Power-aware Dynamic Placement of HPC Applications

Akshat Verma
IBM India Research Lab
akshatverma@in.ibm.com

Puneet Ahuja
IIT Delhi
puneet1986@gmail.com

Anindya Neogi
IBM India Research Lab
anindya_neogi@in.ibm.com

ABSTRACT

High Performance Computing applications and platforms have been typically designed without regard to power consumption. With increased awareness of energy cost, power management is now an issue even for compute-intensive server clusters. In this work, we investigate the use of power management techniques for high performance applications on modern power-efficient servers with virtualization support. We consider power management techniques such as dynamic consolidation and usage of dynamic power range enabled by low power states on servers.

We identify application performance isolation and virtualization overhead with multiple virtual machines as the key bottlenecks for server consolidation. We perform a comprehensive experimental study to identify the scenarios where applications are isolated from each other. We also establish that the power consumed by HPC applications may be application dependent, non-linear and have a large dynamic range. We show that for HPC applications, working set size is a key parameter to take care of while placing applications on virtualized servers. We use the insights obtained from our experimental study to present a framework and methodology for power-aware application placement for HPC applications.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Miscellaneous

General Terms

Performance

Keywords

Power Management, Placement, High Performance

1. INTRODUCTION

Power management in server clusters is an area of increasing interest from research viewpoint as it is backed up by real concerns on energy usage by modern computing systems. Power management in High Performance Computing (HPC) clusters is still a new concept as it is deemed contradictory to the performance demands from such applications. In this paper, we investigate the possibilities enabled by the platform power management and virtualization mechanisms coupled with the savings opportunities present in typical HPC cluster workloads.

Server virtualization presents new opportunities for power management using workload characteristics as controllers can perform power-aware dynamic application placement. Earlier, virtualization had proliferated the desktop segment as it enabled users to run multiple operating systems on the same box and helped the average user overcome OS-specific usability issues. Similarly, till recently power management was relevant only to personal mobile devices with focus on battery and display technologies. Recently, virtualization and power management technologies have both penetrated the server platforms and present a fresh perspective on resource management in cluster computing space.

Server virtualization has a number of advantages. It simplifies the way we manage a set of machines with common management controls. It is easier to manage software stack deployment and lifecycle management through images and the ability to power-on/off, archive, migrate containers and their workloads. Virtualization allows consolidation of a number of smaller physical server workloads into partitions of a larger server. Since the virtualization layer provides a high degree of isolation among partitions, the user achieves the same level of performance and security, but at a lower management cost and possibly lower hardware cost. Virtualization also provides finer grain control of resource allocation. The dynamic resource allocation requirements of a workload can be satisfied by altering the capacity of a virtual machine at runtime. In fact, a virtualization layer also provides live migration capabilities. This enables the resource manager to work around resource bottlenecks and faults by migrating a virtual machine onto a new physical server with a small transient impact on runtime performance.

Even though virtualization has become common in commercial transactional and batch workloads in data centers, it is only very recently that researchers have started to consider HPC workloads on virtualized platforms. HPC applications can benefit from all the advantages of virtualized environments. In addition, such applications often require OS-level customization for maximizing performance. It is easier to deploy and manage multiple OS images and instances on virtualized clusters. However, the key concern about running HPC workloads inside virtual machines is the performance impact of accessing system resources through the Virtual Machine Monitor (VMM). Modern virtualization platforms provide enough optimizations such that most of the instructions and access to system resources can execute efficiently. However, all privileged instructions issued from a guest VM have to be trapped and handled by the VMM for ensuring system integrity. Even though CPU and memory accesses can be effectively virtualized, I/O virtualization poses additional challenges as most of the accesses are to resources shared across VMs. There are various research efforts to create efficient mechanisms, such as bypass paths, to enhance the I/O performance in virtualized environments. Huang et al. [12] have recently implemented efficient access to network resources so that the HPC applications can communicate at near-native performance. Thus it is possible for HPC applications to now use virtualized platforms without a large performance impact.

Power and cooling costs are a subject of growing concern in systems research and need to be addressed in the design of clusters that run HPC applications [23]. Power management techniques in server clusters may be implemented at various levels, such as the processor, virtualization layer resource manager, the cluster resource manager etc. Modern processors, such as the Intel quad-core Xeon [31] and the IBM Power6 [7], have various on-chip power management techniques built into the processor. Feedback-driven pipeline throttling mechanisms are used to reduce the instruction flow and reduce power consumption. Use of multiple power domains inside the CMOS processor enables dynamic voltage and frequency scaling of a processor. Since power is proportional to frequency and is a quadratic function of the supply voltage, reduction in voltage with accompanying reduction in frequency has a large impact on the power consumed by the chip. Modern processors also support an additional low power state which does not idle waiting for work to arrive. This leads to a higher wake-up time but between 10-20% reduction in power consumption compared to idle mode. Since power consumption in the memory subsystem is significant in high-end servers [15] the on-chip memory controller supports logic to turn off memory banks when there is no activity or throttle the memory bandwidth to cap the power consumption. A description of power/thermal capping and performance-sensitive power savings solutions implemented in Power6 is provided in [7].

Processor level power saving techniques provide a large enough workload-dependent dynamic power range to be exploited by intelligent workload placement controllers. In addition, virtualization enables such a placement controller with flexible means to resize resource allocations and migrate workloads dynamically. In this paper, we consider a server cluster which implements virtualization, live VM migration, and hardware level power management technologies that provide significant dynamic power range depending on workload. We assume that HPC applications are submitted as jobs to the cluster, which need to be executed inside VMs and placed on physical servers dynamically by the placement controller. We describe the design issues in building such a power-aware VM placement controller for the server cluster.

The paper is one of the first to propose power management in HPC clusters and in the process makes the following key contributions:

- **Scope of Power Management in HPC Workloads:** We first study traces of large server clusters running HPC applications to investigate if there is scope for power management. We look at variability in the number of jobs running in the system to see if some servers can be switched to low power states or completely switched off. In particular, we look for medium to long term periods of low activity so that the overhead of bringing the server back does not become prohibiting. Further, we look at resource usage distribution to investigate the feasibility of dynamic range based power management. We determine that there is enough variability for both techniques to be applied.
- **Power models:** We experimentally create power models of some benchmark HPC applications on a specific platform to show that there is scope of exploiting the dynamic power range by a power-aware resource manager. These power models not only show variation of power consumed with CPU utilization but also some interesting observations can be made from the variations of power with memory usage. We show that variation in resource usage, as established by our trace-based study leads to a dynamic power range that can be exploited for power management.
- **A study of Virtualization issues due to Consolidation:** The virtualization layer is expected to provide perfect isolation among the various workloads running in different partitions. However, in our experiments with the multiple HPC benchmarks that share the same physical machine, we show through experimental validation that such isolation actually depends on the memory footprint and cache usage characteristics of the applications. This is the *key* contribution of this work and leads to a redesign of an earlier placement controller [30].
- **Placement controller:** The knowledge of power models and impact of virtualization on various types of HPC applications is used to design a power-aware application placement controller. The controller needs to take into account CPU as well as memory and cache usage characteristics of an application for placement purposes, leading to a multi-dimensional packing problem. We show how to build on an existing CPU-based controller pMapper [30] with the help of a case study using benchmark HPC applications.

The rest of the paper is organized as follows. We first investigate the scope for power management on a large HPC cluster in Section 2 using a trace-based study. Section 3 investigates the feasibility of power management for HPC applications, such as power models and virtualization issues. Section 4 describes the power management architecture and a methodology for VM placement with an HPC case study. Section 5 discusses the related work and concludes the paper with a summary and description of future work.

2. SCOPE OF POWER SAVINGS IN HPC APPLICATIONS

A dynamic placement controller is useful only when the workload of the cluster has the right amount of variability. A very high degree of variability makes dynamic placement a very expensive proposition. Low variability makes a case for static placement. We investigated several HPC cluster computing workloads and existing literature on analysis [20, 21] to find out if there exists the right amount of variability

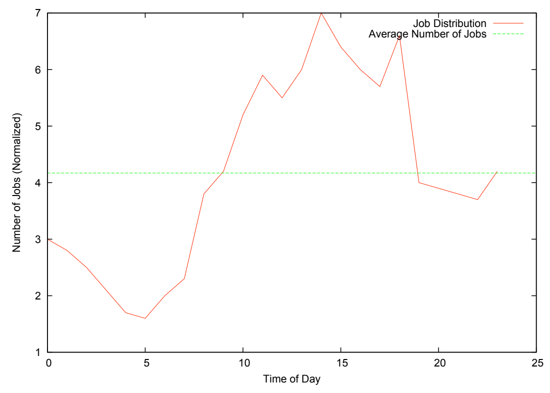

Figure 1: OSC trace showing large variability in the number of jobs during an average day.

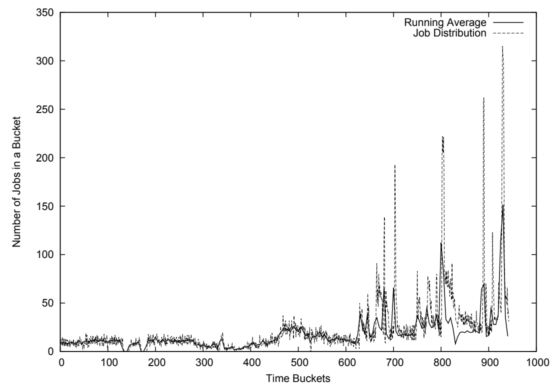

Figure 2: LPC trace showing large variability in the number of jobs during a month. The number indicate the number of jobs in a time bucket of 90 **mins.**

in shared HPC clusters to require a power-aware placement controller. More specifically, we investigate if the aggregate workload is low for a sufficiently long time to justify switching off servers or to switch them to a low power idle state. Further, we study if there is enough resource usage variability at finer grained intervals that may lead to a large dynamic power range.

Figure 1 shows job arrival and processor usage results from a representative server cluster at the Ohio Supercomputing Center [16]. It is apparent from the 24-hour averages that significantly higher number of processors are required to service jobs arriving towards the middle of the day. Thus at other times, a large portion of the workload can be consolidated to a few active servers and the rest of the servers can be switched to low power state. Analysis on LPC traces [17], captured in Figure 2, shows an even larger variation in jobs handled by the cluster during the month. We also observe distinct 'on' and 'off' periods of reasonable durations, thus providing an option to completely switch off many servers during 'off' periods. Similarly, Planetlab traces have been analyzed to show that the 10^{th} and 90^{th} percentiles of memory usage and load averages have almost a ten fold variation [21]. We now drill down further in the LPC trace, which captures a 70 node cluster that is part of a larger grid executing parallel jobs. We observe that the resource usage for various jobs shows huge variability (Fig. 3). Hence, one can expect a large variability in resource usage even for systems that service deterministic batch jobs as well. We observe that the mode and the average of CPU as well as memory are a factor of 10 away. Thus, one can safely assume resource variability even if jobs arrived in a fixed periodic manner.

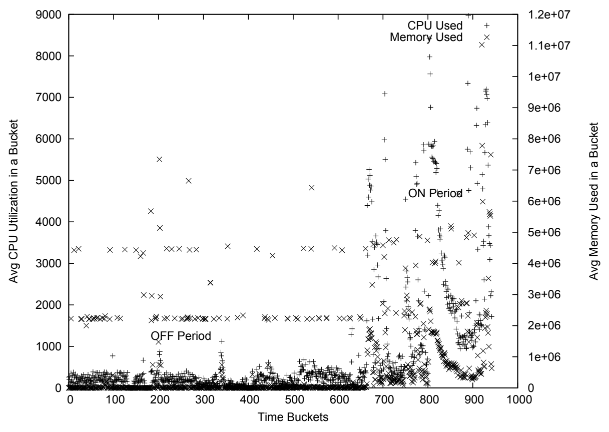

Figure 3: Variability in resource usage with time for LPC trace

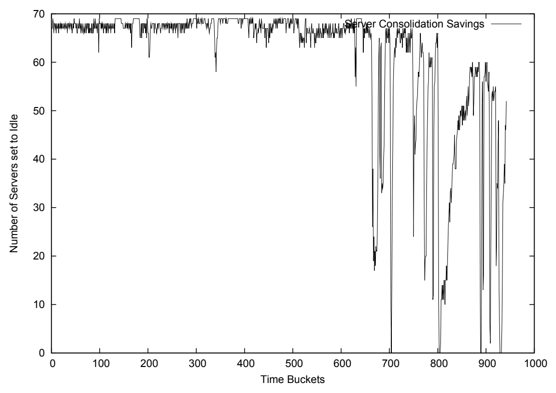

Figure 4: Power Savings with Time for the LPC trace. Servers are switched off if they are not used for 5 successive time windows.

The variability in job arrivals coupled with resource variability of the jobs in terms of CPU and memory leads to a very high variability in the actual resource consumption on the servers. Figure 3 shows that the resource utilization varies with time, thus providing enough scope for dynamic workload consolidation. Figure 4 shows simulation of a dynamic consolidation algorithm over the trace length and projected power savings achieved by turning off servers in the cluster of 70 physical servers. There is a distinct period of low activity towards the start when large number of servers can be turned off for hours. We further drill down in the periods of low and high activity and observe that there is scope for further reduction using dynamic workload-dependent range in the power models. We note that even within the 'off' period, one can observe a reasonably long duration variance by a factor of 2. The same is true for the 'on' period, establishing the feasibility of switching servers to low power state or using dynamic power range based techniques (assuming power depends on resource consumption) at finer granularities. Our experiments thus establish that there is sufficient variability in resource consumption for power management techniques to be applied. We next study the performance implications of power management techniques on a real system.

3. POWER MANAGEMENT FOR HPC APPLICATIONS

We now investigate the feasibility of applying power management techniques to HPC applications. Power management techniques include running a set of power-efficient servers at an operating point that takes advantage of the dynamic range in the server power models and consolidation of workloads on fewer servers during off-peak hours. In this section, we study the power model for some benchmark HPC applications and study the impact of virtualization on application placement. We start with the description of our experimental testbed.

3.1 Experimental Testbed

Our experimental testbed consists of an IBM HS-21 Bladecenter with X-series blades. The blades use an Intel Xeon 5148 quad-core processor, with 2.33 GHz core frequency $4MB$ L2 cache. The blades share a single datastore of size 160 GB and 3.4 GB RAM. The blades run VMWare ESX 3.0 hypervisor for virtualization support. There are 10 Virtual Machines on each blade with CPU and Memory reservation tailored towards the purpose of individual experiments. Each VM running Fedora 6 as the guest operating system is allocated a storage of 8 GB.

We use three different workloads for our experimental study. The *daxpy* routine from an HPC suite [5], leads to the highest power consumption amongst all the HPC workloads tested by us and is used as a representative of typical power-hungry HPC workloads. We also use a Linpack benchmark called HPL [1] that has been extensively used in many performance studies. We use HPL in two modes to create two more workloads. The first mode is single-threaded with large problem sizes. In the second mode we use HPL with a 4×2 process grid that solves multiple medium-sized problems. The multi-process HPL uses *mpirun* for managing the multiple processes. We call the two HPL workloads as $HPL1$ and $HPL8$ respectively in this paper.

We investigate the feasibility of applying workload-aware power capping or application placement techniques for power management in HPC applications. Towards this purpose, we experimentally determine if HPC applications have a reasonable dynamic range for a Power Manager to play with. In order to study the applicability of server consolidation for HPC applications, we investigate the impact of multiple HPC applications sharing the same blade and the impact of performance with multiple VMs on the same server. We start with modeling the power drawn by our HPC workloads.

3.2 Power Models for HPC Applications

We have observed resource usage variability on many real workloads (Section. 2). However, platforms need to have power management techniques built in to cause a variation of power drawn with change in workload resource usage parameters. For e.g. IBM Power5 based systems consume a fixed amount of power independent of the workload intensity. However, IBM Power6 and Intel Xeon-based platforms are capable of workload-sensitive power reduction at processor level. We first study the scope of dynamic power range based savings in our testbed setup with change in resource usage.

Fig. 5 captures the power drawn by *daxpy*, $HPL1$ and $HPL8$ at various utilizations. For this experiment, we power-

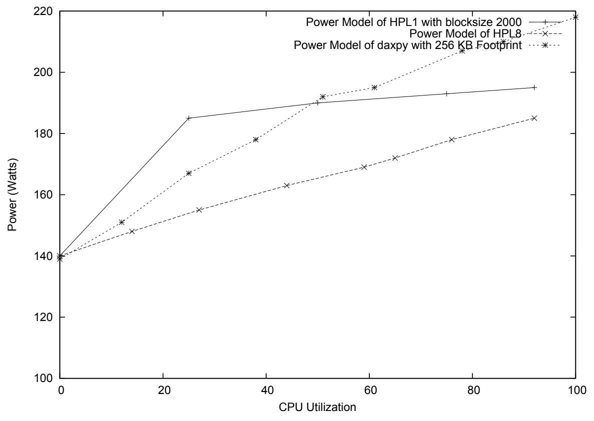

Figure 5: Power drawn with change in utilization for all applications

on only one VM and use the CPU reservation feature to limit the workload to the desired utilization. We observe that the *daxpy* application has a dynamic power range of about $75W$ that can be used by a placement or capping techniques. Further, we observe that the power model of each application is different, with $HPL1$ showing strong non-linearity. This non-linearity can be used for power savings by load unbalancing using a spillage-based workload manager, that loads a server to full utilization before loading any other server. To take an example, if 1.1 load (normalized by server capacity) needs to be placed on 2 servers, a spillage-based workload manager would load the two servers at 1.00 and 0.1 respectively. This would draw a power of $187W$ and $158W$ respectively (total $345W$ by Fig. 5), whereas a load-balanced placement would draw a total of $185 * 2 = 370W$. Similarly, we observe the

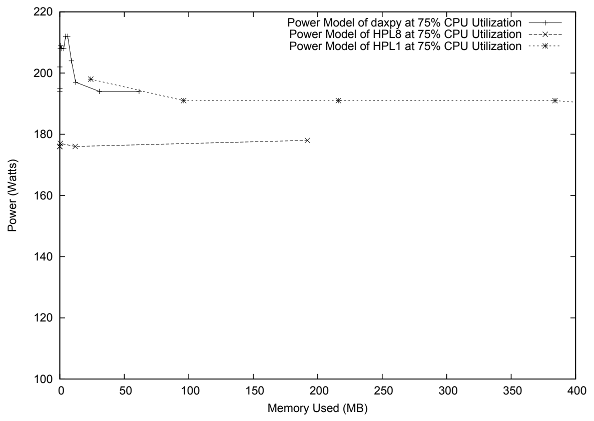

Figure 6: Power drawn with change in memory for all applications

impact of increasing the memory footprint of a workload (by increasing the matrix or array size for the various applications) on the power drawn by a server. Results are captured in Fig. 6. We observe that the problem size of the HPC application does not seem to affect the power drawn by the server. At first glance, the observation seems to contradict a generally accepted belief that power drawn depends on the memory bandwidth drawn by an application. However, we note that the memory footprint does not always determine the memory bandwidth or the number of pages of memory touched by an

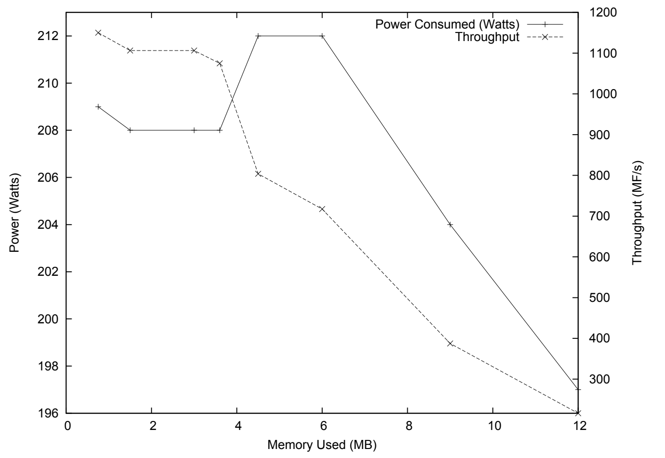

Figure 7: Power and Performance of daxpy with Increased Memory

application per unit time. This is especially true for applications that work on large matrices or arrays where the working set window may be relatively small and independent of the problem size. Hence, even though a large memory footprint application would touch more memory pages in its lifespan, the number of pages touched per second by an application is independent of the footprint. This hypothesis is validated by our study.

We also note that for a linear $O(N)$ application like *daxpy*, the working set is dependent on the size of the array, at least for small to medium array sizes. Hence, as the size of the array increases, an application like *daxpy* may touch more pages of memory and consume more power. We observe this behavior in Fig. 6 for small sized arrays. However, we observe a seemingly strange behaviour at larger memory sizes where the power consumed fell with increase in memory size.

We conjectured that this may be a result of decrease in application throughput as cache size may have become a bottleneck. In order to validate this conjecture, we measure the throughput achieved by *daxpy* application with increase in memory size (Fig. 7). We observe that the throughput decreases as the array size approaches $4MB$. Further, the power drawn also increases at that point and then starts to decrease at an array size of $8MB$. As noted earlier, the L2 cache has a size of $4MB$ and hence, for problem size less than $4MB$, successive iterations are run completely from cache. Hence, the throughput is high but the memory subsystem draws very low power. However, as the array size is increased, more memory pages are touched, initially leading to more power being drawn. However, this also leads to more instructions waiting for memory access, bringing down the throughput. As a result of the decreased throughput, the IDS (instructions dispatched per second) in instruction pipeline is reduced. Since the power drawn by the processor subsystem depends on IDS [8], this leads to a damping effect on the overall power consumed. We observe that the throughput has a very steep fall around $8MB$, and at this point, the decrease in compute power can no longer be compensated by the increase in the power drawn by a memory subsystem. Hence, the overall power drawn by the server decreases with increase in memory footprint.

3.3 Impact of Virtualization on Performnace of Applications

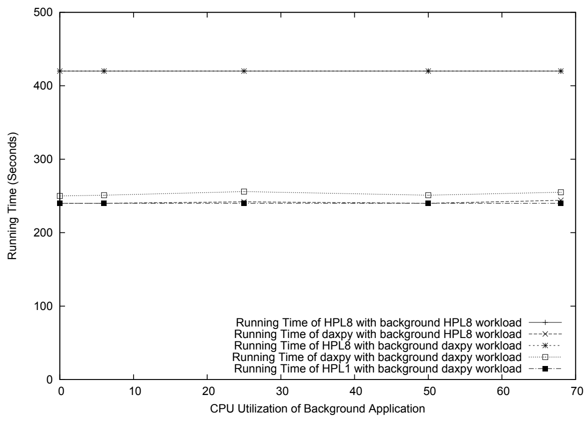

Figure 8: Impact of Small Memory Background Application on Foreground Running Time

Dynamic application consolidation is a technique used to achieve power savings on server clusters. Consolidation is facilitated by dynamic live migration, which requires virtualization support. Further, after consolidation, multiple applications may be hosted on a shared server. Virtualization ensures isolation of applications at processor and memory level only and it is imperative to investigate if an application's performance is not impacted by sharing of other server resources, such as the hardware caches.

If power-aware application placement is used to pack more than one application on a single server, the virtualization platform needs to provide isolation between the multiple VMs. We investigate if such isolation guarantees are workload independent. We use two virtual machines in our study. We designate one of the virtual machines as a container for the background application and another for the foreground application. We study the impact on the foreground application with changes in the characteristics of the background application. Fig. 8 shows the impact on foreground application performance with change in background CPU utilization for a set of foreground and background applications. We observe complete isolation for the set of foreground and background applications considered in Fig. 8.

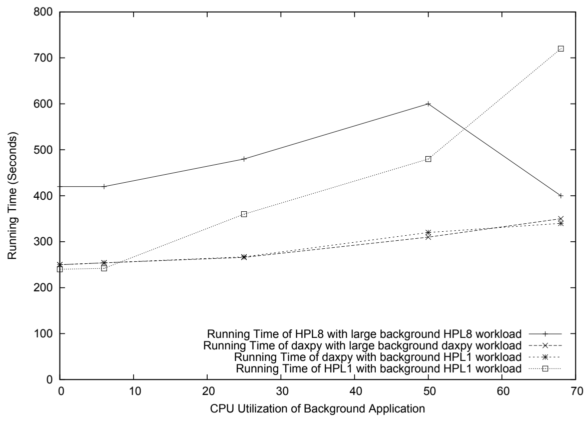

Figure 9: Impact of Large Memory Background Application on Foreground Running Time

However, a similar study for a slightly different selection of foreground and background applications leads to a completely different result. We observe in Fig. 9 that foreground application takes a much longer time to finish as the background CPU utilization is increased.

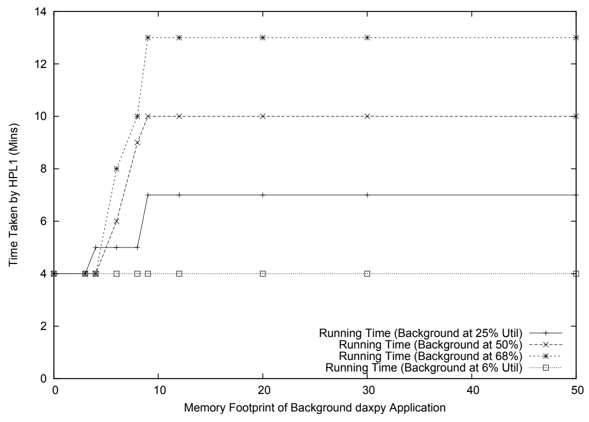

Figure 10: Impact of Change in Memory Footprint of Background Application on HPL1

The key difference between Fig. 8 and Fig. 9 lies in the memory footprint of the background applications. All the scenarios in Fig. 8 capture a background application with small problem size, whereas the background application for the scenarios in Fig. 9 have large problem sizes. Observe that $HPL1$ uses large matrices and is the background for two scenarios. For other scenarios, we use large arrays and matrices for the background *daxpy* and $HPL8$ applications.

The lack of isolation can be attributed to the shared processor cache for which the foreground and background traffic compete with each other. As the background utilization is increased, the throughput of background application also increases. This leads to a higher data rate for the background and consequently, a higher share in the L2 cache. To study this apparent lack of isolation for large memory applications, we conduct further studies with changes in application and memory footprint.

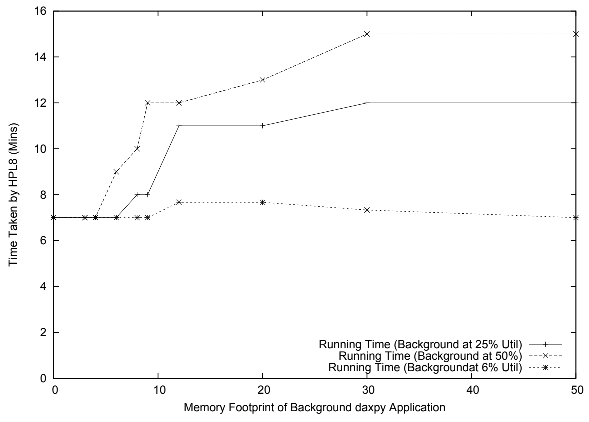

Figure 11: Impact of Change in Memory Footprint of Background daxpy Application on HPL8

In our next experiment, we investigate if large-memory applications are also affected by background applications. We experiment with $HPL1$ as the foreground application and *daxpy* as the background application and study the performance of $HPL1$ with varying array size of *daxpy*. We observe, in Fig. 10, that even large applications are impacted by background applications with large footprint. Hence, the time taken for the $HPL1$ job does not change with change in either foreground utilization or foreground problem size at small problem sizes. However, as the size of the background *daxpy* application approaches around $4MB$, the foreground performance starts to suffer with increase in both background utilization and problem size. Another interesting observation is that the impact of background memory footprint stabilizes beyond $10MB$ and any further increase in background problem size has no effect on the performance of the foreground application.

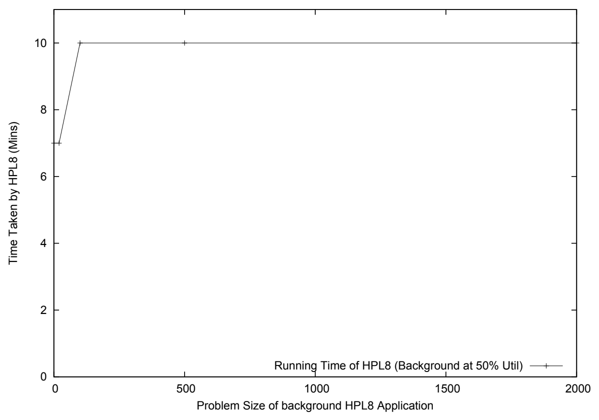

Figure 12: Impact of Change in Memory Footprint of Background HPL8 Application on HPL8

We conduct similar experiments to study the impact of change in memory footprint of the background application on small memory applications. We observe a similar impact on these applications as well. A striking similarity in all the three experiments (Fig. 10,11,12) is the nature of the curves, which is a step function. The nature of the curves do not change with change in either the foreground or the background application. Further, the step starts at approximately the same point in all the curves.

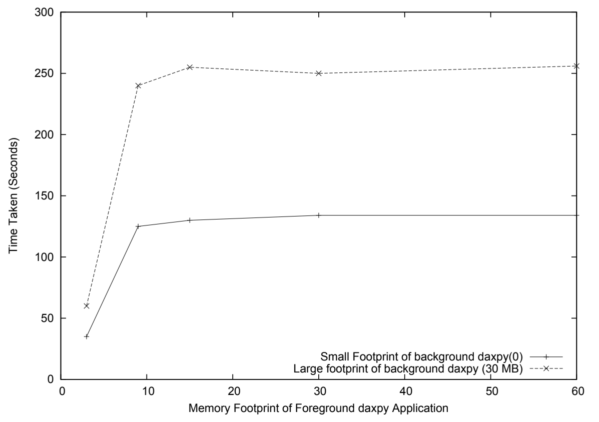

Figure 13: Impact of Change in Memory Footprint of Background daxpy Application on daxpy application

The above observations led us to hypothesize that all applications are impacted by background workload, as soon as the memory footprint of the background application exceeds a specific value. This is true irrespective of whether the foreground application has a small footprint or a large footprint. To verify this conjecture, we conduct experiments for different sizes of the memory footprint of the foreground application. Fig. 13 compares the impact of background memory footprint on foreground *daxpy* application for two different foreground memory footprints. In one case, the foreground *daxpy* uses a very small array, whereas in the second the array size was $30MB$. We observe a striking similarity between the two curves as both are step functions. The value of the step sizes are different for the two cases, with a smaller memory application executing the same number of operations in a much smaller time.

The results in Fig. 13 combined with our earlier observation on memory impacting performance (Fig. 7) signify the importance of overall memory footprint for the performance of the various applications. We surmise that memory footprint impacts performance, independent of the number of applications that are run concurrently. Hence, we conduct experiments to study the impact of increase in size of memory footprint for the same application. The results, captured in Fig. 14, validate our assertion as the step nature of the curve is exhibited even when a single application is run. Hence, the impact of other applications on the performance of a given application is similar to the impact of increase in the memory footprint of the application.

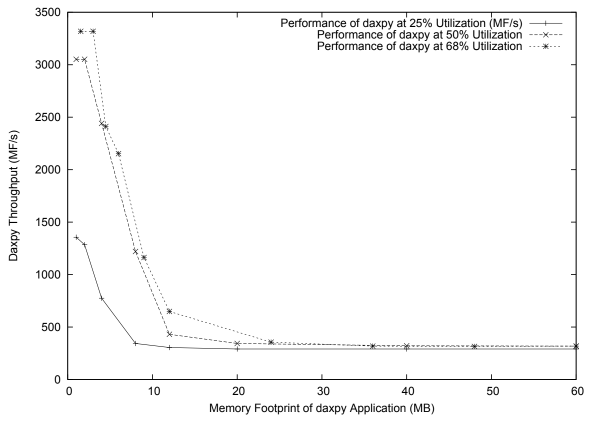

Figure 14: Impact of Change in Memory Footprint on daxpy application

The above observation leads to an important insight that needs to be considered while placing HPC applications on power-managed servers. The performance of small memory applications will not be impacted as long as they are co-located with small memory applications and large memory applications (say, larger than $60MB$) will not see an impact in performance even if they are co-located with other applications. Hence, isolation on virtualized platforms works for homogeneous workloads (in terms of size) as long as they are very small or very large. Hence, if a workload mix has all applications with small memory footprint then they would be isolated. Similarly, if all applications have very large memory footprints then they would also be isolated. We use this insight in designing placement recommendations in Section 4.

3.4 Virtualization Overhead

Virtualization allows dynamic server consolidation with workload-dependent isolation as evident from our experiments. However, virtualization leads to VMM traps on privileged instructions. Further, running multiple virtual machines on the same server leads to fragmentation. Both may lead to degraded performance that HPC applications may not tolerate. Huang et al. [12] show that the performance impact of virtualization can be significantly reduced by using a mechanism called VMM-bypass [13]. They show that HPC applications running within a virtual machine can achieve near-native performance, thus making a strong case for running HPC applications in a virtualized environment.

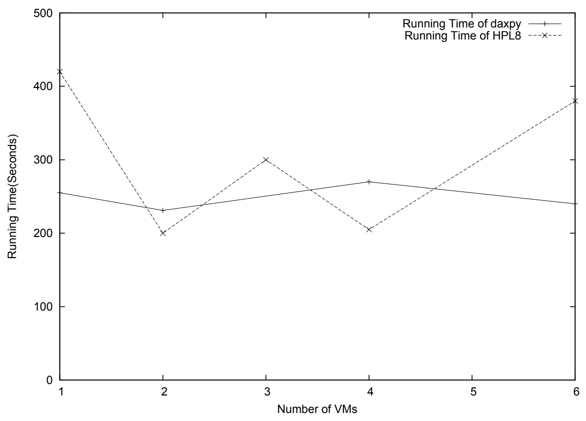

Figure 15: Virtualization Overhead for HPL8 and daxpy

During dynamic consolidation, a physical server may run a large number of VMs, which leads to fragmentation of memory. In order to study if the fragmentation results in degraded performance on a server running multiple VMs, we design an experiment that computes a fixed number of operations. We then run the workload on a single VM and observe the performance. In subsequent runs, we power on more virtual machines on the server and equally distribute the workload amongst multiple VMs. We observe (Fig. 15) that the performance of the workload does not degrade with increasing number of VMs, indicating that the effects of fragmentation are not severe. Hence, even though the performance changes with the number of VMs, there is no visible degradation. We do observe that performance is better when the number of VMs is a power of 2. We attribute this performance benefit to the fact that memory banks as well as cache segments are typically power of 2. Hence, their division amongst 2^i number of VMs leads to minimal fragmentation, and hence improved performance.

Our experiments conclude that a placement controller for a cluster running HPC applications can take advantage of power and virtualization features in the platform without an adverse impact on performance. However, as opposed to our earlier work on placement controller [30] that places VMs on servers based on CPU utilization only, our experiments with the HPC applications underline the importance of taking other factors like cache size and memory footprint into account during dynamic placement.

4. POWER SAVING ARCHITECTURE AND METHODOLOGY

We now present a framework and methodology for power management of HPC applications.

4.1 Methodology

In this subsection, we describe the methodology used to place virtual machines and corresponding workloads onto physical servers. In *pMapper* [29] the placement algorithms assumed that the performance of the workloads can be characterized only by CPU utilization. However, the experimental results on the benchmark applications demonstrate that the memory usage and the working set size also determine the performance. We observe that very small applications (in terms of working set) do not affect the performance of other applications. Also, we observe that very large applications are not affected by other workloads running on the same physical machine.

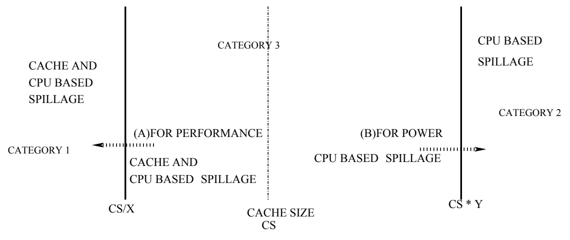

Figure 16: Operating regions

Hence, instead of adopting CPU spillage based allocation, we adopt a placement methodology where CPU and working set size are both taken into account. We define three clear operating regions as shown in Figure 16. Small applications (category 1) are placed in a manner such that the total working set size of the applications is smaller than the physical machine's CPU cache size. These applications will degrade in performance if they are packed with larger applications because of thrashing in the CPU cache. Category 2 applications have a very large working set size whose performance is not impacted by the CPU cache, and they are packed separately. In between there is the third set of applications (category 3) whose working set does not entirely fit into the cache and they will definitely be impacted by other applications on the same machine. For this class of applications, we have a choice of conserving power or maximizing performance.

Our proposed method can thus be characterized by the following policies:

1. Sort all applications in ascending order by their working set size. We assume that working set sizes are provided by the applications, which is the case for most HPC applications.
2. Category 1 applications are packed such that the total working set size of all applications packed on a server does not exceed the cache size of the server. Also, the CPU limit is never violated.
3. Category 2 applications are only packed based on CPU limit without regard to application working set size.
4. Category 3 applications can either be placed in the manner of Category 1 application (for no performance degradation) or in the manner of Category 1 applications (for maximum power savings).

In Figure 16, the boundary between the application category 1 and 2 is parameterized by X ($0 < X \leq 1$), such that X may be used to trade-off resource wastage with performance protection. We can set a high value of X to protect more applications from any performance degradation or set it close to 0 and maximize power savings. A lower value of X leads to more performance impact and less resource wastage. Higher X leads to less use of virtualization but performance protection for more applications. The value of Y is determined experimentally by increasing the footprint of applications till we find isolation.

4.2 Architecture

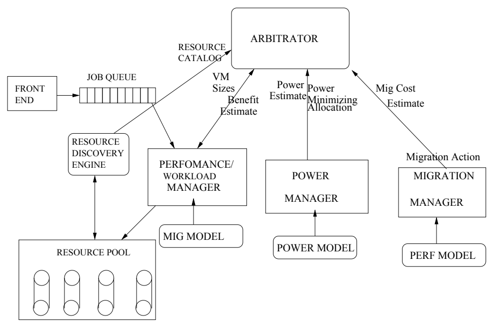

Figure 17: Architecture of performance, power, and migration aware cluster management using virtualized servers.

We now describe an architecture (Figure 17) to implement the power-savings methodology described earlier. Our architecture is based on the *pMapper* controller described in a earlier paper [29]. As in [29], the *performance manager* is used to size the VMs and provide an estimate of the benefit of a configuration given the size of the VMs. The *power manager* supports interfaces to provide power estimates for a given VM size assuming full utilization. The *power manager* also provides a power minimizing allocation for a given set of VM sizes. The *migration manager* gives the cost estimate of a migration and executes a migration, when instructed, by interfacing with the specific virtualization layer mechanisms. The placement controller intelligence lies in the *arbitrator* which explores the entire configuration space of VM sizes and locations to pick the optimal combination.

Our new architecture extends *pMapper* by the following enhancements. We have a job queue to store long-running HPC jobs, which are also scheduled by the *performance cum workload manager*. Hence, the performance manager is also entrusted with the job of scheduling HPC jobs. The second key difference lies in the characterization of the VMs. In the *pMapper* architecture, a VM was characterized only by its CPU utilization (or size). However, our methodology also uses the working set window of applications running on a VM to determine its placement. Hence, a VM is characterized by CPU, as well as memory and cache requirements. Finally, we need to take into account characteristics of the resource pool (CPU, cache sizes, memory sizes etc), which are fed as a resource catalog information to the *arbitrator*.

We have implemented the power management methodology in (i) a runtime application placement middleware that minimizes the power consumed while meeting the applications and uses IBM Enterprise Workload Manager for estimating VM sizes and (ii) an Estimation tool that simulates the placement of applications on servers to minimize the static and dynamic costs and present estimates of the cost savings.

4.3 Example Study

We now present an example case study to illustrate our proposed power management methodology. We use an HPC ap-

App Name	Type	CPU Reqd	Memory Reqd
DAX1	daxpy	15	0.1 MB
DAX2	daxpy	20	0.4 MB
DAX3	daxpy	20	1 MB
DAX4	daxpy	25	1 MB
HP1	HPL1	25	2 MB
HP2	HPL1	25	25 MB
HP3	HPL1	25	25 MB
HP4	HPL1	50	40 MB
HP5	HPL1	50	40 MB
HP6	HPL8	20	4MB
HP7	HPL8	50	4MB

Table 1: Input Applications with Characteristics

plication set with characteristics as described in Table. 1. The set consists of 11 applications with 4 daxpy and 7 Linpack applications, each with varying memory and CPU requirement. The applications need to be packed on a server farm with 11 identical servers, having a cache size of $4MB$.

Server	Apps	CPU	Memory
1	DAX1, DAX2, DAX3, HP6, DAX4	100	6.5 MB
2	HP1, HP2, HP3	75	90 MB
3	HP4, HP5	100	80MB
4	HP7	50	4MB

Table 2: Output allocation by CPU-only Packing

We first use a CPU-spillage based packing (Table 2) that does not take into account the working set sizes. The placement is able to pack the applications on 4 servers, which is the minimal number of servers. Hence, the packing methodology is able to substantially reduce the power consumption (from 11 active servers to 4 active servers). However, the above packing may have huge performance overhead for a large number of VMs.

Server	Apps	CPU	Memory
1	DAX1, DAX2, DAX3, DAX4	80	2.5 MB
2	HP4, HP5	100	90MB
3	HP2, HP3	50	90MB
4	HP1, HP6, HP7	95	10 MB

Table 3: Proposed Output allocation

We next present the packing obtained by our proposed methodology (Table. 3). We use X as 2 in our case study. We label all the *daxpy* applications as Category 1 and pack them on the same server. Similarly, HP2, HP3, HP4, and HP5 are labeled as Category 2 applications and packed by CPU-based spillage only. The remaining applications are labeled as Category 3 and we use a power-minimizing strategy to pack them on as few servers as possible. We observe that we are able to pack the applications on the minimal 4 servers, using our methodology as well.

We now study the performance impact on the applications by consolidating them using virtualization. We observe that our methodology leads to performance impact only for Category 3 applications, whereas existing cache-oblivious strategies lead to performance impact for more than half of the applications. Hence, cache-obliviousness can lead to a huge performance penalty while minimizing power. On the other hand, we can quantify the performance impact by being aware of the cache size. Further, we can use a higher value of X to protect

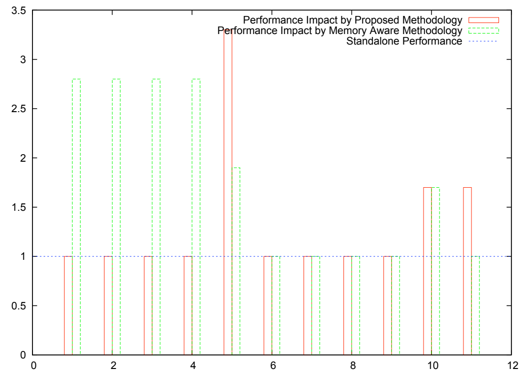

Figure 18: Performance Impact due to server consolidation for Proposed Methodology and Power-Minimizing Methodology

even more applications, if required. Our case study thus underlines the strength and flexibility of our proposed cache and CPU-aware methodology versus cache-oblivious strategies.

5. RELATED WORK AND SUMMARY

Power and migration-aware application placement in virtualized server clusters was first addressed in our earlier work called *pMapper* [29]. In this paper, we focus on HPC benchmark workloads and show how the nature of power models and impact of virtualization can be used by the placement controller logic. It is shown that *pMapper* needs to use not just CPU but memory and cache usage characteristics also for placement of HPC applications.

The viability of HPC applications hosted in virtual machine containers in tightly coupled systems [12, 18] or in loosely coupled grid environments [9] have been proposed earlier. However, this is the first attempt at investigating the feasibility of power management in tightly-coupled virtualized server clusters in the context of HPC workloads. A number of papers have investigated characterization of applications in terms of power and resource consumption. Felter et al. create power models of applications based on CPU and memory usage [8]. There is evidence to show that voltage and frequency setting, that minimizes energy consumption, is dependent on system characteristics and application-specific usage of CPU and memory resources [28, 25]

Resource managers in virtualized environments attempt to either load balance a cluster [27] or only take performance SLAs into consideration [2] when allocating virtual machine resources and during dynamic placement of virtual machines on physical servers using migration facilities. Except *pMapper*, there are no power and migration cost aware cluster resource managers that work in a virtualized environment. However, the area of power and performance trade-off has been addressed extensively. Muse [4] uses an economic model to perform power-aware resource allocation in a cluster. Kephart et al. [14] have also implemented a controller architecture, similar to ours, in which one can specify power-performance objectives. VirtualPower [19] introduces soft power states on top of the Xen Hypervisor to capture application specific power-performance policies.

A large body of work in power management at cluster resource manager level actually addresses the problem of request distribution, where policies can be implemented in a front-end load balancer to meet performance objectives while minimizing the power cost [3, 4, 10, 22, 24, 26]. Cooperative voltage scaling techniques have been also proposed in tightly coupled clusters [6, 11].

In summary, in this paper we make a case for power management in HPC clusters using platform virtualization, working with real workload traces. We investigate aspects of modeling the power consumption of such applications and the impact of platform virtualization. Finally, we describe a power-aware application placement controller based on these models. In future, we need to refine the power models further for various workload types and consider more parameters that may affect power-aware placement in virtualized platforms. We need to build on *pMapper* algorithms to use such power models.

6. REFERENCES

[1] Hpl- a portable implementation of the high performance linpack benchmark for distributed memory computers. In *http://www.netlib.org/benchmark/hpl/*.

[2] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. Dynamic placement of virtual machines for managing sla violations. In *IEEE Conf. Integrated Network Management*, 2007.

[3] J. Chase and R. Doyle. Balance of Power: Energy Management for Server Clusters. Proc. HotOS, 2001.

[4] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centers. In *Proc. of ACM SOSP*, 2001.

[5] DAXPY. http://www.netlib.org/blas/daxpy.f.

[6] E. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. In *Proc. of Workshop on Power-Aware Computing Systems.*, 2002.

[7] M.S. Floyd et al. System power management support for ibm power6 microprocessor. *IBM Journal of Research and Development*, 51(6):733–746, 2007.

[8] Wes Felter, Karthick Rajamani, Tom Keller, and Cosmin Rusu. A performance-conserving approach for reducing peak power consumption in server systems. In *Proc. of International Conference on Supercomputing*, 2005.

[9] Renato J. Figueiredo, Peter A. Dinda, and José A. B. Fortes. A case for grid computing on virtual machines. In *Proceedings of IEEE ICDCS*, 2003.

[10] Taliver Heath, Bruno Diniz, Enrique V. Carrera, Wagner Meira Jr., and Ricardo Bianchini. Energy conservation in heterogeneous server clusters. In *Proc. of ACM PPoPP*, 2005.

[11] Tibor Horvath. Dynamic voltage scaling in multitier web servers with end-to-end delay control. *IEEE Trans. Comput.*, 56(4):444–458, 2007. Member-Tarek Abdelzaher and Senior Member-Kevin Skadron and Member-Xue Liu.

[12] Wei Huang, Jiuxing Liu, Bulent Abali, and Dhabaleswar K. Panda. A case for high performance computing with virtual machines. In *Proc. of the International Conference on Supercomputing*, pages 125–134, New York, NY, USA, 2006. ACM.

[13] B. Abali J. Liu, W. Huang and D.K.Panda. High performance vmm-bypass i/o in virtual machines. In *Proc. of Usenix Annual Technical Conference*, 2006.

[14] Jeffrey O. Kephart, Hoi Chan, Rajarshi Das, David W. Levine, Gerald Tesauro, Freeman Rawson, and Charles Lefurgy. Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs. In *Proc. of ICAC*, page 24, Washington, DC, USA, 2007. IEEE Computer Society.

[15] Charles Lefurgy, Karthick Rajamani, Freeman Rawson, Wes Felter, Michael Kistler, and Tom W. Keller. Energy management for commercial servers. *Computer*, 36(12):39–48, 2003.

[16] The OSC Linux Cluster log. http://www.cs.huji.ac.il/labs/parallel/ workload/l_osc/index.html.

[17] LPC EGEE Cluster Logs. http://www.cs.huji.ac.il/labs/parallel/ workload/l_lpc/index.html.

[18] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott. Proactive fault tolerance for hpc with xen virtualization. In *Proc. of International Conference on Supercomputing*, 2007.

[19] Ripal Nathuji and Karsten Schwan. Virtualpower: coordinated power management in virtualized enterprise systems. In *Proc. ACM SOSP*, 2007.

[20] Logs of Real Parallel Workloads from Production Systems. http://www.cs.huji.ac.il/labs/parallel/workload/logs.html.

[21] D. Oppenheimer, D. A. Patterson, , and A. Vahdat. A case for informed service placement in planet lab. In *Planetlab Technical Report, PDN-04-025*, 2004.

[22] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems, 2001.

[23] Control power and cooling for data center efficiency HP thermal logic technology. An HP Bladesystem innovation primer. http://h71028.www7.hp.com/erc/downloads/4aa0-5820enw.pdf.

[24] K. Rajamani and C. Lefurgy. On evaluating request-distribution schemes for saving energy in server clusters, 2003.

[25] Karthick Rajamani, Heather Hanson, Juan Rubio, Soraya Ghiasi, and Freeman L. Rawson III. Application-aware power management. In *IISWC*, pages 39–48, 2006.

[26] Cosmin Rusu, Alexandre Ferreira, Claudio Scordino, and Aaron Watson. Energy-efficient real-time heterogeneous server clusters. In *Proc. IEEE RTAS*, 2006.

[27] VMWare Distributed Resource Scheduler. http://www.vmware.com/products/vi/vc/drs.html.

[28] David C. Snowdon, Sergio Ruocco, and Gernot Heiser. Power management and dynamic voltage scaling: Myths and facts, September 2005.

[29] A. Verma, P. Ahuja, and A. Neogi. pmapper: Power and migration cost aware application placement in virtualized systems. In *IBM Research Report RI07010 (also under review)*, 2007.

[30] A. Verma and A. Anand. On store placement for respones time minimization in parallel disks. In *Proc. of IEEE ICDCS*, 2006.

[31] Intel Xeon. http://www.intel.com/products/processor/xeon5000/.