbf{t})$. The above computations essentially calculate the combined probability with which the net resource requirement $\mathbf{v}_j^{\mathbf{M}}(\mathbf{t})$ for the resources $\mathbf{r}_j \in \mathbf{R}$ can be satisfied by the resources $\mathbf{a}_j(\mathbf{t})$ available with the host in question.

To see how the above computations can be put into practice, consider the scenario where the VMs in **M** are running independent workloads (implying independent resource requirements). Therefore, the mean and the standard deviation for the net resource requirement $\mathbf{v}_j^{\mathbf{M}}(\mathbf{t})$ corresponding to a resource $\mathbf{r}_j \in \mathbf{R}$ can be calculated by summing the corresponding quantities from the VMs in **M**.

$$\boldsymbol{\mu}_{\mathrm{v}_j^{\mathrm{M}}}(\mathbf{t}) = \sum_{\forall i|\mathbf{m}_i \in \mathbf{M}} \boldsymbol{\mu}_{\mathrm{v}_j^i}(\mathbf{t}) \tag{3}$$

$$\boldsymbol{\sigma}^2_{\mathrm{v}_j^{\mathrm{M}}}(\mathbf{t}) = \sum_{\forall i|\mathbf{m}_i \in \mathbf{M}} \boldsymbol{\sigma}^2_{\mathrm{v}_j^i}(\mathbf{t}) \tag{4}$$

Further, if we assume that the PDF $f_{\mathrm{v}_j^i}(\mathbf{x},\mathbf{t})$ for any time instant **t** follows a normal distribution (this worked well for the workloads used in our experiments when CPU resource usage was considered), then we can calculate the CDF for the net resource requirement $\mathbf{v}_j^{\mathbf{M}}(\mathbf{t})$ using standard statistics [38]:

$$F_{\mathrm{v}_j^{\mathrm{M}}}(\mathbf{x},\mathbf{t}) = \frac{\mathbf{1}}{\mathbf{2}}(\mathbf{1} + \mathbf{erf}\,\frac{\mathbf{x} - \boldsymbol{\mu}_{\mathrm{v}_j^{\mathrm{M}}}(\mathbf{t})}{\boldsymbol{\sigma}_{\mathrm{v}_j^{\mathrm{M}}}(\mathbf{t})\sqrt{\mathbf{2}}}) \tag{5}$$

The error function in this term – erf – cannot be evaluated in closed form, and consequently, this requires discretization of the quantities when Equation 5 is substituted in Equation 2. Thus the resulting formulation of stability for a host hosting a set **M** of VMs can be summarized by the following equation. As mentioned earlier, it calculates the average probability with which the host can continue to provide sufficient resources to the VMs in **M** for a given time **T** into the future.

$$\mathbf{P}(\mathrm{t}_0, \mathrm{t}_0 + \mathbf{T}) = \prod_{\mathbf{r}_j \in \mathbf{R}} \frac{\sum_{\mathrm{t}_0}^{\mathrm{t}_0+\mathbf{T}} F_{\mathrm{v}_j^{\mathrm{M}}}(\mathbf{a}_j(\mathbf{t}), \mathbf{t})}{\mathbf{T}} \tag{6}$$

One can calculate the stability of system composed of multiple hosts by calculating the product of the stability of the individual hosts. Whenever a new VM arrives for initial deployment or when a VM needs to be migrated, the stabilizer generates a new set of assignments by adding the VM in question to the set **M** corresponding to different hosts. The new assignment which results in the highest value for system stability among the different hosts is chosen for deployment. The approach can be easily extended to handle arrival of multiple VMs for initial deployment or when multiple VMs need to be migrated.

2.5 Discussion

The vManage design approach as outlined in this section provides several qualitative benefits summarized below.

vManage takes a structured approach to designing cross-layer coordination, thereby eliminating or improving on manual or ad-hoc approaches. Additionally, vManage uses standards-based protocols and leverages existing CIMOM [25] deployments (more in Section 3) to ensure broader community adoption for greater automation benefits.

vManage provides a framework whereby sensors and actuators from multiple vendors and types across multiple domains can be added and removed from the system easily without any dependencies. This is achieved through the registry & proxy service discovering their presence or absence and informing the policy system in the coordinators accordingly.

[1] We model the platform resources as a function of time because these can change due to events like CPU frequency scaling, failure of a disk in a disk array, etc.

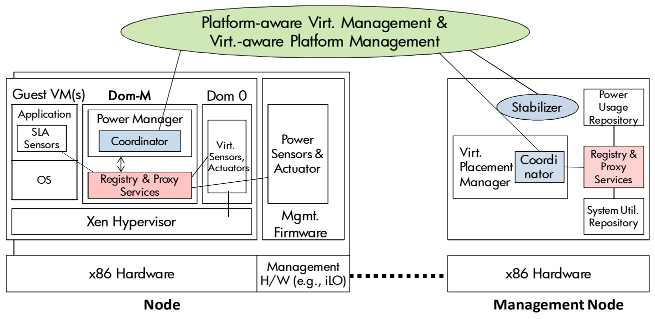


Figure 3: Prototype implementation.

Additionally, the separation of the registry & proxy services from the policy execution in the coordinators allows the policies to be developed without having to deal with low-level implementation issues, such as how to communicate, how to discover the diverse sensors across the virtualization and platforms domains, how to access diverse software vs. hardware sensors, etc. This also aids in portability.

Also, coordinators can be merged with existing controllers as loosely-coupled plug-ins. From a practicality perspective, this preserves prior industry investments in existing products and leverages existing industry trends (e.g., VMWare DRS [7], HP SIM [4] plug-and-play architecture).

vManage is also unique in its implementation of a stabilizer that avoids system oscillations (in terms of continuous VM migrations) when reconciling different individual local constraints and transient workload behavior. Our results in Section 4 discuss how the stabilizer module is key to minimizing the number of VM migrations while achieving effective coordination.

Finally, vManage facilitates coupling of several different policy actions and actuators across multiple management solutions. For example, it facilitates coordination between power, thermal, performance, and reliability management solutions. While our current prototype demonstrates the unification of a limited set of sensors/actuators, additional sensors and actuators such as those for shutting down machines with VM consolidation, fan control, and reliability monitoring can be integrated into the prototype and coordination policies related to these can be implemented in a fairly straightforward manner.

3. PROTOTYPE IMPLEMENTATION

The implementation of vManage is based on Xen (Figure 3). The virtualization-aware platform manager is implemented in a separate privileged driver domain in Xen (*Dom-M* in Figure 3), and the platform-aware virtualization manager is implemented on a management node. We explain the implementation for these two managers next.

The virtualization-aware platform manager on each node relies on IPMI [28], XenMon [29], and application SLA tools for sensor data. XenMon (for VM utilization) is deployed in Dom0, whereas IPMI and special management hardware drivers (such as to access the management processor [13]) are deployed in Dom-M for power data. The application SLA sensors are deployed in each of the guest VMs. To implement these sensors, we wrote Perl scripts that monitor application throughput and response time logs and infer SLA *violation* or *restore*. Specifically, they capture the average execution time for the application periodically (e.g., every 1 second), and record a SLA notification whenever the response time exceeds the SLA threshold. For example, when response time is above 10ms, a violation notification is generated and when the response time is below 50% of 10ms, a restore notification is generated. We have written these SLA sensors for all the applications used in our study.

The registry and proxy service for the virtualization-aware platform manager is instantiated within Dom-M at each node. It relies on CIM (Common Information Model) [25] to connect with the diversity of available sensors and actuators. We use CIM since it appears to be the common standard on which the industry is converging; though our solution can work equally well with other standards. Specifically, a CIM object manager (CIMOM) is installed in each of the guest VMs, as well as in Dom0, Dom-M, and the management hardware. Sensors and actuators in each of these respective domains register to their local CIMOM through CIM providers and are exposed to the outside world through CIM classes. The registry and proxy service then discovers these individual CIMOMs using the Service Location Protocol (SLP) [40]. Once discovered, the registry and proxy service contacts the individual CIMOMs and through the CIM classes, it gathers the meta-data information in the various management domains. In this way, the registry and proxy service is able to gather the diverse meta-data across different management entities at a single unified location. When a request is made for monitoring or actuation, the registry & proxy service redirects the call to the appropriate CIMOM in the respective domain and the monitoring is initiated. The registry and proxy service is able to deal with dynamics (e.g, dynamic addition of VMs or migrations) through the use of leases as part of the discovery process of individual CIMOMs.

The management node hosts the power usage and system utilization repository as LDAP directories. Simple daemons in Dom-M periodically send the power usage, XenMon utilization data, and other attribute values (e.g., reliability) of each node to the management node. This aggregated information for all nodes is stored in the LDAP directories. A separate registry and proxy service is implemented at the management node for the platform-aware virtualization manager. Its current implementation is straightforward and the information about the available repositories is known apriori to the registry and proxy service.

The coordination support for the virtualization and platform managers is implemented at application level, as multithreaded processes where one or more threads perform the monitoring, protocol exchange, coordination, and actuation tasks. They interface with the registry service and bind to the API libraries. The policies described in Section 2 are implemented in the coordinators as C++ functions. We also implemented *traditional* platform and virtualization managers into which the coordinators are statically compiled to make the virtualization manager platform-aware and the platform manager virtualization-aware. Although the prototype currently requires a static compilation of the coordinator, this limitation is being removed through our current efforts to architect the code using commercial plug-in frameworks [39]. This will provide support for dynamic and on-demand plug-ins.

The traditional (non-coordinated) virtualization manager implementation is loosely-based on VMware DRS. (We implemented our own equivalent of this manager given that our prototype was on Xen.) This solution does initial VM placement by considering the VM resource utilization requirements. When CPU utilization exceeds a certain threshold (>80%), it triggers a

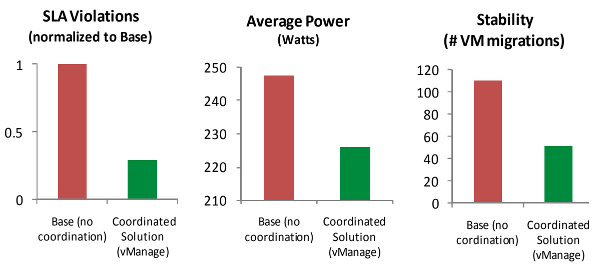


Figure 4: Results. *The vManage architecture achieves more power savings and significantly better QoS while negating the potential ill-effects of migrations through the stabilizer, all at low overhead and with minimal interference to host workloads.*

VM migration to maintain SLAs. The first host meeting the migrating VM's requirements is chosen as the destination.

The traditional power manager implementation does utilization-based power regulation and power budget capping and is loosely-based on HP's PowerRegulator and PowerCapper, but with APIs exposed for coordination. For power regulation, if the system utilization exceeds a certain threshold (80%), CPU frequency is increased by one level, and when CPU utilization drops below a certain threshold (20%), CPU frequency is decreased by one level. (To obtain system utilization, our implementation uses values reported by XenMon; commercial products use hardware performance counters for this purpose, but the functionality is the same.) The power capper periodically monitors the power budget and reduces CPU frequency by one level on a power budget violation. As mentioned earlier, this power manager is deployed in Dom-M, and the virtualization manager is hosted on the management node.

The stabilizer is written as a separate C++ application and is deployed on the management node. It implements the algorithms described in Section 2.4. Specifically, it considers the probability of each host satisfying the migrating VM or new VMs' requirements over a certain period in the future (this period is kept to 15 minutes in our experiments) and picks the host with the highest probability. In the experiments reported in this paper, the stabilizer uses a priori knowledge of the workload model (by analyzing workload traces offline) to arrive at VM resource utilization model. Only the CPU resource type is considered during probability calculation. Experimental studies considering other resource types (e.g., memory) are planned for future work. The coordinator in the virtualization manager requests the stabilizer application for probability values during placement decisions. Similarly, the per-node coordinator in the power manager requests the virtualization manager coordinator whenever a VM migration placement decision needs to be made.

Overall, our prototype consists of about 4000 new/modified C++ and Perl script lines of code.

4. EVALUATION

4.1 Experimental Setup and Results

vManage is evaluated in an experimental testbed consisting of thirteen machines connected through a gigabit network – seven of these are Dell PowerEdge 1950 (compute nodes) which host virtual machines, four of them are client machines generating workload requests for the VMs, and the final two are configured as a storage node (providing VM disk images over NFS), and a management node (running the platform-aware virtualization manager), respectively. The compute nodes are dual-core dual-socket machines containing Intel Xeon 5150 processors with support for three different operating frequencies (2.66 GHz, 2.33 GHz, and 2.0 GHz) and have 4 GB of memory each.

Multiple workloads are used to emulate the typical data center environment. They include the RUBiS [30] online auctioning application (representative of a business application), the Nutch [31] search engine (Web 2.0), WebServer (Internet services) serving static files, and batch mode applications running tight CPU computation (engineering, HPC). RUBiS is a three tier application with an Apache webserver, Tomcat application server, and a MySQL database server. Two application servers are used between the webserver and database server for load balancing. All servers are run inside VMs. Hence, one RUBiS instance consists of 4 VMs. All VMs are uni-processor VMs and depending on the workload, have different requirements for CPU cycles and memory.

We use two end-user request traces to generate workload for the VMs: (i) EPA-HTTP web traffic trace from the LBL Repository [26] to generate workload for the RUBiS applications and (ii) request traces from the Travelport [27] website to generate workload for Nutch and for the static WebServers. The two web traces contain traffic for more than an entire day. We replay these day-long traces multiple times to model a 50-day experiment, and to save time condense the 50 day experiment into a 20 hour experiment (by condensing the 24-hour trace into a 24 minute trace while preserving its shape and other properties). All of the workloads are primarily CPU bound since we are currently considering only CPU resource.

For our power measurements, in the absence of real-time power metering capabilities in the Dell systems in our testbed, we use a model based on CPU utilization that has been proposed and validated in several previous studies (e.g., [5,6,17]). Specifically, our power sensor assumes power consumption varies according to $P = K*U + I$, where P is consumed power, K is a constant, U is the CPU utilization and I is the idle power. K depends on the current operating frequency (i.e., p-state) of the CPU. We determine the values of K and I offline, by calibrating the hosts using a power meter. While not presented here, we also developed a smaller testbed with HP-based servers that do support real-time power metering through the iLO management processor, and our results were qualitatively similar.

Figure 4 summarizes the overall benefits in data center efficiency (power savings, VM migrations) and VM guarantees (SLA violations) gained from coordination. For these experiments, we used 10 instances of Nutch, 3 instances of RUBiS, and 6 instances of static WebServers, resulting in a total of 28 VMs (12 RUBiS VMs, 10 Nutch VMs, and 6 WebServer VMs) on 7 nodes in the ``mini" data center. The *base* case represents the traditional virtualization and power managers without the coordinator plug-ins – these model commercial products in the market currently. The coordinated case is our proposed vManage solution obtained by enabling the registry & proxy service, and the coordinator plug-ins in our prototype. Each result corresponds to the average of four 20-hour runs.

As seen in Figure 4, the vManage solution offers significant benefits. The number of SLA violations in the coordinated solution is reduced by 71% compared to the base case (Figure 4a). The average power reduces from about 250W to 225W, providing

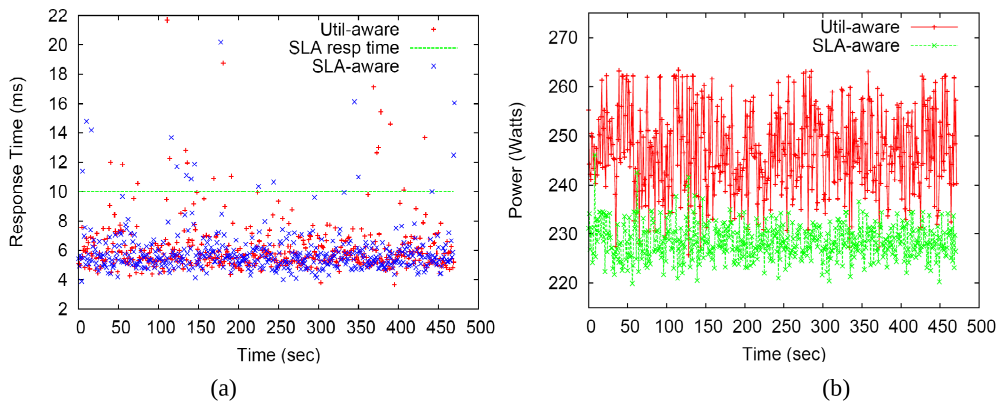


Figure 5: SLA-based power regulation.

approximately 10% power savings (Figure 4b). In addition, as seen from Figure 4c, the number of VM migrations decreases dramatically – by 54% -- from 110 in the base case to only 51 in the coordinated case.

These benefits can be attributed to three key characteristics in the vManage coordination solution – richness of information, richness of actuators, and the stabilizer. We next consider each of these characteristics and present results from controlled sub-experiments that characterize how they individually benefit the efficiency and QoS of the system.

Richness of information in coordinated solution: vManage's coordinated solution has access to information from sensors across the platform *and* virtualization domains. Policies implemented using such a holistic view of the system have system-wide impact by improving *both* data center efficiency and VM QoS metrics, as opposed to improving only one of them. We illustrate this point through a controlled experiment that shows the benefit of using vManage's application SLA feedback (richer information) compared to the traditional processor utilization usage (obtained through performance counters by current power management products) for power regulation. This experiment uses one instance of RUBiS and one instance of the Nutch server on a single host.

Figure 5a shows the response time of the application for both the power regulation policies. For visual simplicity, we show only Nutch; RUBiS' behavior is similar. The SLA threshold is set at 10 ms, so any request execution time above 10 ms is considered an SLA violation. We see that the request execution time and SLA violation characteristic are similar for both policies. Concerning power consumption, however, Figure 5b, which plots the power consumption trace over time for both policies, demonstrates that host power consumption under the SLA-based policy is less than under the utilization-based policy. The SLA-based policy reduces power consumption by 8%. These improvements reflect the fact that high utilization is not often the most appropriate metric to approximate higher-level application SLAs. An application at high utilization may still be maintaining its SLAs and may not need a higher CPU frequency. As a result, power regulation directly done based on SLA violations can often run the CPU at lower frequencies even under high CPU utilization.

This example, though straightforward, serves to illustrate how richer semantic information contributes to improved results in our baseline. Similarly, though not plotted here in interests of space, our controlled experiments demonstrate that the richness of information at the virtualization manager helps it make better placement decisions that not only satisfy VM SLAs, but also reduce power violations.

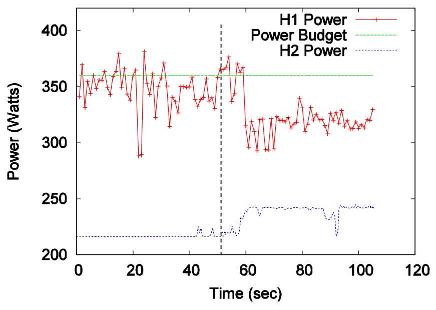
Richness of actuators in coordinated solution: For this controlled experiment, similar to the SLA-based power regulation controlled experiment; we host one instance of RUBiS and one instance of the Nutch server on a single host. However, in addition, we also run two batch mode applications in two other VMs on the host to increase power consumption so that it violates the power budget (360 watts) set for the hosts. In the first instance, we execute a traditional power capper which on a power budget violation reduces the CPU frequency. This, however, causes SLA violations. In the base case, this SLA violation, detected through an increase in system utilization, is handled concurrently through two mechanisms: (i) a VM migration triggered by the virtualization manager, and (ii) a CPU frequency increase triggered by the power manager. Both of these actions take place in a non-coordinated manner, which results in poor efficiency.

The triggering of a VM migration (mechanism (i)) as soon as the utilization exceeds a threshold is naïve, since the SLA violation can likely be handled by just increasing the local CPU clock frequency. This would have avoided the heavy-weight VM migration. However, since the virtualization manager does not have access to platform actuators (ACPI p-states), it is limited to a solution that uses the heavy-weight migration operation only.

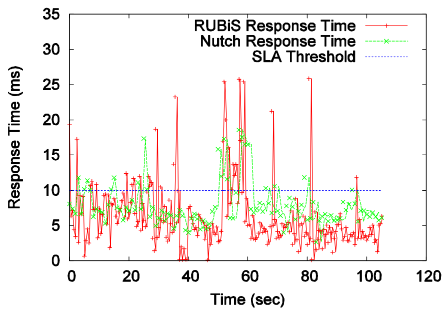
Similarly, the second option of increasing the CPU frequency (mechanism (ii)) can also lead to system oscillations when employed in a non-coordinated manner as described in Figure 1 (Section 1). Such oscillations can be broken if the power manager has access to additional actuators - VM migration in this case - which would help reduce power consumption below the allocated budget without loss in performance. However, since the power manager in the base case is limited in its actuators, both power and SLA violations continue to happen.

Furthermore, in the worst case, it is possible that both mechanism (i) and mechanism (ii) are activated at the same time, resulting in a duplication of actions to address the SLA violations. This leads to unnecessary cost (migration) and wastage of power (p-state increase).

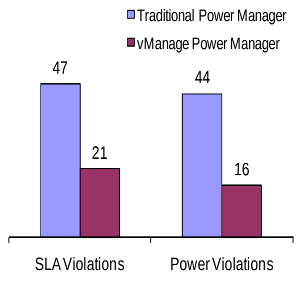
In order to mitigate these problems with the traditional approaches, we run a second instance that uses vManage. Figure



(a)

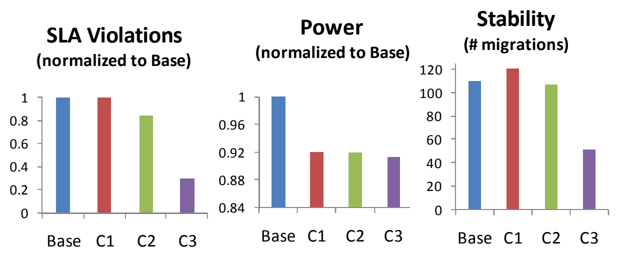


(b)

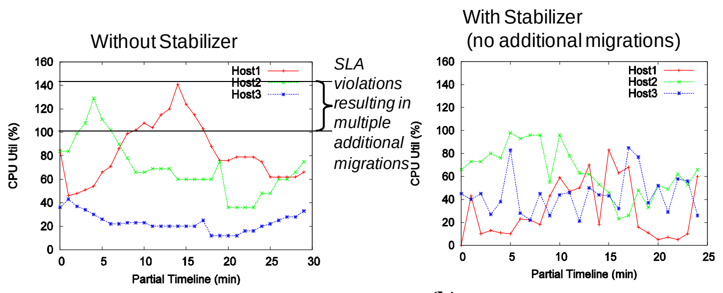


(c)

Figure 6: Virtualization-aware power capping.



(a)



(b)

Figure 7: Illustrating effects of placement algorithms and benefits of stabilizer.

6a shows the results. As seen in the figure, local power management p-state actuators are used until the 52nd second, but after this point, the SLA and power violations at Host 1 cross the threshold that triggers a VM migration of the RUBiS AppServer. This finally brings down power consumption, and it reduces the SLA violation of the Nutch server significantly. The variation in the RUBiS application response time is shown in Figure 6b as it migrates from Host H1 to H2. We see that the migration lasts for about 8 seconds. Note that during this period, RUBiS experiences very high response times because of the VM migration costs (shadow mode paging, VM suspension, and network bandwidth), again demonstrating the importance of reducing the number of migrations. The response time behavior for Nutch application is also shown in Figure 6b.

Figure 6c shows how both power violations and SLA violations are reduced with vManage making it desirable from both a datacenter operator and VM manager's perspective. More generally, the experiment demonstrates vManage's coordinated approach leveraging multiple actuators (p-states, migrations) and coordinating them for multiple metrics (SLA, power violations) enables better SLA and power savings, while reducing heavy-weight VM migrations.

Effects of stabilizer in coordinated solution: We next illustrate the benefits of the stabilizer for our improved coordination solution results. Figure 7a shows the evaluation of three placement strategies in the same experimental setup as for the baseline results in Figure 4. The base case is also repeated for reference. The remaining policies all use the same virtualization-aware platform manager, but differ in the placement strategy at the platform-aware virtualization manager, both during initial provisioning and migration. *Coord1* (C1) uses "first fit" - the first host that satisfies both VM and power budget requirements, *Coord2* (C2) uses "best fit" – the most idle host for a VM satisfying power budget requirements, while *Coord3* (C3) uses the stability-aware equations developed in Section 2.4 when deciding which host to pick.

As seen in Figure 7a, the use of *Coord1* results in significantly better power savings compared to the base case. However, it also incurs slightly more SLA violations and VM migrations because of the reactive nature of the SLA-based policies as compared to the proactive utilization based policies of the base. However, we see that *Coord2* can mitigate this drawback through a better placement by using a best-fit destination host matching both VM and power requirements compared to a first fit algorithm. This reduces the total number of SLA violations and VM migrations significantly. *Coord3* takes this one step further; the stabilizer uses the probability of a host satisfying VMs' CPU requirements for 15 minutes (in the condensed time) in the future when making decision to minimize migrations. This results in an additional 55% reduction in SLA violations and 53% fewer VM migrations compared to the *Coord2*, and 71% reductions in SLA violations and 54% fewer VM migrations compared to base.

Figure 7b further highlights this point through a timeline graph of CPU utilization for 3 hosts. Without the stabilizer, we see that a bad placement leads to workloads not finding sufficient CPU resources to meet their required workload demands (indicated by the total required utilization on the hosts exceeding 100%). This leads to a performance loss (SLA violations) which is then handled by the system through VM migrations. With the stabilizer, the good placement ensures sufficient resources are available to meet the workload's demands. As a result, SLA violations are reduced and additional VM migrations are avoided.

4.2 Quantifying vManage Overheads

The previous sub-section demonstrates the benefits of the coordination architecture in vManage. A key question, however, pertains to the additional overhead added for vManage mechanisms and policies. We performed several measurements and additional controlled experiments to determine the overheads of our solution. Our results are summarized below:

- The static phase during which only the registry services are active, performing discovery and meta-data registration has very negligible overhead (<1%), and it has almost no impact on the runtime operation of the system workloads.
- The CPU usage for runtime unified monitoring when initiated by the virtualization-aware platform manager is less than 2%. Similarly, the additional latency for monitoring introduced by our architecture is very modest (< 1%). These small overhead numbers were observed even when the number of VMs was varied from 1 to 32.
- The average CPU usage for policy execution at the coordinator in the virtualization-aware platform manager is less than 5% for typical policies, including under stress conditions.
- For the coordinator providing platform-aware virtualization management at the management node, we ran a simulation experiment where we increased the number of hosts from 100 to 2500. We found that the CPU usage and the policy execution time increase linearly, with the maximum values for these metrics, at 2500 hosts, being 8% and 6ms, respectively. These numbers include the effects of the stabilizer as well. This shows that the coordinator plug-in and stabilizer scale well for sizes typical for small data centers. For larger systems, a data center may be divided into multiple smaller sizes, separately managed clusters, coordinated in a tree-like structure.
- vManage has negligible interference on the host workloads; measurements on our testbed showed no overheads on performance when vManage was enabled.

4.3 Summary

Overall, the results demonstrate that the vManage solution achieves more power savings and significantly better QoS with better stability than silo-ed solutions. Specific results show over 71% less SLA violations, with 10% power savings, and 54% fewer migrations. Our individual experiments demonstrated how these overall benefits can be related back to the richness of information and actuators provided by the coordination solution and the dynamic assessment of decisions by the stabilizer. Furthermore, vManage achieves these benefits at low overhead (<5%), and minimal interference to the host workloads. In addition, the loose-coupling provided by our approach provides several qualitative advantages like minimal disruption to existing management controllers and entities, transparency to low-level implementation issues, and extensibility for future sensors, actuators, and controllers.

5. RELATED WORK

To the best of our knowledge, our work is the first to consider a more systematic loosely-coupled and practical approach to coordination across platform (hardware) and virtualization management (software) layers. The loosely-coupled coordination approach provides the advantages of *working* with most of the existing management infrastructures, easy *plug-and-play* of coordination policies, and *reduction in dependencies* among individual controllers as well as the coordinator in normal system operation. We are also the first to consider a *stability* formulation to reduce the number of migrations in a coordinated solution. Furthermore, unlike several previous studies that rely on simulation, we implement our approach in a prototype and present results with enterprise benchmarks running on a large number of virtual machines.

There are several platform and virtualization management solutions today – some of these are available as industry products [4,7,8,13] and many others are published in the research literature [9,10,14,15,16,17]. However, these solutions exist in isolated silos and represent partial sub-system level solutions. Indeed, the individual uncoordinated controllers in our baseline experiments model many of these approaches. Where attempts have been made to provide more unified management, e.g., with VMWare DPM [32], the solutions are designed to deal with limited actuators only (e.g. only the power shut-down actuator for DPM). vManage on the other hand provides a holistic framework that can be leveraged by existing management products for the use of multiple sensors and actuators so as to provide more effective data center management. There has also been a large body of work on distributed middleware and publish-subscribe solutions proposed in the literature [33,34,35]. While vManage is the first to consider these in the context of datacenter coordination, we leverage these known mechanisms where appropriate. Our work similarly leverages industry standards and emerging protocols [25,34] where appropriate.

A few recent studies have started addressing issues surrounding coordinated management across the platform and virtualization layer. Raghavendra et al. [5] and Nathuji et al. [6,18] have developed point solutions addressing the coordination among power controllers and the virtualization layer. Verma et al. [19] propose power & migration cost aware application placement addressing various policies using simulation. Kephart et al. [20, 21] address the coordination of multiple autonomic managers for power/performance tradeoffs by using a utility-function approach in a non-virtualized environment. Gmack et al. [22] take an integrated approach to resource pool management by combining a workload placement controller with a reactive controller. Chen et al. [23] conduct energy-aware server provisioning for Live Messenger workloads. Finally, GRACE [24] has explored global resource adaptation through cooperation and demonstrated benefits of power and performance coordination for multimedia workloads. None of these approaches have taken a systematic systems/architecture approach to the coordination problem across the hardware-software layer. They have focused on integrated or ad-hoc solutions or have only demonstrated modeled or simulated results. Further, several of these approaches have not considered virtualized environments or implementations on actual hardware and testbeds.

6. CONCLUSIONS AND FUTURE WORK

Current industry offerings and most academic work on platform and virtualization management have been designed in isolation without any emphasis on the interaction across these layers. The benefits from cross-layer coordination – in terms of better performance, stability, and efficiency – motivate new solutions for future data centers.

In this paper, we present vManage, a loosely-coupled solution for achieving coordinated cross-layer management across virtualization and platform managers. Our solution makes several design choices to address practical considerations in future datacenters including compatibility (and minimal disruption) to existing multi-vendor management controllers and entities, transparency to low-level implementation issues, and extensibility for future sensors, actuators, and controllers. In particular, our design incorporates the use of a (1) registry and proxy service to discover and register the individual sensors and actuators in the various layers, and (2) coordination plug-ins that can implement federation or loose-coupling across individual controllers, including a stabilizer that provides a formal approach to avoid excessive VM migrations and redundant actions. To the best of our knowledge, we are not aware of any prior work that has examined such a design. Our solution has been implemented in Xen, and evaluation results show significant improvements in overall data center efficiency and in meeting service-level agreements, at small additional overhead compared to unmanaged or silo'ed systems.

As part of future work, we are planning to release to the broader research community, a standards-based coordination toolkit encapsulating the design that we discussed in this paper. We are also working on extending the framework for larger scale data centers using distributed coordinators and scalable registry and proxy services, and incorporating trust and security.

References

[1] Gartner TCO Reports. 2005-08. www.gartner.com/

[2] C. Clark *et al*. Live migration of virtual machines. NSDI'05.

[3] S. Kumar and K. Schwan. Netchannel: a VMM-level mechanism for continuous, transparent device access during VM migration. In VEE, 2008.

[4] HP Proliant Esentials, Systems Insight Manager. www.hp.com/go/proliantessentials, www.hp.com/go/sim

[5] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No "power" struggles: coordinated multilevel power management for the data center. In ASPLOS, 2008.

[6] R. Nathuji and K. Schwan. VirtualPower: Coordinated power management in virtualized enterprise systems. In SOSP'07.

[7] VMWare Virtual Center. www.vmware.com/products/vi/vc/

[8] Microsoft Virtualization Management. www.microsoft.com/VIRTUALIZATION/solution-tech-management.mspx

[9] T. Wood *et al.* Black-box and gray-box strategies for virtual machine migration. In NSDI, 2007.

[10] P. Padala *et al*. Adaptive control of virtualized resources in utility computing environments. Eurosys 2007.

[11] V. Kumar *et al*. A state space approach to SLA based management. In NOMS, 2008.

[12] J. Hellerstein *et al.* Feedback Control of Computing Systems. John Wiley & Sons, 2004.

[13] HP iLO Management Processor, www.hp.com/go/ilo

[14] J. Chase *et al.* Managing energy and server resources in hosting centers. SOSP 2001

[15] J. Donald and M. Martonosi. Techniques for multicore thermal management: Classification and new exploration. In ISCA 2006

[16] C. Lefurgy *et al.* Energy management for commercial servers. In *IEEE Computer,* pp. 39-48, December 2003.

[17] P. Ranganathan *et al.* Ensemble-level power management for dense blade servers. In ISCA 2006.

[18] R. Nathuji and K. Schwan. VPM tokens: virtual machine-aware power budgeting in datacenters. In HPDC 2008.

[19] A. Verma *et al*. pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems. In Middleware 2008.

[20] Jefferey O. Kephart *et al*. Coordinating Multiple Autonomic Managers to Achieve Specified Power-Performance Tradeoffs. In ICAC 2007.

[21] R. Das *et al*., Autonomic multi-agent management of power and performance in data centers. In AAMAS 2008

[22] D. Gmack *et al*. An Integrated Approach to Resource Pool Management: Policies, Efficiency and Quality Metrics. In DSN 2008.

[23] G. Chen *et al*. Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services. In NSDI 2008.

[24] S. Adve *et al*. The Illinois GRACE project: Global resource adaptation through cooperation, SHAMAN 2002.

[25] Common Information Model. http://www.dmtf.org/standards/cim/

[26] EPA-HTTP - a day of HTTP logs from EPA WWW server. http://ita.ee.lbl.gov/html/contrib/EPA-HTTP.html.

[27] Worldspan by Travelport. http://www.worldspan.com

[28] Intelligent Platform Management Interface (IPMI). http://www.intel.com/design/servers/ipmi/index.htm.

[29] D. Gupta, R. Gardner, and L. Cherkasova. Xenmon: QoS monitoring and performance profiling tool. Technical Report HPL-2005-187, HP Labs, 2005.

[30] RUBiS: Rice University Bidding System. http://rubis.objectweb.org/

[31] Nutch: Open Source Web-Search Software. http://lucene.apache.org/nutch/

[32] VMWare Distributed Power Management (DPM). http://www.vmware.com/files/pdf/DPM.pdf

[33] ObjectManagementGroup, The Common Object Request Broker: Architecture and Specification, 2.0 ed, July 1995.

[34] Web-based Enterprise Management (WBEM). http://www.dmtf.org/standards/wbem/

[35] P.R. Pietzuch "Hermes: A Scalable Event-Based Middle-ware". Ph.D. thesis, Computer Laboratory, Queens' College, University of Cambridge, February 2004.

[36] B. Urgaonkar *et al*., Resource overbooking and application profiling in shared hosting platforms. In OSDI 2002

[37] V. Talwar et al., Modeling remote desktop systems in utility environments with application to QoS management, To Appear IM 2009.

[38] A. Allen. *Probability, Statistics, and Queueing Theory with Computer Science Applications*. Academic Press Professional, Inc. 1990.

[39] Qt Plug-Ins. http://www.qtsoftware.com/products/

[40] E. Guttman, E. Service Location Protocol: Automatic Discovery of IP Network Services. *IEEE Internet Computing* 3, 4 (Jul. 1999).