M-Channels and M-Brokers: Coordinated Management in Virtualized Systems

Sanjay Kumar, Vanish Talwar*, Partha Ranganathan*, Ripal Nathuji, Karsten Schwan
College of Computing, Georgia Institute of Technology Atlanta, GA 30332
{ksanjay, rnathuji, schwan}@cc.gatech.edu

*HP Labs, Palo Alto CA 94306
{vanish.talwar, partha.ranganathan}@hp.com

ABSTRACT

Management and automation are important issues in enterprise environments especially in virtualized enterprises, often consuming the largest fraction of the overall IT budget. A key challenge here is coordination across multiple management solutions deployed at different system layers, including across hardware and software, across different levels of abstraction, and across different hosts. This paper proposes novel abstractions that provide a uniform basis for management across different system levels and domains: (1) *M-channels* constitute the base mechanisms for communication between management embedded in hardware, in virtual machines, and in applications, and (2) *M-brokers* enable policy coordination across different management layers and solutions. The value of the approach is demonstrated with an implementation of coordinated power management policies that operate across the hardware-software boundary, for multiple virtual machines, and across different physical platforms. Coordination is shown useful for meeting application-level SLAs and/or to enforce power caps, with or without assistance from guest virtual machines. Further demonstrations of utility involve other management tasks, including storage backup, inventory, and trust management. Experimental evaluations of M-brokers and M-channels use the Xen hypervisor and are attained on server-class AMD platforms.

1. INTRODUCTION

Management (or manageability) in IT infrastructures refers to a range of operations required to maintain system resources through their life-cycle phases [9] - bring up, operation, failures/changes, and retirement/shutdown. Tasks performed at each of these lifecycle stages include provisioning and installation of servers, monitoring performance and health of systems, security and protection against attack, runtime resource management, backup, fault diagnostics and recovery, and asset management. With rising complexity and scale in today's enterprise IT, many of these management tasks have become non-trivial in computation, design, and in the number of execution steps performed. This coupled with the fact that management and automation can consume a large fraction (in some cases, 60-70%) of the total IT budgets for data centers today [3], has caused substantial growth in the development of management applications, and an adoption of automation software that reduces overall management costs [5, 4].

A key challenge for management solutions is coordination across different management domains, including across hardware and software [16, 18], across different levels of abstraction [8], and across multiple hosts [1]. Recent trends towards increased adoption of multicore systems and virtualization exacerbate this problem, in part because virtualization technologies create new levels of flexibility like dynamic VM migration and other runtime mappings between virtual and physical resources. Furthermore, VM resident management applications often lack the ability/privilege to access management hardware, and they also have the potential to conflict with each other because of the autonomous, uncoordinated nature of VMs. Platform/VMM resident management applications, on the other hand, have lack of visibility into the service level agreements (SLAs) and performance requirements of the VMs' applications.

Power management provides an interesting illustration of these problems. Power management exploits hardware support for dynamic voltage and frequency scaling (DVFS) [7], scheduling to increase device or component idle times [14], and a plethora of methods for improving the power behavior of individual subsystems, such as network devices [11]. Virtualization threatens the ability of these techniques to operate as intended, not only because of the VM's lack of privilege in accessing the DVFS hardware controls, but also because of the need for coordination. The latter is because the power management actions of one VM does not understand and therefore, cannot take into account the aggregate behavior of the set of VMs currently running on the same platform. In fact, even platform-level solutions cannot understand how their DVFS actions affect VM behavior and performance, unless VMs can share with such solutions their performance and SLA requirements and statistics. With VM migration, power management becomes more complex because the association of VM and platform can change dynamically and both have to be aware of this change and coordinate with their new associations and their potentially different policies.

Prior solutions addressing the virtualization layer, and management hardware have typically relied on ad-hoc approaches or point solutions and are limited in the coordination they provide. The HP PowerRegulator for example, implements power management in the firmware of the processor and has no feedback about how its power management decisions are affecting the SLAs of applications running on top of it. With the growing proliferation of management applications, a more general-purpose approach is needed.

This paper presents such an approach, borrowing from the rich prior work in the domains of adaptive and autonomic systems [19, 9], but using a light-weight implementation approach suited to the hypervisor and system levels it targets.

The management architecture developed and evaluated in this paper offers three technical contributions focused on the coordinated nature of infrastructure management:

- Two building-block abstractions key to coordinated management in virtualized systems are: (1) *M-brokers* and (2) *M-channels*. M-brokers defi ne policy managers with well-defi ned application-level protocols to perform information exchange, coordination, and actuation across different system layers. M-channels defi ne seamless communication between different M-brokers. Both abstractions provide generic interfaces to management applications in virtualized environments, thereby hiding underlying implementation and platform details. We instantiate these abstractions under Xen [2], and identify tradeoffs between different implementations of these management abstractions in this instantiation.
- M-brokers provide the framework to uniformly carry out management actions at different levels of the management hierarchy, such as in guest VMs, dedicated and privileged *management VM (MVM)*, platform hardware, enclosure, server rack and the data center. The management VM (MVM) is a new, per-platform layer in the management hierarchy, which runs M-brokers responsible for overall management of the platform, and coordinates with other MVMs in the data center.
- Coordinated management is shown effective by using it to implement and evaluate different solutions for online power management for server-class systems. Results demonstrate the flexibility and power of the M-channel and M-broker abstractions for improved management functionality at reduced application complexity.

Coordinated management with M-channels and M-brokers has multiple benefi ts. First, the management framework constructed in this fashion makes it easier to deploy layer specifi c policies using M-brokers and coordinate among them using M-channels. Second, coordination between M-brokers running at the VM, VMM, hardware and the data-center levels provides the flexibility to run policies which is much harder in existing solutions, as demonstrated with a software solution for power capping shown in this paper. Third, the M-channel and M-broker abstractions and their interface implementations can reduce the cost and time of developing management applications, because they offer functionality common to all such solutions. They also constitute a generic framework that can be applied to a wide range of management applications and can be used as a standard by management application vendors.

The remainder of the paper is organized as follows. Sections 2 and 3 describe the M-channel and M-broker abstractions as well as their implementation under Xen. In Section 4, these abstractions are used to implement several power management solutions that cross-cut management domains. Section 5 qualitatively discusses other applications of these abstractions. Section 6 discusses prior work, and Section 7 summarizes the paper.

2. COORDINATED MANAGEMENT IN VIRTUALIZED SYSTEMS

This section describes the M-channel and M-broker abstractions designed to support management architecture for enterprise IT infrastructures. We fi rst present the assumed system environment, and then describe details of the abstractions.

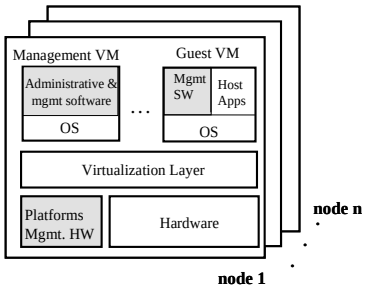

Figure 1: Target System Model

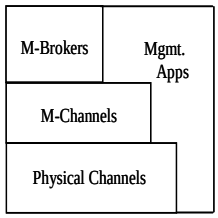

Figure 2: Coordination Stack with Management Abstractions

2.1 System Model

Figure 1 depicts the system model as a distributed environment wherein each of the nodes is a virtualized system. Hardware can have platform management controllers that perform hardware management. Each platform hosts a special privileged Management VM (MVM) that can run platform wide control and management software. Additional management capabilities may reside in guest VMs, at system or application levels. In today's systems, these individual management entities – those in the guest VMs, the MVM, and the platformsâĂŹ management hardware – operate within isolated silos, leading to reduced management functionality and potential in-effi ciency.

Fundamental to management architectures for system models like those depicted in Figure 1 are two building block abstractions: a communication abstraction, termed *M-channels*, and a coordination abstraction, termed *M-brokers* in Figure 2. As with event channels in distributed systems [19], M-channels implement bidirectional communication among participating entities, in our case including the hardware, guest and management virtual machines, and applications. M-brokers are the containers in which management and coordination policies are run, leveraging information provided via the uniform APIs presented by M-channels. As seen in Figure 2, our implementation of M-channels is built on top of existing physical channels, such as shared memory and network sockets, but lower level implementations like those required for communication across hardware components can exploit other facilities (e.g., the IPMI standard [6]). The remainder of this section presents details of the proposed abstractions.

2.2 M-Channels

The M(anagement)-channels communication channels used for transferring commands and associated information items between MVM and other VMs, as well as between MVM and management hardware. These channels can be instantiated on a single system or in a distributed setting. On a single platform, different M-channels are used for communication among VMs and MVM and for communication among MVM and hardware components. In a distributed system, M-channels are used for exchanging information among MVMs running on different platforms In both settings, M-channels can carry simple monitoring data, like system-provided information about current CPU usage (e.g., for utilization based power management), or they can be used for rich data and control exchanges between VMs and MVM (e.g., SLA related data about VMs' applications). M-channels continue to function under VM migration, by supporting dynamic disconnection and reconnection between VMs and MVM. We rely on underlying physical channels for the reliable, unreliable, or ordered delivery of messages. The event notifi cation model uses interrupts to notify the coordination

application.

2.3 M-Brokers

M-brokers execute coordination policies, since our management architecture assumes the presence of management policies embedded in hardware, MVMs, and guest VMs. A given system, therefore, has multiple M-brokers, different ones for different levels (VMs, MVMs, platform hardware etc.) and for different management applications (e.g., power vs. reliability management). Further, M-brokers may be distributed across guest VMs, MVM, and the hardware. Some M-brokers have more limited tasks, such as interacting with existing applications and hardware management applications and serving as proxies for sending management information to other brokers. Referred to as M-agents (or simply agents), these do not execute any coordination code, but are only responsible for monitoring, passing information, and receiving actuation commands. Examples provided in this paper concern M-agents used for power management, imparting information about application-level SLAs to power M-brokers resident in MVMs.

M-brokers and agents exchange messages via M-channels. Such exchanges belong to the phases of discovery, communication establishment, connection termination, and most importantly, runtime coordination. We have not yet implemented rich naming schemes and associated dynamic discovery mechanisms, currently relying on simple naming and discovery schemes. Addressing among brokers and agents uses identifi ers unique for both hardware and software components and across a distributed system.

As shown in Figure 2, an M-broker uses the M-channel APIs to implement its coordination modules. Core coordination modules implementing basic protocols for communication among brokers and agents are reused across different coordination brokers. Application-specifi c extensions can be provided over this basic framework.

2.4 Benefits

Management architectures and applications built with M-channels and M-brokers has several advantages compared to using adhoc or point solutions. The generic nature of the architecture permits it to be used for multiple management applications. This helps reduce the costs of developing and administering these applications and may lead to standardization efforts across different platforms and systems. Further, by abstracting common parts of management applications into well-defi ned communication and coordination abstractions, application developers can focus on the management- and coordination-specifi c logic required by applications, infrastructures, and execution environments.

An important functionality provided by M-channels is the dynamic association of M-brokers. For instance, when a VM is migrated, the M-broker inside the VM must be dissociated from the MVM on the original platform and associated with the new platform's MVM. M-channels are implemented to support dynamic and endpoint-transparent reconnection between brokers, including those residing in VMs and MVMs. A specifi c use of that functionality explained in Section 4 is the use of VM migration to implement effi cient method for power capping in data center environments. M-channels provide standard APIs found in a typical middleware system like operations on channels, e.g., open, close, read, write etc. In addition, the M-channels also transparently handle the virtualization layer (apart from hiding hardware details) and VM migration and expose the same set of APIs irrespective of underlying physical channel. M-channels also provide primitives for accessing management hardware and associating VMs with the host platform. The code fragment below shows the M-channel's internal support for VM migration. It registers suspend_vm and resume_vm callback functions with the underlying VMM, which during VM migration, get called when the VM is suspended on the old machine and resumed on the new machine, respectively thereby handling VM migration transparently.

```
suspend_vm() {
  flush_message_queue()
  save_channel_state()
  close_connection()
  wait_for_VM_resume()
}
resume_VM () {
  discover_the_new_MVM()
  establish_connection()
  restore_channel_state()
  resume_normal_operation()
}
```

Uniform APIs and channel-based interactions between guest VMs, MVMs, hypervisor, and the hardware platform provide management applications with a *holistic view* of the system, enabling coordinated management actions superior to those taken in isolated silos. By hiding the details of hardware management units, M-brokers enable developers to focus on management functionality and actions instead of the nuances of certain hardware components. Finally, the design of the framework permits the reuse of coordination modules across different management applications and is easily extended with additional functionality.

3. XEN IMPLEMENTATION

To experiment with and evaluate these abstractions, they have been implemented in Xen as a representative VMM. Figure 3 shows the Xen-specifi c implementation where multiple hosts are communicating over M-channels and where one of the host's virtualized structure is shown in detail. The MVM is instantiated in Xen's ""Dom0", and the guest VMs are referred to as "DomUs". The intra-machine M-channels are implemented using Xen-provided inter-domain shared memory communication. The M-broker and M-agent interfaces to M-channels support both user- and kernel-based implementations. Each of these components is described in more detail below.

3.1 M-Channel Implementation

Xen realization of M-channels uses multiple implementation methods, described in more detail below and used for power management in Section 4. For intra-platform communications and to meet the need for high rate, low overhead communications (e.g., for reliability monitoring), M-channels are mapped onto Xen's VMM-provided inter-domain communication channels, Xenbus. This in turn uses shared memory regions mapped by the VMM to the communicating VMs. More specifi cally, a management frontend (mgmtfront) driver module runs inside the guest VM. This module communicates with the management backend (mgmtback) inside the MVM. Mgmtfront and mgmtback represent the M-channel endpoints for the guest VM and MVM, respectively. Communication is asynchronous and hence, uses two different communication rings for the two directions (send and receive). Both mgmtfront and mgmtback export a fi le interface (/dev/mgmt) to user-level brokers and an API interface to brokers running at kernel level. If the sent or received data size is more than the ring element size, the data is passed by sharing the page containing the data and passing pointers to it.

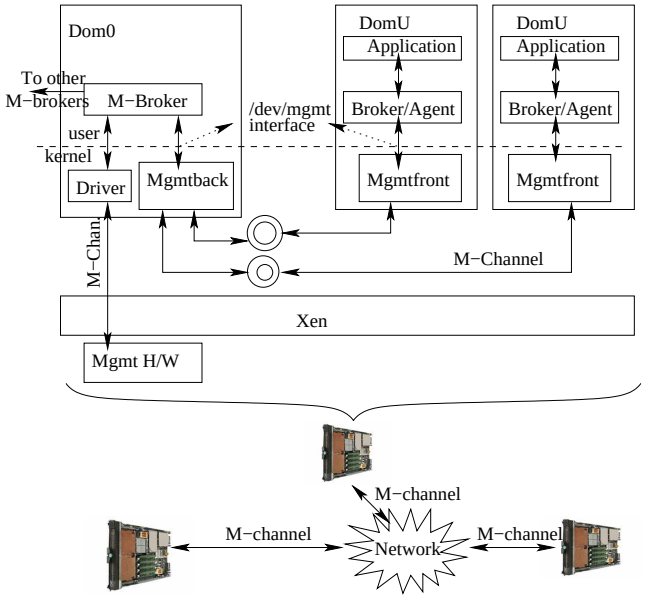

Figure 3: M-Channel and M-Broker Implementation in Xen. Intra-platform M-channelsare implemented as shared memory based while inter-platform channels are network sockets based

The second implementation of M-channels uses network sockets for cross-machine communications, providing the same APIs as the shared memory implementation. This permits VMs to transparently communicate and coordinate across multiple machines, but offers higher latencies and overheads than shared memory channels. In both implementations the management-specific agents and brokers define their own message formats for M-channels which provides flexibility to the applications.

The M-channel to the management hardware is implemented as a device driver which communicates with the hardware provided communication interfaces (e.g. PCI interface in case of the management processor). This driver also exports a file interface and API interface (similar to shared memory M-channels) and provides the same basic interfaces as the VM-VM M-channels.

An important attribute of the M-channel implementation is that brokers and agents can continue communicating during VM migrations. During migration, the mgmtfront and mgmtback modules are notified of the VM migration event. This triggers a new set of disconnections and reconnections (using suspend and resume callbacks described in Section 2.4) i.e., the older mgmtback breaks its connection with the mgmtfront and new mgmtback establishes a new connection with the mgmtfront. The outcome is that the agents and brokers inside guest VMs need not be cognizant of migration actions, even when the M-brokers with which they interact are changed (e.g., M-broker in the new platform's MVM).

3.2 M-Broker Implementation

M-brokers and agents are implemented as multi-threaded applications, with core threads responsible for coordination and communication and others executing actual management application-specific broker code. They can run at kernel or user levels, in both cases seeing the same M-channel APIs. To access the management hardware, the MVM's M-broker utilizes the "driver"-provided M-channel interface (similar to /dev/mgmt) to read from and write to it. In the current implementation, the MVM's M-Broker also works as the bridge between the VMs and the management hardware, i.e., all accesses to the management hardware from VMs must go through the M-broker in the MVM.

Security (i.e., authentication and authorization) is not implemented in the current M-Broker and M-channel interfaces and is on-going work. Because of the M-channel handling of VM migration, the M-brokers and agents remain oblivious to such events, and local M-channels get re-established after migration.

4. BENEFITS FOR POWER MANAGEMENT

We construct two different M-brokers in our experiments to evaluate the benefits of SLA-aware power management. The purpose of these implementations is to show that M-channel and M-broker abstractions can be used to construct different power management architectures and policies, ranging from traditional utilization-based management to novel SLA-based management and in coordination with other VMs and platform requirements. Power management is based on the active management of power via VM migration for consolidation and via dynamic voltage and frequency scaling (DVFS) of the CPUs. In the next two sections, we detail the particular implementations of these brokers, completing with a review of the evaluation summary.

4.1 M-brokers for Power Management

4.1.1 M-Broker1

The first power management M-broker instantiated on this architecture runs inside the MVM, and agents run in guest VMs. The M-broker establishes M-channels with the VM agents. The VMs run an application with some specified service level agreements (SLAs) which is monitored by agents. The individual agents use the M-channel to notify the broker of SLA violations. For power management, the M-broker runs administrator-specified policies, which set the power state (p-state) of the physical CPUs by using the SLA violations as inputs. The application-specific message formats allow the power management algorithms to have the flexibility of implementing arbitrary SLA models (e.g., hard and soft SLA violations, different metrics for SLA violations, etc.) The M-broker, then, uses inputs from the M-channels to drive decision algorithms that set the power states of CPUs.

Our testbed consists of dual-core, dual-socket (a total of 4 cores) AMD 64-bit Opteron based HP C-class blades. These processors provide hardware frequencies of 2.4 GHz, 2.2 GHz, 2.0 GHz, 1.8 GHz, and 1.0 GHz. This testbed is used to experiment with SLA-based power management using this broker. Guest VMs are non-SMP and run the HTTP server workload, and httperf is used to generate load for this server. The VM agent monitors the server side response time of individual requests as the SLA parameter.

SLA Model: with this evaluation setup, three sample power management policies are evaluated.

(1) Basic policy: the agent records the number of SLA violations within epochs (500 ms each), and at the end of each epoch, if the percentage of violations exceeds a threshold (1%), a SLA violation notification is sent to the power broker. Upon receiving this notification, the broker increases the p-state of the VM's vcpu to the next higher value. A parallel thread also monitors the VM's cpu utilization and whenever the utilization goes below a certain threshold (80%), the VM's vcpu is reduced to the next lower p-state. For a particular physical CPU, all of the vcpus (currently mapped onto that physical CPU) p-states are examined, and the highest p-state is set as the p-state of the physical CPU.

(2) Richer policy: a more sophisticated notification criteria is one in which the SLAs are defined as *hard* (higher response time threshold) or *soft* (lower response time threshold). With each notification,

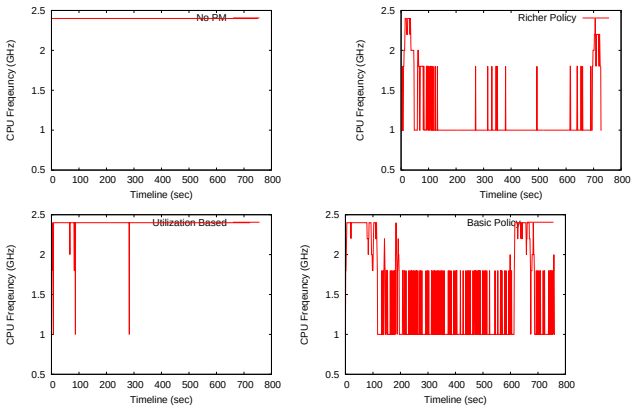

Figure 4: CPU frequency traces for various policies

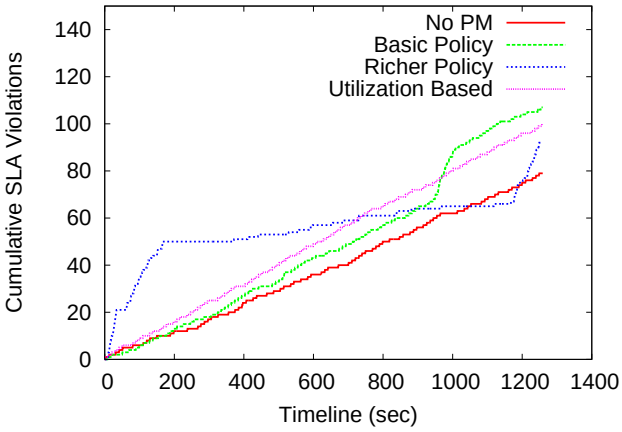

Figure 5: Cumulative SLA violations for various policies

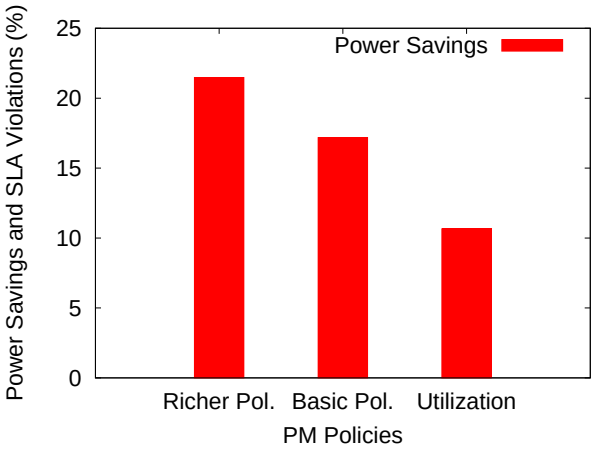

Figure 6: Power Savings

then, the VM agent sends information whether a hard SLA violation or a soft SLA violation was experienced. The power broker can, then, employ more sophisticated management policies. In our example, on hard violations, the policy reacts the same as the *Basic Policy*. For soft violations, however, the policy records these violations for 10 consecutive epochs and if the soft violations happen more than 30% of the time, it is treated as a hard SLA violation and action is taken according to the *Basic Policy*, else soft violations are ignored. Here, the richness of M-channel interface allows the agent to define various degrees of sophistication in notification to the power broker which then allows the broker to implement more sophisticated policies.

(3) Utilization-based: guest VMs with built-in power management policies are used in the third set of experiments, based on the Linux VM's *ondemand* power governor. This governor decides the power state of the vcpu based on its current vcpu utilization, but to 'translate' such settings to actual p-state changes, the power broker receives this information through the M-channel. It then uses it as input to *Basic Policy* to decide the actual p-state of the CPU.

We compare the three policies against the "no power management (No PM)" case for a varying httperf workload. These results are explained in more detail next, but we note that the most important insight derived from them for this paper is the relative ease with which these different power management solutions and architectures could be built, thereby demonstrating the flexibility and richness of M-broker and M-channel abstractions and their ability to implement SLA-aware power management. Each of these policies were implemented with approximately only 300 lines of code.

Results: Figure 4 shows the trace of the frequency of the CPU running the http server VM as the experiment progresses. We note that, the Utilization-based policy runs the CPU at the highest speed for most of the time because it sees the virtual CPU utilization to be high most of the time. Although this leads to less power savings as shown in Figure 6, it is still able to save about 10% power by putting the three other less utilized cores into lower power states. The Basic policy reacts much better to SLA violations and changing domain utilization compared to the Utilization-based policy, because it coordinates with the MVM using the MVM's view of CPU utilization. This underscores the fact that VM-based policies are often inefficient in system monitoring because of the virtualization layer. The MVM in turn uses SLA violations as input for making p-state decisions. It saves power by operating mostly at the lower CPU speeds while still trying to maintain SLAs. The Richer policy is even more aggressive in saving power because of the more flexible SLA model defined for it (hard and soft SLAs with threshold values of 6ms and 3 ms, respectively). This leads to a reduced number of total hard violations over intervals, which allows the policy to run the CPU predominantly at the lowest speed. Figure 6 shows that this leads to the Richer policy providing the most power savings (22%). Figure 5 shows the cumulative number of SLA violations for different policies. We see that the 'No PM' policy expectedly performs the best. However, the Basic, Richer, and Utilization-based policies show similar characteristics with the Utilization-based policy only slightly outperforming the Basic one. The results also demonstrate that for policies focused on maintaining application SLAs, SLA-based power management proves to be more efficient at power savings than traditional utilization-based policies.

4.1.2 M-Broker2

To demonstrate the hardware/software coordination between the VMs and platform's power limitations, we implement a power capping M-broker which takes platform's power limits (both average and peak power) and enforces the limits. We use HP's C-class blades, with their built-in management processor called iLO (integrated lights out). This processor continually records the current power consumption of the platform. The M-broker maintains a M-channel with the agent running on iLO management processor to obtain current power consumption, and it also establishes

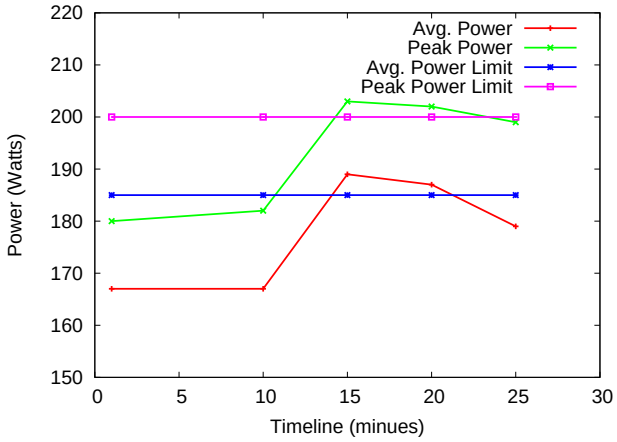

Figure 7: Power Capping

additional M-channels to coordinate with M-brokers running on other machines' MVMs in the data center. This co-ordination also helps in implementing data-center wide power capping policies by making sure the overall power consumption of the data-center stays within limits besides individual nodes power consumption.

SLA Model: the M-broker periodically (every 5 minutes) queries the management processor for current average and peak power consumption. If these limits are violated (e.g., because of an increase in VM load), the M-broker queries other MVMs in the data center to find a less heavily loaded machine (with power consumptions well within limits) and migrates the VM to that machine to bring the power consumption within limits. VM migration is transparent from the M-broker's point of view because the M-channel internally handles migration, as described in Section 3. **Results:** Figure 7 shows the results of this experiment. A machine is running 4 VMs containing a WebServer which gets heavily loaded by httperf clients. The average and peak power limits are defined as 185 and 200 watts, respectively. We see that after 10 minutes, as the load increases in all VMs, the power limits (both average and peak) are violated. In response, the broker finds a suitable machine and migrates the most heavily loaded VM to that machine. This reduces power consumption, but it is still above the set limit. In response (at 20 minutes), the broker migrates another VM, which finally brings down the power consumption within desired limits. The experiment demonstrates that the M-channel between the broker and the iLO processor enables the system to directly react to power limit violations, which otherwise is not possible or will have to rely on indirect metrics (e.g., cpu utilization) as a proxy for power consumption. This example also clearly shows the importance of coordination between management policies and platform hardware to ensure efficient data center operation.

4.1.3 Summary

The ability to easily implement various power management architectures and policies underscores the flexibility of the M-channel and M-broker abstractions and therefore, their appropriateness for providing general management functionality. Experiments underscore the importance of coordinated management in virtualized environments, both across VMs and across hardware and software. Another key insight shown by these experiments is that SLA-aware power management enabled by our architecture provides good power savings while maintaining VMs' SLA levels. Section 5 qualitatively describes how other management applications can be implemented using this architecture, thereby further demonstrating its generality.

5. BENEFITS FOR OTHER USE CASES

5.1 Storage Backup

Storage backup can be improved in many ways by using this management architecture. Backup in virtualized environments can be done either from inside the VM (just like in non-virtualized environments) or from outside the VM (e.g., inside the MVM or as in VMWare's Consolidated Backup [22]. The former solution can potentially lead to performance interference among VMs sharing the disk, because of the lack of coordination about when backups are performed. The performance interference can be reduced by having VMs' backup agents coordinate with the MVM to avoid backing up data simultaneously on shared disks. If backup is done from outside VMs, e.g., from the MVM, the VM backup agents can also notify the MVM about when the VM's disk activity is low, whereupon the MVM can perform the VM's backup at times of low activity so as to minimally affect the VM's performance because of backup. Further, since storage virtualization involves all disk accesses from the VMs being handled by the VMM (or in many cases, the privileged disk driver VM), this provides us with the opportunity of externally tracking VM disk activity. Hence, the MVM can coordinate with the VMM or the disk driver VM, using an M-channel to keep track of all of the disk blocks a VM modifies. This information can then be used to create a disk replica by writing modified blocks to the replicated disk in parallel (essentially implementing a software based RAID-1) and using the replica for backup, incurring zero freeze time. This information can also be used to perform efficient incremental backup since a list of modified blocks is maintained. Finally, an M-channel to the disk drive can be used to access the S.M.A.R.T. data from the MVM. This can then be used by the MVM's backup agent or passed to the VM's backup agent to estimate disk health and predict impending failures and thus perform proactive backup. In such cases, to minimize possible data loss, the backup system can prioritize blocks that are known to be modified.

5.2 Inventory Management

Inventory management can be performed by using M-channels for information exchange among inventory M-brokers at various system layers, e.g., at the hardware, software, and virtualization layers. These brokers perform coordination across hardware and software, as well as across VMs. For example, if a correlation is desired between the software and hardware inventory, the M-brokers at the guest VM and firmware unify such information. If a correlation is desired for all of the information at the hardware, software, and virtualization layers, the M-broker at the MVM can perform the unification. The M-broker at the MVM also provides a single client interface to simplify administrator access to inventory information. In such a design, the M-broker at the MVM hosts an inventory registry broker to which all of the inventory agents at the guest VMs, firmware, and MVM register themselves. On receiving a request, the inventory registry broker acts as a proxy and re-directs the request to one or more of the agents at the guest VMs, at the MVM, and the one hosted in the firmware (or management processor). The communication takes place using M-channels. On receiving the results from the inventory agents, the M-broker may do correlation and unification of data as needed before sending it to the client.

This enables the creation of a unified picture with inventory information collected from the hardware, VMM, and guest VMs, even in the absence of any common attributes across the individual

inventory tables. Further, leveraging a single access point at the MVM aids in reduced administrator cost. This holds even greater significance with advances in virtualization wherein several VMs would be hosted per host, each of which contains a separate inventory agent. Even if we assume 10 access points per server/blade, with a single plain point interface provided by our solution, an administrator reduces his number of steps by 90% – a large complexity and cost savings.

5.3 Trust Management

Trust management concerns with the trust placed in a platform to faithfully and securely carry out certain application tasks. The problem, of course, is that these trust levels vary over time, depending on current platform properties (e.g., temperature levels or observed recent failure rates) and depending on static or dynamic VM properties (e.g., OS configuration with/without certain virus checkers or recently observed OS behaviors). Hardware/software management, then, can (1) observe components and VMs, (2) use observations as inputs to dynamic trust models, and (3) run policies that continually monitor total platform trust and/or ensure certain minimum levels of trust to be present for running applications. Here, trust is built incrementally by the VMM using the Trusted Platform Module (TPM) for basic platform trust, then using this trust to certify trust in MVM, (termed 'trust controllers' in the literature [10]), then having MVM perform monitoring, execute trust models, and finally, trigger actuators to maintain the levels of trust desired by applications or data center administrators.

Here, VMs' agents can assist in VM monitoring, using M-channels to provide trust data to MVM, but MVM must have its own, additional monitoring methods to deal with misbehaving VMs. All policies are run in MVMs, with each single MVM responsible for its platform and platform-resident VMs, and multiple MVMs cooperating to establish the end-to-end notions of trust required by applications. For instance, for a multi-tier application, trust is not a meaningful concept unless it can be applied across all tiers used by requests, thus requiring coordination across all MVMs involved in tier execution.

6. RELATED WORK

Modern data centers require the simultaneous deployment of multiple management solutions. Examples include hardware based management solutions such as temperature control, cooling [15] etc. Multiple vendors have built-in support for management capabilities in their systems, e.g., iLO from HP, iAMT from Intel, ALOM from SUN, SP from IBM, etc. These solutions, however, are not seamlessly integrated into the host OS's management applications. Additionally, multiple management standards have been defined to manage resources in distributed environments, such as SNMP, WBEM, CIM etc. Various vendors use these standards to provide management solutions to enterprises [4, 5]. These are, however, mainly software based solutions and usually do not coordinate with the platform hardware on which they run. With virtualization technologies like Xen and VMWare [2, 21] entering the data center, the manageability and coordination problem becomes even more complex e.g., SNMP, WBEM, and CIM do not account for virtualization layer or its side affects (e.g., VM migration) leading to inefficient management.

Various methods have been proposed to perform power management of computing platforms. For example, DVFS (Dynamic Voltage and Frequency Scaling) mechanism can be extended using hardware solutions [13] or OS-level techniques that set processor states based upon predicted application behavior [7]. Other methods utilize the notion of performance slack for real-time workloads to aggressively reduce frequencies while meeting deadlines [17]. Power budgeting solutions for single platforms have also utilized processor control to provide fine grain power capping capabilities [12]. Solutions for power budgeting across multiple systems have considered intelligently enforcing power budgets at individual machines and blade enclosure granularities [18]. Providing system support for extending such management techniques to virtualized environments with existing virtual machine interfaces has also been studied [16, 20].

The point solutions listed above solving individual management problems, however, make management very complex when deployed in a large virtualized data-center. The generic nature of our management architecture significantly reduces this complexity while providing better support for developing management applications.

7. CONCLUSIONS AND FUTURE WORK

This paper presents two powerful abstractions, termed M-channel and M-broker, for coordinated management in virtualized execution environments. These abstractions provide generic interfaces for implementing diverse management policies, resident in VMs, in management VMs, and in hardware. We have shown the usefulness of these abstractions by implementing them in the Xen environment and using them to realize coordinated power management policies. We also present a qualitative analysis of additional management applications, including storage backup, inventory management, and trust management, to further demonstrate the utility of these abstractions and the generality of our approach.

This remains an active work in progress, with ongoing work to implement additional management applications such as reliability and trust using these abstractions. Further, we are working on an integrated management solution where VMs and host platforms can be managed by managing a variety of metrics, e.g., SLA, power, reliability, trust etc. in coordination with each other, compared to existing solutions where these metrics are managed in isolation from each other. The current MVM implementation is also being generalized to become a virtual appliance, suitable for dynamic deployment and upgrades in realistic enterprise systems.

8. REFERENCES

[1] S. Agarwala, F. Alegre, , K. Schwan, and J. Mehalingham. E2eprof: Automated end-to-end performance management for enterprise systems. In *DSN*, June 2007.

[2] P. Barham et al. Xen and the art of virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[3] Gartner TCO Reports. 2005, 2006, 2007. `http://www.gartner.com/`.

[4] HP Systems Insight Manager. `http://h18002.www1.hp.com/products/servers/management/hpsim/index.html`.

[5] IBM Tivoli Software. `http://www-306.ibm.com/software/tivoli/`.

[6] Intelligent Platform Management Interface. `http://www.intel.com/design/servers/ipmi/index.htm`.

[7] C. Isci, G. Contreras, and M. Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *MICRO-39*, December 2006.

[8] J. Kephart et al. Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs. In *ICAC*, June 2007.

[9] J. O. Kephart. Research challenges of autonomic computing. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, 2005.

[10] J. Kong, I. Ganev, K. Schwan, and P. Widener. Cameracast: Flexible access to remote video sensors. In *Proceedings of the ACM Multimedia Computing and Networking Conference (MMCN)*, 2007.

[11] R. Kravets and P. Krishnan. Application-driven power management for mobile communication. In *MOBICOM*, pages 263–277, October 1998.

[12] C. Lefurgy, X. Wang, and M. Ware. Server-level power control. In *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC)*, June 2007.

[13] H. Li, C. Cher, T. Vijaykumar, and K. Roy. Vsv: L2-miss-driven variable supply-voltage scaling for low power. In *MICRO-36*, 2003.

[14] Y. Lu, L. Benini, and G. Micheli. Low-power task scheduling for multiple devices. In *Proceedings of the 8th International Workshop on Hardware/Software Codesign*, pages 39–43, 2000.

[15] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making scheduling "cool": temperature-aware workload placement in data centers. In *Proceedings of the USENIX Annual Technical Conference*, Berkeley, CA, USA, 2005.

[16] R. Nathuji and K. Schwan. Virtualpower: Coordinated power management in virtualized enterprise systems. In *SOSP*, Oct. 2007.

[17] P. Pillai and K. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *SOSP*, October 2001.

[18] P. Ranganathan, P. Leech, D. Irwin, and J. Chase. Ensemble-level power management for dense blade servers. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2006.

[19] R. E. Schantz, J. P. Loyall, C. Rodrigues, and D. C. Schmidt. Controlling quality-of-service in distributed real-time and embedded systems via adaptive middleware. *Softw. Pract. Exper.*, 36(11‐12):1189–1208, 2006.

[20] J. Stoess, C. Lang, and F. Bellosa. Energy management for hypervisor-based virtual machines. In *Proceedings of the USENIX Annual Technical Conference*, June 2007.

[21] J. Sugerman, G. Venkitachalam, and B.-H. Lim. Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor. In *Proceedings of the USENIX Annual Technical Conference*, 2001.

[22] VMware Consolidated Backup. `http://www.vmware.com/products/vi/consolidated\_backup.html`.