M-Channels and M-Brokers: New Abstractions for Co-ordinated Management in Virtualized Systems

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

Vanish Talwar, Partha Ranganathan
HP Labs
Palo Alto, CA 94306
{vanish.talwar, partha.ranganathan}@hp.com

ABSTRACT

Management and automation are important issues in enterprise environments, often consuming the largest fraction of the overall IT budget. A key challenge here is co-ordination across multiple management solutions deployed in different management domains, including across hardware and software, across different levels of abstraction, and across different hosts. This paper makes three contributions to addressing this problem. First, we propose a novel management co-ordination architecture for virtualized environments. Our architecture includes two powerful abstractions – *m-channels*, which provide mechanisms for communication between the hardware, virtual machines, and applications and *m-brokers*, which allow high-level policy co-ordination across different management agents. Second, we discuss *Dom-M*, an instantiation of our architecture in the context of the Xen hypervisor, and identify tradeoffs between different implementations of such management domains. Finally, to demonstrate the effectiveness of our approach, we implement and evaluate different cross-level solutions for power management using our abstractions, and also discuss qualitatively other applications including storage backup, inventory management, and trust management.

1. INTRODUCTION

Rising complexity in modern data centers has led to increased costs for managing data center resources and applications. Indeed, various studies have shown that management and automation often consume the largest fraction (in some cases, 60-70%) of the total IT budgets for data centers today.

In response to these trends, a spectrum of management solutions have been deployed in the data center, including hardware monitoring, power and thermal management, diagnostics, provisioning, SLA management, storage backup, etc. A key challenge, however, is co-ordination across different management domains, including across hardware and software, across different levels of abstraction, and across different hosts.

Recent trends towards increased adoption of multicores and virtual machines exacerbate the co-ordination problem. For example, management applications running inside virtual machines (VMs) often lack the ability/privilege to access management hardware, because of the virtualization abstractions. Similarly, management agents in different VMs in a virtualized environment have the potential to conflict with each other because of the autonomous, uncoordinated nature of policies. Lack of visibility into the SLA and performance requirements of the guest VM applications at other layers is another cause.

Power management provides an interesting illustration of these problems. Power management agents in the guest VMs (e.g., the Linux P-state scheduler [17]) react to local resource usage and make changes to actuators corresponding to virtualized hardware. The virtualization layer, may in turn, do its own power management across a collection of VMs, for example, by migrating virtual machines to reduce power. At the physical platform layer, hardware controllers may manage the power based on aggregate information visible at that layer. Similar scenarios can be identified for other management applications, as discussed in Section 6 of this paper.

Prior solutions addressing the isolation across guest applications and operating systems, virtualization layer, and management hardware have typically relied on ad-hoc approaches. They include proprietary approaches to interface across different management domains and system-specific implementations and are limited in the co-ordination they provide. With the growing proliferation of management applications, a more general-purpose approach is needed. This paper addresses this problem of management co-ordination, with the following three contributions:

- We present two building-block abstractions that are key to co-ordinated management in virtualized systems. *M-channels* define seamless communication between layers with well-defined and restricted semantics and APIs to abstract from underlying implementation and platform details. *M-brokers* define policy managers with well-defined application-level protocols to perform information exchange, co-ordination, and actuation across layers. We instantiate these abstractions under Xen, and identify tradeoffs between different implementations of these management abstractions in this instantiation.
- We discuss *Dom-M*, a dedicated and privileged management VM (MVM) which runs M-brokers responsible for overall management of the platform, and coordinates with other DomMs in the data center for the overall data center management. We also discuss the advantages and tradeoffs of implementing Dom-M as a separate MVM in the management infrastructure.
- Finally, to demonstrate the effectiveness of our approach, we implement and evaluate different solutions for power management using our proposed abstractions. Results show the flexibility and power of our m-channel and m-broker abstractions for improved management functionality at reduced application complexity.

The remainder of the paper is organized as follows. Section 2 provides additional background. Sections 3 and 4 discuss our proposed M-channel and M-broker abstractions and the implementation tradeoffs under Xen. Section 5 discusses the use of M-channels and M-brokers to implement several power management solutions that cross-cut management domains. Section 7 qualitatively discusses other applications of our abstractions. Section 8 discusses prior work and Section 9 summarizes the paper.

2. PROBLEM BACKGROUND

2.1 Existing Management Solutions

Management (or Manageability) refers to the range of operations required to maintain system resources through their lifecycle phases – bring up, operation, failures/changes, and retire/shutdown. Tasks performed at each of these lifecycle stages include provisioning and installation of servers; monitoring performance and health of systems; security protection against viruses and spyware; runtime performance and resource management; backup protection against disasters; disk maintenance to improve performance; fault diagnostics and recovery; and asset management to track resources. With rising complexity and scale in today's enterprise IT, several of these management tasks have become non-trivial in computation, design, and in the number of execution steps performed. Further, although virtualization technology helps in managing data centers by providing isolation, consolidation, VM mobility, and VM appliances, it also creates a new set of challenges to manageability because of virtualization hazards like VM migration, dynamic mapping between virtual and physical resources etc. As a result, we are seeing substantial growth in the development of manageability applications, and an adoption of automation software that reduces overall maintenance costs.

Contemporary manageability solutions can be classified as hardware-based and software-based solutions. Examples of hardware-based solutions are those related to hardware monitoring, power management, thermal management, and diagnostics. These solutions are deployed in firmware either at the management processor or other hardware peripherals such as BIOS, smart disks, or sensors. Examples of software-based solutions are those related to provisioning, SLA management, and file-based backup. These solutions are deployed at the host system, typically as user space programs.

2.2 Co-ordination Problems and Challenges

While individual hardware- and software-based management solutions have proved useful, their use in virtualized environments provides new challenges for three main reasons. (1) Since VMs are not the privileged entities on a platform, they typically lack the ability to access management hardware, since that requires certain privileges. This prevents them from carrying out VM- or application-specific management actions. (2) Even with privileged access, since VMs are highly 'autonomous' and usually run their own management policies, a coordination problem arises among multiple VMs, since they may try to manage the underlying system with potentially different policies. (3) Virtualization creates virtual resources, and does VM migration, preventing VM-level management from taking care of tasks like inventory management or any per platform management actions, thereby further worsening the problem of hardware-software co-ordination. We illustrate these problems through specific examples below.

Power management exploits hardware support for dynamic voltage and frequency scaling (DVFS) [8], scheduling to increase device or component idle times [13, 16], and a plethora of methods for improving the power behavior of individual subsystems, such as network devices [10, 21]. Thermal management further deepens this work, including with load balancing methods to idle selected system components [14]. Virtualization threatens the ability of these techniques to operate as intended, not only because of lack of access to DVFS hardware controls, but also because the management actions of any single VM do not understand and therefore, cannot take into account the aggregate behavior of the set of VMs currently running on the same platform. Further, they cannot consider differences among VMs, such as different degrees of memory boundedness, I/O utilization, etc. In fact, even platform-level solutions cannot understand how the choices they make for different VMs affect VM behavior and performance, unless VMs can share with such solutions their SLA requirements and in addition, provide feedback concerning the effects of certain actuation decisions on these requirements.

Storage backup is another important management activity in enterprise data centers, to improve fault tolerance and recovery. Backup in virtualized environments can be done either from inside the VM (just like in non-virtualized environments) or from outside the VM, such as VMWare's Consolidated Backup. The latter has the problem that it is hard to find out which files have been modified by the VM and hence which files to backup. The former can potentially lead to performance interference with other VMs sharing the disk, and it has the associated property that it cannot take disk health into consideration. With many modern drives providing disk health related SMART data, such information can be used for proactive backup and hence reduce data losses and recovery time. Similar issues and opportunities arise for fault diagnosis. A *fault diagnosis* method running in a single VM will only have a partial view of underlying hardware behavior, and a method running at platform level may not have sufficient information about the system or applications actions that triggered faults.

Inventory management relates to the capture of assets - both hardware and software – in the data center, to provide for a view of the actual state of components comprising the infrastructure. Specific examples of inventory information include those relating to processor and peripherals configuration, applications hosted, system software and middleware stack, as well as monitoring sensors installed at the servers. With emerging trends towards virtualization, consolidation, and hardware advancements, the amount of inventory information relating to the software and hardware per server in modern data centers is growing exponentially. Existing designs, however, collect such growing inventory information through multiple agent end-points. For example, hardware inventory information is provided through agents typically hosted in firmware at the management processor; inventory information related to the hypervisor is provided through agents in the Control Domain (e.g., Dom0 in Xen); and agents in each guest VM provide software and application inventory information. The inventory information obtained from these individual agent end-points is uncorrelated. This is a hindrance towards a unified management in the data center. Additionally, having multiple non-coordinated agents leads to increased administrator workload which leads to higher overall costs in the data center.

In all of these examples, hardware/software coordination and coordination among VMs is either non-existent or uses some ad-hoc mechanisms leading to overall inefficiency and reduced functionality of management applications.

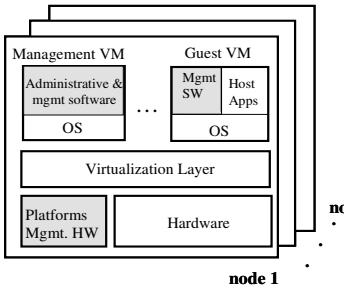

Figure 1: Considered System Model

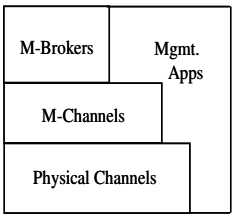

Figure 2: Coordination stack with proposed abstractions

3. CO-ORDINATED MANAGEMENT IN VIRTUALIZED SYSTEMS

The discussion in the previous section motivates the need for general abstractions to enable coordinated management in the data center. In this section, we propose and discuss some such abstractions. We first present the assumed system environment, and then describe details of the abstractions.

3.1 System Model

Figure 1 depicts the system model as a distributed environment

wherein each of the nodes is a virtualized system. The system hardware is assumed to have platform management controllers that perform hardware management. Further, we assume the existence of a special privileged management VM (MVM) that hosts system wide control and management software.

In today's systems, these individual management entities – those at the guest VMs, MVM, and the platforms management hardware – operate within separate isolated silos non-coordinated among each other. As was explained in Section 2, this leads to reduced management functionality and potential in-efficiency. This paper seeks to determine the key abstractions that we need to support for effective co-ordination among the different management entities in virtualized systems. Specifically we propose two building block abstractions we believe to be fundamental for co-ordinated management: *M-channels* and *M-brokers*. M-channels provide bi-directional information exchange among the hardware, virtual machines, and applications. M-brokers provide a framework for implementing co-ordination policies, leveraging information provided through M-channels. Figure 2 shows the deployment stack with the proposed abstractions. As seen in the figure, M-channels are an abstraction built on top of existing physical channels, such as shared memory and sockets. M-brokers use these M-channels for communication and execute co-ordination policies. They interact closely with existing management applications and enhance their functionality. M-channel and M-brokers do not prescribe how management tasks are carried out, permitting applications to either use one M-broker servicing multiple platforms or distributed implementations in which multiple broker instantiations cooperate. Similarly, M-channels can simply carry simple monitoring data, like system-provided information about current CPU usage, or they could be used for rich data and control exchanges between VMs and management VM. The M-agents explained below play an important role in determining M-channel usage. The remainder of this section presents details of the proposed abstractions.

3.2 M-Channels

The M-channels or management channels are special inter-VM and hardware-software communication channels which transfer commands and information between MVM and other VMs, as well as between MVM and management hardware. These channels can either be instantiated at a single system or in a distributed system context, as seen in Figure 3. At a single system, different M-channels are used for communication among VMs and communication among hardware components, with the help of a M-channel communication bridge instantiated at the MVMs. The bridge re-directs the messages from one channel to the other according to some set of policies. This re-direction is transparent to the sending and receiving management entities. At a distributed system, the M-channels are used for exchanging information among the management virtual machines (MVMs), as shown in Figure 3.

M-channels are bi-directional in nature and can be used for exchange of both control information and actual data. To continue to function under VM migration, M-channels support dynamic disconnection and reconnection between VMs and management VM, and for flexibility, they deliver messages using both point-to-point and broadcast semantics. Both of these semantics have uses for the co-ordination applications we are considering. Similarly, delivery of messages can be synchronous or asynchronous depending on the co-ordination application being supported. Our implementation currently uses asynchronous messaging which we found to be useful when streaming monitoring data on a continuous basis. We rely on underlying physical channels for the reliable, unreliable, or ordered delivery of messages, and wherever the physical channel supports these options, we provide support for the same. The event notification model uses interrupts to notify the co-ordination application.

We also note that the M-channels among hardware components can be built on top of the IPMI standard [7].

3.3 M-Brokers

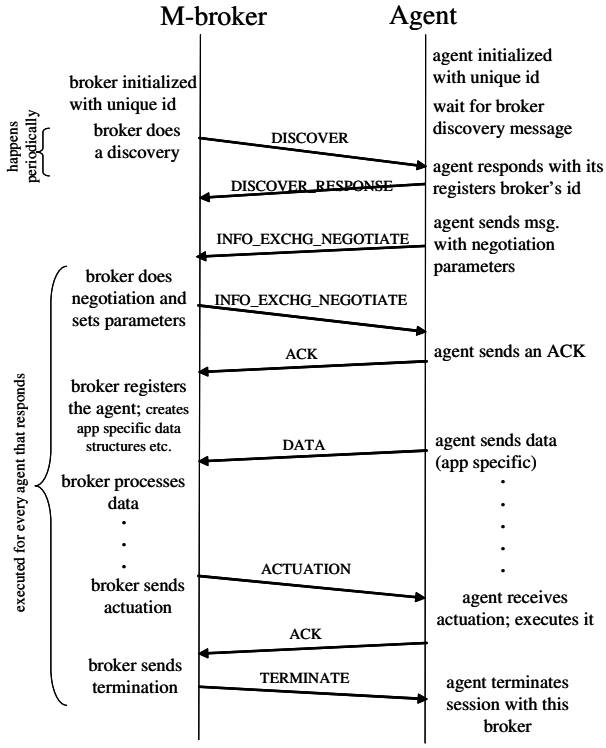

Figure 6: Message Exchange between the M-broker and Agents

The M-brokers are policy managers that execute co-ordination policies. Co-ordination is between management hardware and applications, as well as across virtual machines. A given system may have several M-brokers, one for each management application. Further, M-brokers may be distributed across guest VMs, MVM, and the hardware. Figure 4 shows how M-brokers fit into the bigger picture, serving as a bridge between existing application level management and platform management, with actuations applied to applications and hardware. Closely tied with the notion of M-broker is that of a M-agent (or agent). Such an agent is a software module that interacts with existing application and hardware management applications and serves as a proxy for sending management information to other agents or broker. Unlike brokers, however, an agent does not execute any co-ordination code as such and is only responsible for monitoring, passing information and receiving actuation commands.

M-brokers can be instantiated with different deployment strategies at a single and distributed system, as illustrated in Figure 5. Figure 5(a) shows the case when the broker is at the MVM and all the agents are in the guest VMs and hardware. In this case, the broker can cater to the co-ordination across VMs and also across hardware and software. The other possibility (Figure 5(b) is for the broker to reside at the guest VMs themselves, or at the platforms' management controllers. In this case, the co-ordination of interest is hardware-software co-ordination either initiated at the guest VM or at the hardware. Finally, Figure 5(c) shows the case when we have distributed brokers either in a single system or in a distributed system context. Here, the brokers in the different VMs communicate and jointly make co-ordination decisions. In the case of distributed systems, these correspond to brokers in the MVMs.

The brokers and agents periodically exchange messages for purposes of co-ordination. As mentioned, these messages are exchanged using m-channels. A well-defined application-level protocol is used by the brokers and agents to perform information exchange, co-ordination, and actuation. Figure 6 shows an example timeline of message exchanges between a broker and agent. The messages in

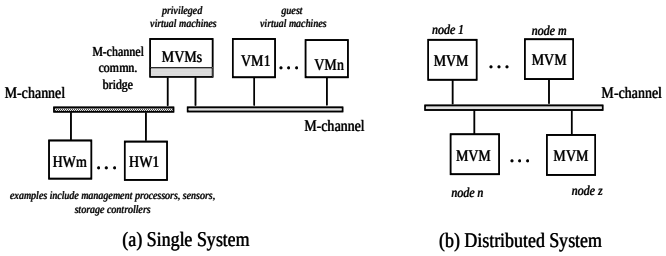

Figure 3: Conceptual view of M-channels. It can be instantiated at a single system or in a distributed system context

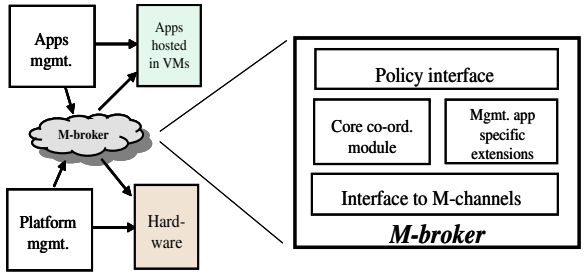

Figure 4: M-broker framework and its relationship to existing system

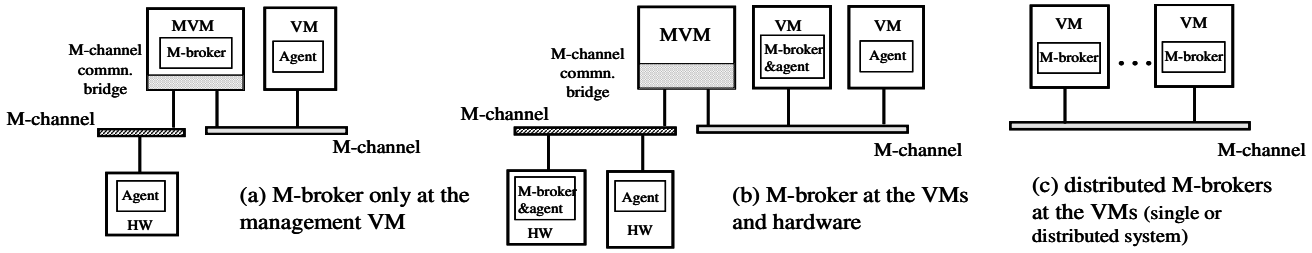

Figure 5: Various example deployment strategies for M-brokers

the figure are annotated with the message types. Overall, message exchanges belong to the phases of discovery, communication establishment, message exchange, and connection termination. The addressing among the brokers and agents take place using a unique identifier associated with each of them. Such an addressing needs to span both hardware and software components, and must be uniquely identifiable over a distributed systems. We use a tuple $< Component\_ID, App\_ID >$ as the identifier. For the agent or M-broker in the hardware component, $< Component\_ID >$ corresponds to the machine serial number. For the agent/M-broker in VM, $< Component\_ID >$ corresponds to the $< MVM\_IP, VM\_ID >$ where MVM_IP is the IP address of the management VM and VM_ID is the unique virtual machine ID assigned by the management VM to the VMs in the system (including itself). App_ID is the unique name given to the management application for which the co-ordination is being done, for example, "Power", "Storage", etc. These names for management applications are uniquely assigned by administrators.

Figure 4 also shows a skeleton structure of the m-broker framework. As shown in Figure 2, the M-broker builds on top of m-channels, and it uses the M-channel APIs to implement the co-ordination modules. There are some core co-ordination modules that implement the basic protocol for communication among the brokers and agents. These modules can be reused by all co-ordination brokers for the different management applications. Extensions specific to management applications can then be provided over the basic framework, thus constituting an extensible design.

3.4 Benefits

Overall Benefits. The proposed abstractions provide a *generalized* framework for manageability co-ordination in virtualized systems. A distinctive aspect is the co-ordination across hardware and software layers, as well as across VMs. Having better interaction between guest VMs, privileged VMs, hypervisor, and the hardware platform provides the management applications with the *holistic view* of the system and helps them take better management decisions. The addressing scheme and M-channel abstractions hide the hardware type details from the VMs allowing the brokering to not be concerned about low level hardware component nuances. This also ensures management continuity *across VM migrations*. Finally, M-channel names and access are independent of VM location, essentially implementing a publish/subscribe paradigm. As a result, VMs use the same M-channels throughout their operation, including upon migration. The MVM implementation of M-channels performs the appropriate name resolution to ensure this property. In summary, our approach results in *improved and enhanced capabilities* for management applications. Given the importance and growth of management applications in next generation enterprises, these improvements provide significant benefits over current state of the art.

M-Channel Specific Benefits. M-channels provide *seamless* communication among VMs, hardware, and MVM. This not only provides for new management possibilities but from an infrastructure development point of view, it also hides the details of the underlying platforms and physical channels from the management co-ordination applications. Further, its *restricted and clear set of APIs* can be used for co-ordination, with pre-built support and error checking for management control structures. The delivery model of messages is *flexible* thus catering to the needs of a wide variety of management applications.

M-Broker Specific Benefits. The *well defined structure* and model provided by M-brokers provides for a systematic development and deployment of co-ordination systems in virtualized environments. Leveraging m-channels allows the broker code to be *seamless* across various deployment strategies and configurations. The design of the framework allows for *re-usability* of modules across different usage scenarios and is easily *extensible*.

4. XEN IMPLEMENTATION

In this section, we show how the abstractions presented in Section 3 map onto a real virtualization environment. We use Xen as an example VMM to implement the abstractions. Figure 7 shows the Xen specific implementation. The MVM is instantiated as""Dom-M" and the guest VMs are referred to as "DomUs". The intra-machine M-channels are implemented using Xen provided inter-domain shared memory communication API. The M-broker and M-agent interface to M-channel supports user and kernel based implementations. The rest of the section describes each of these components in more detail.

4.1 M-Channel Implementation

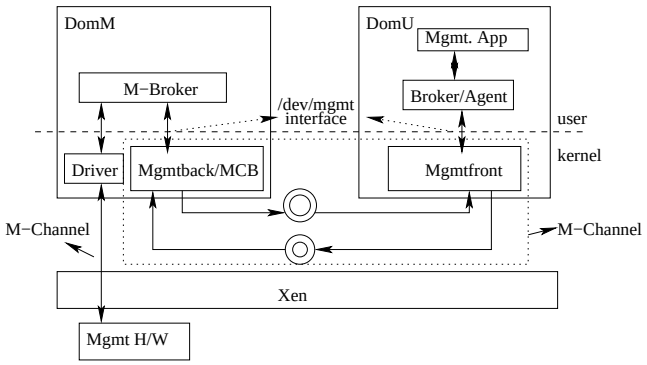

Figure 7: M-Channels and M-Brokers implementation in Xen

Challenges in realizing M-channels are caused by their use in different settings and for different management strategies. For instance, since M-channels must continue to operate even when VMs migrate between different computing platforms, they must be capable of using message-based data transports. At the same time, potentially high rate interactions between VMs and management domains when monitoring for failures in reliability management require an implementation enabling substantial data exchanges between VMs and MVMs. To accommodate diverse requirements, our Xen realization of M-channels uses multiple implementation methods, described in more detail below.

M-channel is an abstraction which can be realized through different implementations. M-channel between MVM and management hardware can be implemented by using a device driver in MVM which handles the particular hardware and exports device specific interfaces to M-brokers. The M-channels between VMs and MVM can be implemented in three ways, Specially (1) Using traps/interrupts, in which case the VMs implicitly generate traps or interrupts (e.g. by executing some privileged instruction) which causes the VMM to transfer control from one VM to another. The data can be transferred using processor registers or through memory. This method usually doesn't require any changes to the guest VM but at the same time this method is not generic and may not work for all management applications. (1) The M-channel can be implemented explicitly using the VMM provided inter-domain communication channels. These channels typically use shared memory communication with very good latency characteristics which could enable management applications to react quickly to various notifications. This method usually requires modifications to the guest VMs (to provide explicit communication) but at the same time it is a generic solution for an M-channel which should work for all management agents. (2) The M-channel can also be implemented over the network (e.g. using Socket APIs). This is also an explicit communication channel and hence a generic solution to be used with management agents. This solution, however, has higher latency compared to shared memory communication. This solution, however, enables M-channels to span multiple physical machines creating M-channels between MVMs running on different platforms. Table 1 evaluates the latency characteristics of the two realizations of M-channels.

Table 1: Latency characteristics of different M-channels

M-channel type	Latency of small (64b) data transfer
Trap Based	8 micro sec.
Shared Memory Based	7 micro sec.
Socket Based	14 micro sec.

The last two implementations of M-channels are used for inter-VM communication depending on whether the VMs are on the same machine or on the network. The M-channels between local VMs utilize the inter-domain communication APIs provided by *Xenbus* to provide shared memory communication. A management frontend (FE) driver module runs inside guest VM which communicates with the management backend (BE) inside MVM. FE and BE represent the M-channel endpoints for the guest VM and MVM respectively. The communication is asynchronous and hence uses two different communication rings for the two directions (send and receive). When the FE driver loads, it allocates pages for the two rings and shares with the backend. Both FE and BE export a file interface (e.g /dev/mgmt in Dom-M) to user-level brokers and applications and an API interface to broker and agents running in 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. A typical M-channel ring data element consists of three fields; an M-channel header, followed by an application-specific header and followed by application specific data if any. This design creates a generic, simple and yet flexible infrastructure for transferring management related messages between VMs.

For distributed M-channel implementation, it uses a socket based communication interface which provides the same APIs as the shared memory based implementation. Specifically, all the M-channel endpoints run a TCP/IP server which listens on a well-known port and accepts connections from other M-channel endpoints. While currently not implemented, authentication mechanisms can be used to implement to establish secure channels between different VMs. In both the implementations the application 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. As shown in Figure 7, the M-channel communication bridge *(MCB)* between the various FEs and the BE are implemented as part of the BE which routes messages based on the address contained in the M-channel header. Similarly the bridge between the VMs and the management hardware is implemented as part of M-broker inside Dom-M which routes messages between them according to its policies (e.g. pass-though vs. virtualized access).

The M-channel enables the brokers and agents to co-ordinate over VM migrations. During VM migration, the mgmtfront and mgmtback modules get notified of the VM migration event which triggers a new set of disconnections and reconnections i.e., the older mgmtback breaks its connection with the mgmtfront and new mgmtback establishes a new connection with the mgmtfront. This enables the agents and brokers inside guest VMs to remain transparent to migration and still be able to communicate with the current platform's M-broker in Dom-M.

4.2 M-Broker Implementation

The M-brokers and their agents, as presented in Figure 4(b) are implemented as multi-threaded applications. The actual management application-specific broker code is implemented as another thread in this application. These brokers and agents, however, are application specific and may be implemented in kernel level as well. It needs to be multi-threaded because it has to communicate with multiple entities which usually operate independent of each other (e.g., management h/w, policy maker, M-channel etc.). Different threads of the M-broker handle the communication with other VMs, management h/w and decision making algorithms. These brokers communicate with other agents and brokers on local machine using shared memory M-channels and with brokers on other machines using socket based M-channels. To access the management h/w, the M-broker utilizes the "driver" provided M-channel interface (similar to /dev/mgmt) to read from and write

to the device. In the current implementation, the M-Broker's interface channel component also works as the bridge between the VMs and the management h/w, i.e. all the accesses to the management h/w from VMs must go through the M-broker.

Security (i.e., authentication and authorization) is not implemented in the current M-Broker and M-channel interfaces and is an on-going work. Because of the M-channel handling of VM migration, the Brokers and agents remain oblivious to such events, and local M-channels get re-established as local channels after migration. We have implemented a power-broker for power management which evaluates various power management policies and an inventory registry broker which builds co-ordinated inventory tables. Section 6 qualitatively describes other use cases surrounding a storage backup broker and a "trust management broker".

4.3 Dom-M: The Management VM

Dom-M is a privileged management VM which is the dedicated point to control all management tasks on a single platform. E.g., Dom-M co-ordinates between VM requirements and platform management policies with the help of specific management hardware and inputs from VM's applications. Dom-M internally consists of three main components: (1) one or more M-brokers, (2) a high-level policy maker, and (3) various M-channels. The policy maker is a management application that provides access to various policy controls to the system administrator. It can set the policies and goals for the M-broker and provide the status of management actions to the administrator. Dom-M is a privileged and trusted VM with direct access to platform's hardware which sense and actuate useful management actions (e.g., management processor).

In some VMM solutions (e.g., VMWare's ESX server) where there is no existing management VM, Dom-M will be a separate, new entity. Other VMM solutions, which already have a privileged and trusted VM as part of their architecture (e.g., Dom0 in Xen or the host VM in VMWare's GSX server), permit Dom-M to be realized as part of such specialized and privileged VMs. However, there are several advantages of separating and isolating Dom-M which are: (1) Security and Isolation:- since Dom-M has direct access to management hardware, running it as a separate VM will isolate it with respect to driver failures/hardware misbehavior (similar to driver domains for I/O devices in Xen). (2) Selective Privileges:- Although Dom-M requires certain privileges to do management, its applications don't require all the privileges of privileged VM like Dom0. Hence, to further improve the reliability of the system, it is advisable to use a separate VM to which selective privileges can be provided. (3) Specialized Management Cores:- Future heterogeneous multicore systems will contain cores which would be specialized to certain type of processing. One such type would be management cores (e.g., the current off-chip management processor could move inside the chip). In such scenarios, it would be more efficient to abstract out all management applications inside a VM and run this VM on a dedicated management core. (4) Virtual Appliances: future virtualized environments are likely to make heavy use of virtual appliances [25]. Having a separate virtual machine for Dom-M simplifies its deployment in the form of a virtual machine appliance, since in this form, Dom-M can be independently deployed from the VMM, i.e., it can be dynamically added or removed from a system.

In our deployment Dom-0 is used to implement Dom-M. Currently we are working on creating Dom-M as a separate privileged VM and packaging it as virtual appliance.

5. BENEFITS FOR POWER MANAGEMENT

Effective power management requires timely and intelligent use of underlying management capabilities in order to balance possible savings with application performance requirements. Our architecture allows for management methods that meet this criteria with its abstractions of M-channels and M-brokers. Specifically, our evaluation of its usefulness in power management highlights two significant aspects of the system: (1) the M-channel abstraction provides flexibility in the implementation of coordination allowing for either a distributed or centralized M-broker for power management and (2) the ability to define management specific data to be passed through M-channels provides generality for utilizing varying levels of abstractions and also allows for richer information flow for improved management. In the remainder of this section, we describe and evaluate three M-brokers applied for coordinated power management with our architecture.

We utilize two different testbeds in our experiments to evaluate the benefits of power management across platforms based on both AMD and Intel processors. Table 2 summarizes the experimental methodology for the three M-brokers. Our implementation is based around active management of power via dynamic frequency and voltage scaling (DVFS) of the CPUs and VM migration. The purpose of these implementations is to show that M-channel and M-broker abstractions can be easily used to implement various power management policies ranging from traditional utilization based management to novel SLA based management and in co-ordination with other VMs and platform requirements. In the next few sections, we detail the particular implementations of these brokers and complete the section with a review of the evaluation summary.

Table 2: Experimental Methodology

Platform	Broker	Co-ord. Type	Actuator	Metric
AMD Opt.	B1	VM-VM	DVFS	SLA and Util.
AMD Opt.	B2	VM-Platform	VM Migra.	Power Cap
Intel P4	B3	VM-VM	DVFS	SLA

5.1 Implementation of M-brokers for Power Management

5.1.1 M-Broker1

The first power management M-broker instantiated on this architecture corresponds to Figure 5(a). In this approach, a power M-broker runs inside Dom-M and VM agents run inside guest VMs. The power broker establishes M-channels with VM Agents. The VMs run an application with some service level agreements (SLAs) specified for them which is monitored by VM agents. The individual agents use the M-channel to notify the broker of SLA violations which helps the broker in doing power management. The broker runs administrator specified policies, which sets the power state of the physical CPUs by using the SLA violations as inputs and coordinating among them. The application specific message formats within the generic M-channel headers 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 power states of resources.

Our first testbed consists of dual-core dual-socket (total 4 cores) AMD 64-bit Opeteron based HP 's 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 for co-ordination among VMs. Guest VMs are non-SMP and run the HTTP server workload, and httperf is used to generate load for the 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 which increases the p-state

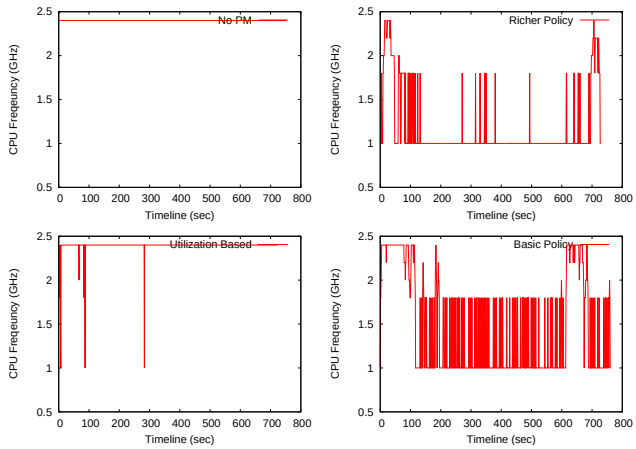

Figure 8: CPU frequency traces for various policies

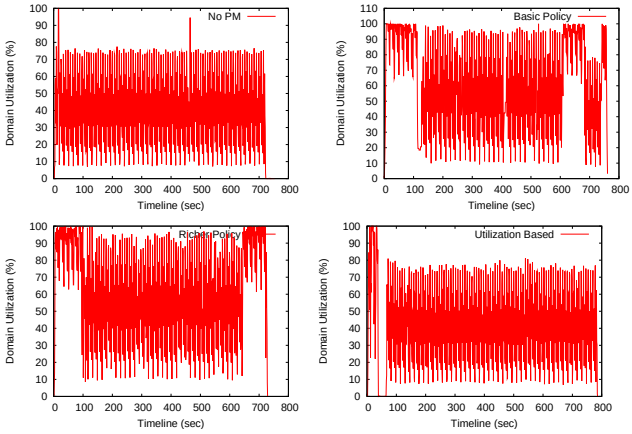

Figure 9: Domain utilization traces for various policies

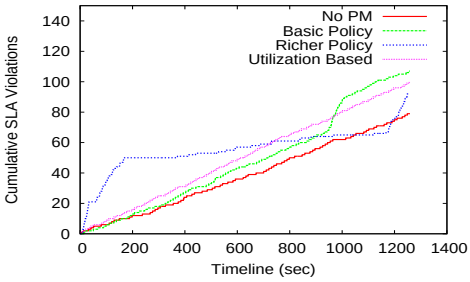

Figure 10: Cumulative SLA violations for various policies

of the VM's vcpu to the next higher value. A parallel thread also monitors VM's cpu utilization and whenever the utilization goes below a certain threshold (80%) the VM's vcpu is reduced to next lower p-state. For a particular physical CPU, all the VMs' vcpu p-states are examined and the highest p-state is set as the p-state of the CPU.

(2) Richer policy: In the second policy, we deploy a more sophisticated notification criteria where the SLAs are defined as *hard* (higher response time threshold) and *soft* SLAs (lower response time threshold). With each notification, 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 SLA Policy 1. For soft violations, however, the policy records these violations for 10 consecutive epochs and if the soft violations happen more than 30%, it is treated as a hard SLA violation and action is taken according to *Basic Policy*, otherwise 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 further allows the broker to implement more sophisticated policies.

(3) Utilization-based: In the third policy, we use an agent, which utilizes a Linux VM based *ondemand* power governor to decide the power state of the vcpu based on its current vcpu utilization. This power state information is notified to the power broker through an M-channel which uses the coordination policy of *Basic Policy* to decide the final p-state of the CPU. We compare the three policies against the "no power management (No PM)" case for a varying httperf workload. It is important to note that all the thresholds and parameters in the above policies were chosen arbitrarily and the purpose was to demonstrate the flexibility and richness of M-broker and M-channel abstractions.

Results: Figure 8 shows the trace of the speed 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 is shown in Figure 11, it is still able to save about 10% power by putting the three other less-utilized cores into lower power states. Figure 9 shows the VM's cpu utilization trace as seen from Dom-M and as the VM is run. We see that Dom-M actually sees the real CPU utilization for Utilization based policy to be less than Basic and Richer policies underscoring the fact that VM based policies are often inefficient in system monitoring because of virtualization layer and must coordinate with Dom-M's M-broker. The Basic Policy reacts well to SLA violations and changing domain utilization because it coordinates with Dom-M using its view of CPU utilization and Dom-M in turn uses SLA violations as input in taking decisions. It saves power by operating mostly at the smaller CPU speeds while still trying to maintain the SLA. The Richer Policy in 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 lesser number of total hard violations over intervals which allows the policy to run the CPU predominantly at the lowest speed. Figure 11 shows that this leads to the Richer Policy providing the most power savings. Figure 10 shows the cumulative number of SLA violations for different policies. We see that 'No PM' policy expectedly performs the best. However, Basic Policy, Richer Policy and Utilization Based policies show similar characteristics with Utilization based policy only slightly outperforming the Basic one. The results also demonstrate that for policies focused on SLA violations instead of CPU utilization, SLA based power management prove to be more efficient at power savings that traditional utilization based policies.

5.1.2 M-Broker2

To demonstrate the hardware/software co-ordination between the VMs and platform's power limitations, we have extended the *Basic Policy* broker which takes SLA inputs from the VMs and platform's power limits (both average and peak power) and en-

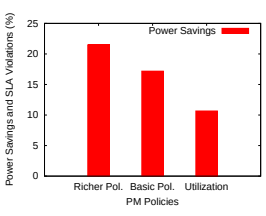

Figure 11: Power Savings

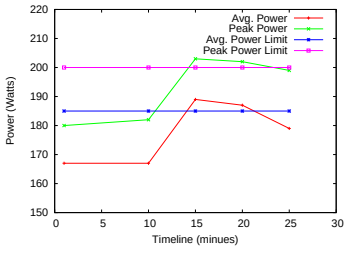

Figure 12: Power Capping

forces the limits while trying to maintain the SLA levels for the VM. We have used HP's C-class blades, which have a management processor called iLO (integrated lights out) built into them which records current power consumption of the platform. The M-broker establishes M-channels with other M-brokers running on other machines in the data center to co-ordinate for VM migration as well as an M-channel with iLO management processor.

SLA Model: The M-broker periodically (every 5 minutes) queries the management processor for current avg. and peak power consumption. If these limits are violated (probably because of increase in the VM load), as an example policy, the broker first reduces the p-state of a processor (or a set of processors) to bring the power consumption within the limits. If this action causes a sharp increase in SLA violations, the M-broker queries other broker's 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 maintain the SLA levels of the application. The current prototype implements VM migration and the final version of the paper will include numbers in co-ordination with SLA based power management.

Results: Figure 12 shows the results of this experiment. A machine is running 4 VMs containing a WebServer which get heavily loaded by httperf clients. The avg. and peak power limits are defined as 185 and 200 watts respectively. We see that after 10 minutes, as the load increases in all the VMs, the power limits (both avg. and peak) are violated. The broker finds out a suitable machine and migrates the most heavily loaded VM. This brings down the power consumption slightly but it is still above the set limit. The broker (at 20 minutes) migrates another VM which finally brings down the power consumption within limits. The channel between the broker and the iLO processor enables the system to react quickly to power limit violations, which otherwise is not possible or will have to use some other metric (e.g. cpu utilization) as a proxy for power consumption which will not be as accurate. This example clearly shows the the importance of having co-ordination between the management policies and the platform hardware to ensure efficient operation of the data center.

5.1.3 M-Broker3

In this design, the functionality of application aware power management is distributed between the Dom-M and the guest VMs (see Figure 5). In particular, each guest VM runs its own application specific power management broker based upon power states exported to it via ACPI [5]. These brokers are charged with performing power management themselves by determining power states for the respective VM's resources based on certain policies (e.g. CPU utilization based, SLA violations based etc.). These brokers, then, send the requested power states to the M-broker in Dom-M over M-channel interface.

This M-broker uses the testbed consisting of dual core Intel Pentium 4 based platform. These processors provide hardware frequencies of 3.2GHz and 2.8GHz. In addition, we implement the use of soft scaling [17] wherein the service time provided to VMs is reduced in addition to frequency scaling in order to increase idle time thereby reducing power consumption and thus providing 3 additional virtual frequencies 2 GHz, 1.6 GHz and 800 MHz. This testbed is used for experimentation with the distributed M-broker usage model and a workload based upon the Nutch open source web search engine.

SLA Model: To drive the system, we define a differentiated service model wherein there are three levels of client classes. We define a light request and a heavy request where the heavy request needs twice as many search responses to be found for completion. The three client classes are defined as 1, 2 and 3 which consist of light, a mix of light-heavy and heavy requests respectively. Their SLA metric is defined as request execution time which is set to 20ms, 25ms and 30ms respectively for the three classes. The M-broker inside VMs monitors the request execution times for each client

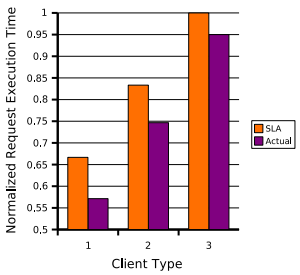

Figure 13: SLA results with Nutch

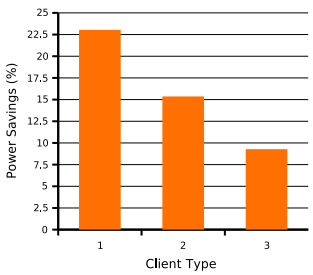

Figure 14: Power Saving with Nutch

class and calculates averages within epochs lasting five seconds. Within these epochs, at each second interval the policy looks at the current measured average, and if there is performance slack greater than 25% of the SLA, the policy requests a reduced power state. If the performance slack is less than 25% but positive, the policy leaves performance where it is, and if it drops below the SLA, it requests an increased power state. The second half of the distributed M-broker in Dom-M, responds to these requests with direct mappings to physical frequencies and processor allocation capacities [17].

Results: Figure 13 presents performance results with our Nutch based experimentation. We see that across all of the runs, the co-ordinated management based upon the distributed M-broker is able to meet SLA requirements. Figure 14 presents the power savings resulting from coordinated management of Nutch. The results provide power savings compared to a system without M-channel support wherein the M-broker in Dom-M cannot perform any management since it has no application specific information. Moreover, with the Nutch workload, even utilization based management based on observing VM CPU usage is ineffective since utilization is always high.

5.1.4 Summary

The ability to easily implement various power management policies architectures with different characteristics using M-broker approach to utilize M-channels underscores the generality and flexibility of the M-channel abstraction and therefore its appropriateness for providing general management functionality. All the above experiments underscore the importance of co-ordinated management in virtualized environments, both across VMs and across hardware and software. Another key insight shown by these experiments is that different policies have different performance characteristics under different situations and the management infrastructure should be able to provide a flexible model to implement those policies and our M-broker and M-channel abstractions provide just that.

6. BENEFITS FOR OTHER USE CASES

6.1 Storage Backup

Storage backup can be improved in many ways by using this management architecture. Storage virtualization involves every disk access from VMs being handled by the VMM (or in many cases the disk driver domain) which accesses the disk on the VM's behalf, thereby providing the opportunity of externally tracking VM disk activity. Hence *Dom-M* can co-ordinate with the VMM or disk driver domain using an M-channel to keep track of all the disk blocks a VM modifies. This information can be used to create a replica by writing modified blocks to a 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 do efficient incremental backup since a list of modified blocks is maintained.

Most disk drives implement the S.M.A.R.T. system which provides online information about disk health and performance. An M-channel to the disk drive can be used to efficiently use this information for predicting impending failures and thus doing proactive backup. In such cases, to minimize possible data loss, the backup system can prioritize blocks that are known to be modified. Since *Dom-M*'s backup broker can also monitor disk activity of VMs using M-Channels to the disk driver domain, it can dynamically trigger backup when a VM's disk activity is relatively low and pause backup when disk activity becomes high.

6.2 Inventory Management

Inventory management presents a different category of management applications dealing with systems management issues in data centers. Our proposed *Dom-M* implementation addresses the inventory management problem by providing the abstractions through M-channels for information exchange among the various inventory agents at the hardware, software, and virtualization layers. The coordination m-brokers in this solution are distributed at the guest VM, the firmware, and the Dom-M. 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, then 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, then the m-broker at the Dom-M can perform the unification. The m-broker at the Dom-M management VM also provides a single client-facing interface to simplify administrator access to inventory information. In such a design, the m-broker at the Dom-M hosts an inventory registry broker to which all of the inventory agents at the guest VMs, firmware, and Dom0 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 Dom0, and the one hosted at the 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, hypervisor, and guest VMs, even in the absence of any common attributes across the individual inventory tables. This information can now be used for better unified management in the data center. Further, as mentioned earlier, leveraging a single access point at the *Dom-M* 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 containing 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 huge complexity and cost savings.

6.3 Trust Management

Virtualization can be used to implement entirely new platform-level functionality, enabled by management activities transparently associated with platforms and applied to the VMs that use them. One example of such functionality concerns 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 improve total platform trust and/or ensure certain minimum levels of trust to be present for running applications. Here, trust is built incrementally by the hypervisor using the Trusted Platform Module (TPM) for basic platform guarantees, then using these to certify trust in *Dom-M*, (termed 'trust controllers in the literature [9]), then having *Dom-M* perform monitoring, execute trust models, and finally, trigger actuators to maintain the levels of trust desired by applications or data center administrators.

Here, VMs' M-agents can assist in VM monitoring, using M-channels to provide trust data to *Dom-M*, but *Dom-M* must have its own, additional monitoring methods to deal with misbehaving VMs. All policies are run in *Dom-M*s, with each single *Dom-M* responsible for its platform and platform-resident VMs, but multiple *Dom-M*s 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 co-ordination across all *Dom-M*s involved in tier execution. Further, to deal with untrusted platforms, *Dom-M*s must coordinate to move VMs from less to better trusted machines, and/or to kill and reinstate VMs if necessary.

7. DISCUSSION AND LESSONS LEARNED

The M-channel and M-broker abstractions improve system manageability by making it possible to coordinate management actions intended by VMs, carried out by management domains, and assisted or partly implemented by hardware. Further, the cost of manageability is reduced by provision of a common framework for developing and deploying rich, automated management applications.

Several interesting lessons were learned from this research. First, there is a need for multiple M-channel implementations, so that it is equally easy to interact with local management hardware as it is to remotely access and perform management actions (e.g., due to VM migration). Further, such channels must persist under disconnection and reconnection and use naming schemes suitable for these facts. In other words, M-channels must provide a relatively reliable communication model. Second, the M-channels and M-brokers must be flexible and rich in that, they should provide management applications the flexibility to implement various policies and define their own communication formats. Having this flexibility further improves manageability and generic nature of these abstractions. Third, M-channels and M-brokers must permit multiple implementations of management policies, distributed spatially as per platform or per rack, distributed by the kind of management being performed (e.g., power or reliability management), etc. Regardless of this fact, however, it is always the case that they must facilitate potentially extensive co-ordination across such policies.

Our realization of the M-channel and M-broker abstractions has several implications on participating or affected system components. Specifically, while they provide a convenient infrastructure for deploying and running management applications, they also require changes to VMs (e.g., to provide M-channel interfaces) and to management applications (i.e., to provide M-brokers and agents). We believe, however, that with para-virtualization becoming increasingly popular, these modifications do not constitute an undue burden. Moreover, we do not require changes to hypervisors, other than creating MVMs and imbuing them with the levels of privilege required by the management actions they carry out.

8. RELATED WORK

Modern data centers require the simultaneous deployment of multiple management solutions. Examples include hardware based management solutions such as temperature control, cooling,... *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 resource in distributed environments, such as SNMP, WBEM, CIM etc. These are mainly software based solutions and usually do not co-ordinate with the platform hardware on which they run.

With virtualization technologies such as Xen and VMWare [1,

23] entering the data center, the manageability and co-ordination problem is becoming increasingly important, underscoring the need to integrate system support for effective management into virtualization architectures.

Various management methods have been proposed to address power characteristics of platforms. Mechanisms for exploiting dynamic voltage and frequency scaling (DVFS) capabilities in modern processors while minimizing performance degradation have leveraged memory bound phases of workloads. These solutions can be extended using hardware solutions [12] or OS-level techniques that set processor states based upon predicted application behavior [8]. Other methods utilize the notion of performance slack for real-time workloads to aggressively reduce frequencies while meeting deadlines [18, 19]. Power budgeting solutions for single platforms have also utilized processor control to provide fine grain power capping capabilities [11]. In the presence of virtualization, the ability to perform energy accounting can also allow for power budgeting [22].

Addressing multiple platforms, methods for reducing power by turning servers on and off based on demand have been proposed [2]. Since datacenter environments often consist of heterogeneous sets of resources, intelligent distribution of service requests [4] and allocation of workloads based upon varying platform characteristics or management capabilities [15] have proven to be valuable ways to manage power as well. Solutions for power budgeting across multiple systems have considered non-uniform allocations to maximize performance within envelopes [3] as well as intelligently enforcing power budgets at blade enclosure granularities [20]. Providing system support for extending such management techniques to virtualized environments with existing virtual machine interfaces has also been studied [17].

Modern disks support the self-monitoring, analysis, and reporting technology (S.M.A.R.T.). Systems can utilize this support to actively predict impending disk failures and perform associated actions to prevent data loss. Industry is also addressing the need for intelligent backup systems, including integrated solutions for commodity systems [6] and extensions to virtualization solutions [24].

9. CONCLUSIONS AND FUTURE WORK

This paper presents two powerful abstractions, termed M-channel and M-broker, for co-ordinated management in virtualized execution environments and an abstraction of a dynamic, and privileged management VM termed Dom-M. These abstractions provide high level generic interfaces for implementing diverse management policies, resident in VMs, in management domains, 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 a few other 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 future steps to include implementing additional management applications using these abstractions and implementing the infrastructure required for co-ordination across the many, distributed systems resident in the data center. Further, the current *Dom-M* implementation is being generalized to become a virtual appliance, suitable for dynamic deployment in realistic enterprise systems Additional work in this context involves using specialized or dedicated cores in heterogeneous multicore systems for running *Dom-M*, to enable rich, continuous management with low overheads and costs.

Overall, as management applications continue to grow in importance, we believe that abstractions like the ones we proposed in this paper will become even more important in the future. With the final paper, we also plan to release the the M-channel and M-broker APIs as libraries to the open source community, and our hope is that this will encourage broader development around these abstractions and subsequent refinement.

10. REFERENCES

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

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

[3] M. Femal and V. Freeh. Boosting data center performance through non-uniform power allocation. In *Proceedings of the IEEE International Conference on Autonomic Computing (ICAC)*, 2005.

[4] T. Heath, B. Diniz, E. V. Carrera, W. Meira Jr., and R. Bianchini. Energy conservation in heterogeneous server clusters. In *Proceedings of the 10th Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2005.

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

[6] HP Data Protection. http://www.hp.com/sbso/bus_protect/data_protect.html.

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

[8] 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.

[9] 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.

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

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

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

[13] 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.

[14] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making scheduling "cool": temperature-aware workload placement in data centers. In *ATEC'05: Proceedings of the USENIX Annual Technical Conference 2005 on USENIX Annual Technical Conference*, pages 5–5, Berkeley, CA, USA, 2005. USENIX Association.

[15] R. Nathuji et al. Exploiting platform heterogeneity for power efficient data centers. In *ICAC*, June 2007.

[16] R. Nathuji and K. Schwan. Reducing system level power consumption for mobile and embedded platforms. In *Proceedings of the International Conference on Architecture of Computing Systems (ARCS)*, March 2005.

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

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

[19] C. Poellabauer, L. Singleton, and K. Schwan. Feedback-based dynamic frequency scaling for memory-bound real-time applications. In *Proceedings of the 11th Real-Time and Embedded Technology and Applications*

Symposium (RTAS), March 2005.

[20] 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.

[21] E. Shih, P. Bahl, and M. Sinclair. Wake on wireless: An event driven energy saving strategy for battery operated devices. In *Proceedings of the Eighth ACM International Conference on Mobile Computing and Networking (MOBICOM)*, pages 160–171, September 2002.

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

[23] 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.

[24] VMware Consolidated Backup. http://www.vmware.com/products/vi/consolidated_backup.html.

[25] VMware Virtual Appliance. http://www.vmware.com/appliances/.