An Integrated Approach to Resource Pool Management: Policies, Efficiency and Quality Metrics

Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, Guillaume Belrose, Tom Turicchi, and Alfons Kemper
HP Laboratories
HPL-2008-89

Keyword(s):
automation, enterprise applications, shared resource pools measurements, capacity management, workload placement and migration controllers, performance models, workload analysis

Abstract:
The consolidation of multiple servers and their workloads aims to minimize the number of servers needed thereby enabling the efficient use of server and power resources. At the same time, applications participating in consolidation scenarios often have specific quality of service requirements that need to be supported. To evaluate which workloads can be consolidated to which servers we employ a trace-based approach that determines a near optimal workload placement that provides specific qualities of service. However, the chosen workload placement is based on past demands that may not perfectly predict future demands. To further improve efficiency and application quality of service we apply the trace-based technique repeatedly, as a workload placement controller. We integrate the workload placement controller with a reactive controller that observes current behavior to i) migrate workloads off of overloaded servers and ii) free and shut down lightly-loaded servers.

To evaluate the effectiveness of the approach, we developed a new host load emulation environment that simulates different management policies in a time effective manner. A case study involving three months of data for 138 SAP applications compares our integrated controller approach with the use of each controller separately. The study considers trade-offs between i) required capacity and power usage, ii) resource access quality of service for CPU and memory resources, and iii) the number of migrations. We consider two typical enterprise environments: blade and server based resource pool infrastructures. The results show that the integrated controller approach outperforms the use of either controller separately for the enterprise application workloads in our study. We show the influence of the blade and server pool infrastructures on the effectiveness of the manageme nt policies.

External Posting Date: July 6, 2008 [Fulltext] Approved for External Publication

Internal Posting Date: July 6, 2008 [Fulltext]

Published in the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Netwroks, DSN'2008, June 24-27,

An Integrated Approach to Resource Pool Management: Policies, Efficiency and Quality Metrics

Daniel Gmach*, Jerry Rolia†, Ludmila Cherkasova†, Guillaume Belrose‡, Tom Turicchi**, and Alfons Kemper*

* Technische Universität München
85748 Garching, Germany
firstname.lastname@in.tum.de

Hewlett-Packard Laboratories
† Palo Alto, CA, USA, ‡Bristol, GB
firstname.lastname@hp.com

** Hewlett-Packard
ESS Software, Richardson, TX, USA
firstname.lastname@hp.com

Abstract

The consolidation of multiple servers and their workloads aims to minimize the number of servers needed thereby enabling the efficient use of server and power resources. At the same time, applications participating in consolidation scenarios often have specific quality of service requirements that need to be supported. To evaluate which workloads can be consolidated to which servers we employ a trace-based approach that determines a near optimal workload placement that provides specific qualities of service. However, the chosen workload placement is based on past demands that may not perfectly predict future demands. To further improve efficiency and application quality of service we apply the trace-based technique repeatedly, as a workload placement controller. We integrate the workload placement controller with a reactive controller that observes current behavior to i) migrate workloads off of overloaded servers and ii) free and shut down lightly-loaded servers.

To evaluate the effectiveness of the approach, we developed a new host load emulation environment that simulates different management policies in a time effective manner. A case study involving three months of data for 138 SAP applications compares our integrated controller approach with the use of each controller separately. The study considers trade-offs between i) required capacity and power usage, ii) resource access quality of service for CPU and memory resources, and iii) the number of migrations. We consider two typical enterprise environments: blade and server based resource pool infrastructures. The results show that the integrated controller approach outperforms the use of either controller separately for the enterprise application workloads in our study. We show the influence of the blade and server pool infrastructures on the effectiveness of the management policies.

1 Introduction

Virtualization is gaining popularity in enterprise environments as a software-based solution for building shared hardware infrastructures. Forrester Research estimates that businesses generally end up using somewhere between 8 and 20 percent of the server capacity they have purchased. Virtualization technology helps to achieve greater system utilization while lowering total cost of ownership and responding more effectively to changing business conditions. For large enterprises, virtualization offers an ideal solution for server and application consolidation in an on-demand utility.

The consolidation of multiple servers and their workloads has an objective of minimizing the number of resources, e. g., computer servers, needed to support the workloads. In addition to reducing costs, this can also lead to lower peak and average power requirements. Lowering peak power usage may be important in some data centers if peak power cannot easily be increased.

Applications participating in consolidation scenarios can make complex demands on servers. For example, many enterprise applications operate continuously, have unique time-varying demands, and have performance-oriented Quality of Service (QoS) objectives. To evaluate which workloads can be consolidated to which servers, some preliminary performance and workload analysis should be done. In the simple naive case, a data center operator may estimate the peak resource requirements of each workload and then evaluate the combined resource requirements of a group of workloads by using the sum of their peak demands. However, such an approach can lead to significant resource over-provisioning since it does not take into account the benefits of resource sharing for complementary workload patterns. In this work, to evaluate which workloads can be consolidated to which servers we employ a trace-based approach [21] that assesses permutations and combinations of workloads in order to determine a near optimal workload placement that provides specific qualities of service.

The general idea behind trace-based methods is that historic traces that capture past application demands are representative of the future application behavior. In our past work, we assumed that the placement of workloads would be adjusted infrequently, e. g., weekly or monthly [21]. However, by repeatedly applying the method at shorter timescales we can achieve further reductions in required capacity. In this scenario, we treat the trace-based approach as a workload placement controller that periodically causes workloads to migrate among servers to consolidate them while satisfying quality requirements. Such migrations [6] are possible without interrupting the execution of the corresponding applications. We enhance our optimization algorithm to better support this scenario by minimizing migrations during successive control intervals.

Though enterprise application workloads often have time varying loads that behave according to patterns [23][8], actual demands are statistical in nature and are likely to differ from predictions. Therefore, to further improve the efficiency and application quality of service of our approach, we manage workloads by integrating the workload placement controller with a reactive workload migration controller that observes current behavior to i) migrate workloads off of overloaded

servers and ii) free and shut down lightly-loaded servers.

There are many management policies that can be used to guide workload management. Each has its own parameters. However, predicting and comparing the long term impact of different management policies for realistic workloads is a challenging task. Typically, this process is very time consuming as it is mainly done following a risky "trial and error" process either with live workloads and real hardware or with synthetic workloads and real hardware. Furthermore, the effectiveness of policies may interact with the architecture for the resource pool of servers so it must be repeated for different alternatives.

To better assess the long term impact of management policies we introduce a new host load emulation environment. The emulation environment: models the placement of workloads on servers; simulates the competition for resources on servers; causes the controllers to execute according to a management policy; and dynamically adjusts the placement of workloads on servers. During this simulation process the emulator collets metrics that are used to compare the effectiveness of the policies.

A case study involving three months of data for 138 SAP applications is used to evaluate the effectiveness of several management policies. These include the use of the workload placement and workload migration controllers separately and in an integrated manner. The study considers trade-offs between i) required capacity and power usage, ii) resource access quality of service for CPU and memory resources, and iii) the number of migrations. We consider two typical enterprise environments: blade and server based resource pool infrastructures. The results show that our integrated workload management method outperforms each approach separately. Our results show that the integration of controllers outperforms the use of either controller separately.

For the blade pool, the integrated controllers offered CPU quality that was 20% better than for either controller separately, while using only 11% percent more capacity than the corresponding ideal case where future workloads were known in advance. For the server resource pool, the hourly CPU quality penalty was nearly 7 times better than either controller separately, while using 18% percent more capacity than the corresponding ideal case. Because the blades were memory bound for the workloads in our study, they had lower CPU quality penalties per hour. For the workloads under study, the blade based resource pool uses only 2/3 the peak power of the server based resource pool and about 30% less power in total with only a slightly higher cost. Finally, for the blade and server pool infrastructures and workloads of our case study, the management policies we proposed achieved average CPU utilizations of approximately 42% and 66%, respectively.

The rest of this paper is organized as follows. Section 2 describes the workload placement and migration controllers, management policies, and metrics. The host load emulation environment is introduced in Section 3. Section 4 presents case study results. Section 5 describes related work. Finally, conclusions are offered in Section 6.

2 Management Services, Policies, and Quality Metrics

Our management policies rely on two controllers. This section describes the workload placement controller and the reactive workload migration controller. The management policies exploit these controllers in several different ways. Quality metrics are used to assess the effectiveness of management policies.

2.1 Workload Placement Controller

The workload placement controller has two components.

- *A simulator component* emulates the assignment of several application workloads on a single server. It traverses the per-workload time varying traces of historical demand to determine the peak of the aggregate demand for the combined workloads. If for each capacity attribute, e. g., CPU and memory, the peak demand is less than the capacity of the attribute for the server then the workloads fit on the server.
- *An optimizing search component* examines many alternative placements of workloads on servers and reports the best solution found. The optimizing search is based on a genetic algorithm [11].

The workload placement controller is based on the Capman tool that is described further in [21]. It supports both consolidation and load leveling exercises. Load leveling balances workloads across a set of resources to reduce the likelihood of service level violations. Capman supports the controlled overbooking of capacity that computes a required capacity for workloads on a server that may be less than the peak of aggregate demand. It is capable of supporting a different quality of service for each workload [5]. Without loss of generality, this paper considers the highest quality of service which corresponds to a required capacity for workloads on a server that is the peak of their aggregate demand.

For this paper, we enhance Capman to better support the workload placement controller paradigm. The enhancement exploits multi-objective functionality offered by the genetic algorithm approach [1]. Instead of simply finding the smallest number of servers needed to support a set of workloads, Capman now also evaluates solutions according to a second simultaneous objective. The second objective aims to minimize the number of changes to workload placement. When invoking Capman an additional parameter specifies a target t for the number of workloads that it is desirable to migrate. Limiting the number of migrations limits migration overheads and reduces the risk of incurring a migration failure. If it is possible to find a solution with fewer than t migrations, then Capman reports the workload placement that needs the smallest number of servers and has t or fewer migrations. If more changes are needed to find a solution, then Capman reports a solution that has the smallest number of changes to find a feasible solution.

2.2 Workload Migration Controller

The migration controller is a fuzzy-logic based feedback control loop. An advisor module of the controller continuously monitors the servers' resource utilization and triggers a fuzzy-logic based controller whenever resource utilization values are too low or too high. When the advisor detects a lightly utilized, i. e., *underload situation*, or *overload situation* the fuzzy controller module identifies appropriate actions to remedy the situation. For this purpose, it is initialized with information on the current load situation of all affected servers and workloads and determines an appropriate action. For example, as a first step, if a server is overloaded it determines a workload on the server

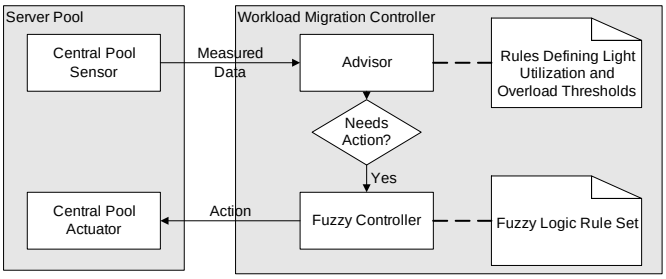

Figure 1. Architecture of the Workload Migration Controller

that should be migrated and as a second step it searches for a new server to receive the workload. Furthermore, these rules initiate the shutdown and startup of nodes. The architecture of the workload migration controller is illustrated in Figure 1.

The implementation of the workload migration controller uses the following rules. A server is defined as overloaded if its CPU or memory consumption exceed a given threshold. Furthermore, an underload situation occurs whenever the average CPU and memory usage of all servers in the system drop below a specified threshold. A fuzzy controller module identifies overload and underload situations. It implements two separate fuzzy controllers for each of these situations.

In an overload situation, the first fuzzy controller determines a workload to migrate away and the second controller chooses an appropriate target server. The target server is the least loaded server that has sufficient resources to host the workload. If such a server doesn't exist we start up a new server and migrate the workload to the new one.

In an underload situation, the first fuzzy controller chooses the least loaded server and tries to shut it down. For every workload on this server, the second fuzzy controller determines a target server. If a target cannot be found for a workload then the shutdown process is stopped. In contrast to overload situations, this controller does not ignite additional servers.

Section 4.3 shows the impact of various combinations of threshold values for overload and underload management. A more complete description of the fuzzy controller and its rules are presented in [23].

2.3 Policies

Our study considers the following management policies:

- *MC* – migration controller alone;
- *React* – placement controller with a reactive policy;
- *React%20HR, React+2S* – placement controller with a reactive policy and headroom;
- *Hist* – placement controller with a historical policy;
- *Hist+MC* – integrated workload placement controller with a historical policy and the workload migration controller.

The *MC* policy corresponds to using the workload migration controller alone for on-going management. The workload placement controller causes an initial workload placement that consolidates workloads to a small number of servers. The workload migration controller is then used to periodically migrate workloads to alleviate overload and underload situations. The workload migration controller operates at the time scale that measurement data is made available. In this paper, we assume the migration controller is invoked every 5 minutes.

The *React* policy employs the workload placement controller alone. The controller is invoked once per control interval. However, we assume the control interval is larger than for the migration controller, e. g., 4 hours instead of 5 minutes. The controller uses recent workload demand trace information from the current control interval to compute a more effective workload placement for the next control interval.

React%20HR and *React+2S* are similar to the React policy. However, *React%20HR* leaves 20% of the CPU capacity unused on each server to improve quality. We refer to the 20% as headroom. The *React+2S* policy is similar to *React* but after consolidation two servers are added and the workloads balanced across all the servers.

With the *Hist* policy, the workload placement controller uses historical workload demand trace information from the previous week that corresponds to the next control interval to compute a more effective workload placement for the next control interval. The historical mode is more likely appropriate for workloads that have repetitive patterns for workload demands.

Finally, we consider an integration of the placement and migration controllers. We refer to this as the *Hist+MC* policy.

2.4 Efficiency and Quality Metrics

To compare the long term impact of management policies we consider several metrics. These include:

- total server CPU hours used and server CPU idle hours used;
- normalized server CPU hours used and normalized server idle CPU hours;
- minimum and maximum number of servers;
- the distribution of power usage in Watts;
- CPU and memory resource access quality per hour; and,
- the number of migrations per 4 hours[1].

The total server CPU hours used corresponds to the sum of the per workload demands. Total server idle CPU hours is the sum of idle CPU hours for servers that have workloads assigned to them. The server idle CPU hours shows how much CPU capacity is not used on the active servers. Normalized values are defined with respect to the total demand of the workloads as specified in the workload demand traces. Note that if normalized server CPU hours used is equal to 1 and normalized server CPU hours idle are equal to 1.5 then this corresponds to an average CPU utilization of 40%.

The minimum and maximum numbers of servers for a policy are used to compare the overall impact of a management policy on capacity needed for server infrastructure. This determines the cost of the infrastructure.

Each server has a minimum power usage p_{idle}, in Watts, that corresponds to the server having idle CPUs, and a maximum power usage p_{busy} that corresponds to 100% CPU utilization. The power used by a server is estimated as

$$p_{idle} + u \cdot (p_{busy} - p_{idle})$$

where u is the CPU utilization of the server [7].

We introduce a new quality metric that is based on the number of successive intervals where a workload's demands are not satisfied. Longer epochs of unsatisfied requirements incur greater penalty values as they are more likely to be perceived

[1] We explain the choice of the 4 hours later in this subsection.

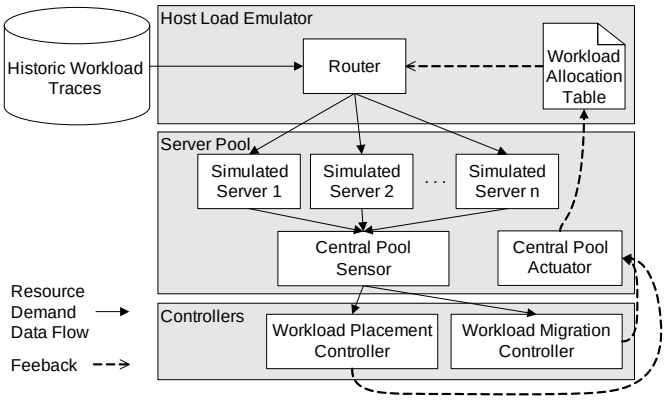

Figure 2. Architecture of the Host Load Emulator

by those using applications. We define a penalty value for i successive overloaded measurement intervals as i^2. The sum of these penalty values over all workloads is the quality value for the metric. We use this approach for both CPU and memory. We divide the total quality value for an experiment by the number of simulated hours to facilitate interpretation.

Finally, the number of migrations is the sum of migrations caused by the workload placement and workload migration controllers. A smaller number of migrations is preferable as it offers lower migration overheads and a lower risk of migration failures. We divide the total number of migrations for an experiment by the number of 4 hour intervals in the experiment to facilitate interpretation. Four hours is used because that is the chosen control interval for the workload placement controller in our case study.

3 Host Load Emulator

Predicting the long term impact of integrated management policies for realistic workloads is a challenging task. We employ a flexible host load emulation environment to evaluate many management policies for resource pools in a time effective manner.

The architecture of the host load emulation environment is illustrated in Figure 2. The emulator takes as input historical workload demand traces, an initial workload placement, server resource capacity descriptions, and a management policy. The server descriptions include numbers of processors, processor speeds, real memory size, and network bandwidth. A routing table directs each workload's historical time varying resource requirement data to the appropriate simulated server. Each simulated server uses a fair-share scheduling strategy to determine how much of the workload demand is and is not satisfied. The central pool sensor makes time varying information about satisfied demands available to management controllers via an open interface. The interface also is used to integrate different controllers with the emulator without recompiling its code.

Controllers periodically gather accumulated metrics and make decisions about whether to cause workloads to migrate from one server to another. Migration is initiated by a call from a controller to the central pool actuator. In our emulation environment this causes a change to the routing table that reflects the impact of the migration in the next simulated time interval. During the emulation process the metrics defined in Section 2.4 are gathered. Different controller policies cause different behaviors that we observe through these metrics.

4 Case Study

This section evaluates the effectiveness of the proposed management policies. Section 4.1 describes the workloads we consider for the study and specifies the configuration of blade and server resource pool infrastructures. Section 4.2 evaluates the sensitivity of results to the choice of migration overhead and justifies the choice used in our study. The impact of various threshold values for the workload migration controller are explored in Section 4.3. This guides our choice of thresholds for the remainder of the study. Next, Section 4.4 decides upon an appropriate control interval for the workload placement controller. This section assumes perfect knowledge of future workloads to decide upon ideal workload placements. This gives us a baseline for capacity and quality that we strive for using the management policies. The impact of the management policies on performance, quality, and power are discussed in Section 4.5. Finally, Section 4.6 shows the impact of limiting the workload placement controller's migrations on performance, quality, and the number of migrations.

4.1 Workloads and Emulated Resource Pools

We use the host load emulation environment to evaluate the integration of workload placement and migration controllers for managing resource pools. The evaluation uses real-world workload demand traces for 138 SAP enterprise applications. Traces captures average CPU and memory usage as recorded every 5 minutes for a three month interval. The host load emulator operates on this data walking forward in successive 5 minute intervals. The workloads typically required between two and eight virtual CPUs and had memory sizes between 6GB and 32GB. One workload had a memory requirement of 57GB. In our simulations, we scale the CPU demands in the traces by a factor of 1.5 to reflect a target CPU allocation that corresponds to a utilization of 0.66 which is typically desirable to ensure interactive responsiveness for enterprise workloads. Quality metrics are reported with respect to these allocations.

We consider two different resource pool configurations:

- *Blades pool* consists of blades having 8 x 2.4-GHz processor cores, 64 GB of memory, and two dual 1 Gb/s Ethernet network interface cards for network traffic and virtualization management traffic, respectively. Each blade consumes 378 Watts when idle and 560 Watts when it's fully utilized.
- *Server pool* consists of servers having 8 x 2.93-GHz processor cores, 128 GB of memory, and two dual 10 Gb/s Ethernet network interface cards for network traffic and virtualization management traffic, respectively. A server consumes 695 Watts when idle and 1013 Watts when it's fully utilized.

We note that the power values for the blades include enclosure switching and fan power. Neither of these estimates includes the power of external switching.

4.2 Impact of the Migration Overhead Using a Workload Placement Controller

This section considers the impact of CPU overhead caused by migrations. Many virtualization platforms incur virtualization overhead. Virtualization overhead depends on the type of the virtualization and its implementation specifics. Typically,

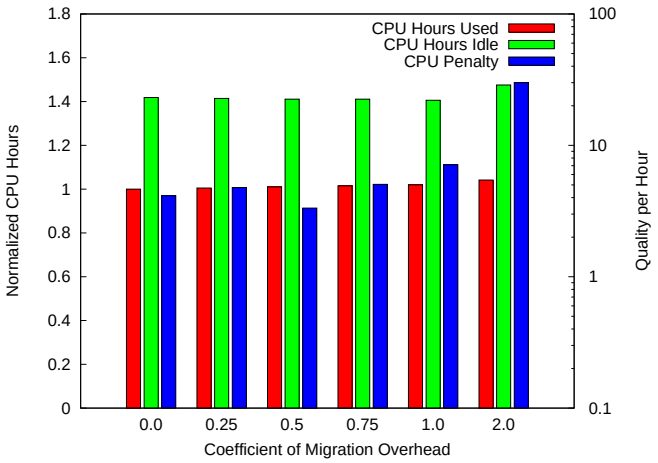

Figure 3. Migration Overhead with Blades

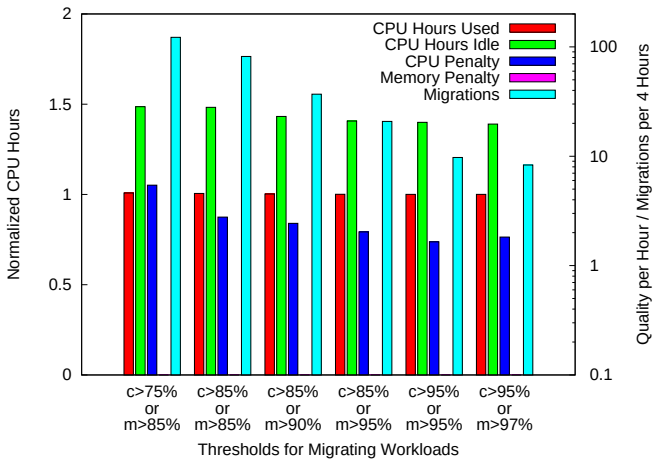

Figure 4. Aggressiveness of MC for Overload using Blades

the "amount" of CPU overhead is directly proportional to the "amount" of I/O processing [4][10]. A migration requires the memory of a virtual machine to be copied from the source server to a target server. Supporting the migration causes CPU load on both the source and target servers. The emulator reflects this migration overhead in the following way. For each workload that migrates, a CPU overhead is added to the source and destination servers. The overhead is proportional to the estimated transfer time based on the memory size of the virtual machine and the network interface card bandwidth. It is added to the source and destination servers over a number of intervals that corresponds to the transfer time. We assume that we use no more than half of the bandwidth available for management purposes, i. e., one of the two management network interface cards. For example, if a workload has 12 GB memory size and the networking interface is 1Gb/s then additional CPU time is used for migrating the workload is $(C_{migr} \cdot 12\ GB)/1\ Gb$, where C_{migr} is the coefficient of migration overhead.

To evaluate an impact of the additional CPU overhead caused by I/O processing during the workload migrations, we employ the workload placement controller with a 4 hour control interval. All workloads migrate at the end of each control interval. Figure 3 shows the results for the blade environment using migration overhead coefficient C_{migr} varied from 0.25 to 2. The figure shows several different metrics. These include the normalized CPU hours used, the normalized idle CPU hours, and the CPU quality penalty per hour.

A higher migration overhead requires more CPU resources. The impact on CPU hours used is only noticeable in Figure 3 when $C_{migr} \geq 1$. The CPU quality value, with logarithmic scale on the right of the chart, clearly degrades for $C_{migr} \geq 1$. In general, we find our results to be insensitive to values of C_{migr} in the range between 0.25 to 1.0. We choose $C_{migr} = 0.5$ used during a workload migration for the remainder of the study. This value is not unreasonable because the network interface cards we consider support TCP/IP protocol offloading capabilities.

We omit a similar figure for the server pool. The migration overhead has even less impact because of the 10 Gb/s network cards that have correspondingly lower estimated transfer times.

4.3 Workload Migration Controller Thresholds

In this section, we evaluate threshold values for the workload migration manager's overload and underload controllers. We perform a sensitivity analysis to consider the following questions. What are the suitable thresholds for migrating workloads from the overloaded servers? When should the controller start consolidating workloads from the lightly-loaded servers in order to shut them down? For all simulations we use the workload placement controller to determine an initial workload placement but subsequently we only use the workload migration controller. The workload migration controller works with 5 minute data to initiate workload migrations, shutdown and startup servers as required.

First we aim to evaluate an impact of varying *overload* thresholds. Figure 4 shows the impact of these pairs on the average number of migrations per 4 hour interval. The value is illustrated using a logarithmic scale on the right of the chart. The lower the overload thresholds the more aggressive the migration controller is when deciding whether to migrate workloads. We consider CPU/Memory threshold pairs that range from (75%,85%) through to (95%,97%). The CPU threshold has the most impact because CPU demands change more rapidly than memory demands. As we move to the right of the chart the controller identifies fewer overloads and as a result the number of migrations per 4 hours decreases from a high of 122 to a low of 8.3. Interestingly, we also see the CPU quality penalty decrease as we move to the right. On the left, CPU quality is poor due to a hysteresis between the overload and underload controllers. As the overload controller migrates workloads, sometimes adding servers, the underload controller identifies underload conditions forcing the removal of servers. This thrashing behavior introduces additional migration overheads thereby lowering quality. Furthermore, by increasing the thresholds the peak number of servers needed varies from 34 on the left to 32 on the right. Finally, we note that the memory demand changes very slowly for the workloads. As a result we are able to set the memory threshold to be high without any memory quality penalty.

In the experiments above, there are two controllers: overload and underload controllers. While we vary thresholds for the overload controller as discussed above, the underload controller operates as follows: it considers a server as a candidate for being freed and shut down when its average CPU load drops below 50% CPU and the average memory load is below 80%.

From results not illustrated, the impact of the overload thresholds in the server based environment is very similar. We

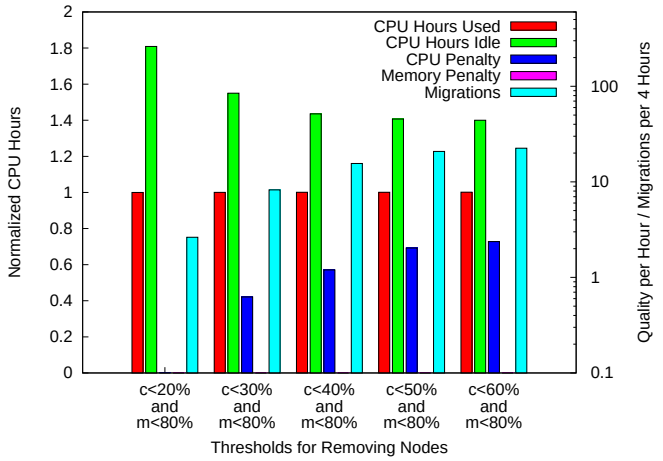

Figure 5. Aggressiveness of MC for Underload using Blades

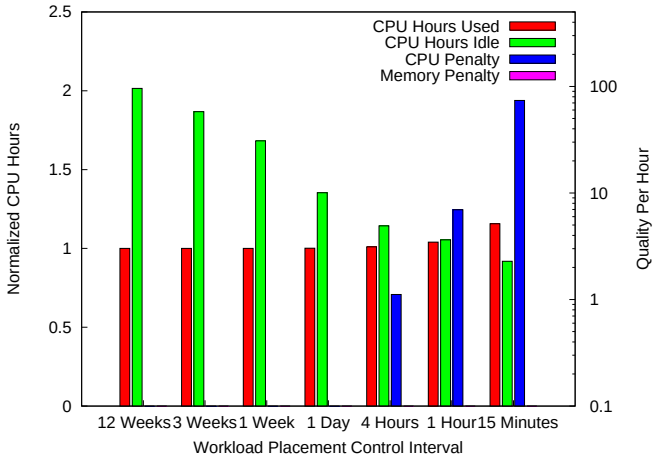

Figure 6. Results for Blade Pool Assuming Perfect Knowledge

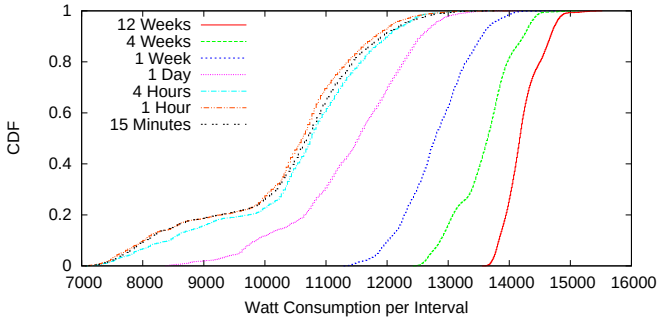

Figure 7. Power Consumption for Blades Assuming Perfect Knowledge

conclude that once high enough the exact level of the thresholds does not have much impact on the quality and resource utilization because in most cases when bursts in demands are too high they exceed all the chosen thresholds. The migration controller can't prevent all situations where some resource demands are not satisfied. To summarize, for both blade and server scenarios, we choose overload values of (85%,95%) to avoid hysteresis.

We now consider impact of varying underload thresholds. For this sensitivity analysis we use the chosen overload thresholds of (85%,95%) for the overload controller. Figure 5 considers underload thresholds that range from (20%,80%) up to (60%,80%). If the CPU and memory load drop below the specified thresholds, the migration controller tries to migrate the workloads off the least loaded server and shuts it down. In contrast to the overload controller, the underload controller does not ignite additional servers. As expected the figure shows that low CPU thresholds of 20% or 30% cause high idle CPU hours. Increasing the CPU threshold above 50% doesn't reduce CPU usage because the blades become memory bound.

From results that are not shown, the server environment also benefits most from a CPU threshold of 50%. Beyond that there are fewer idle CPU hours but the CPU quality penalty begins to increase. The server environment is not memory bound so the memory threshold has no impact.

To summarize, for the remaining experiments we choose the following thresholds for CPU and memory utilization. The overload thresholds are (85%,95%). The underload thresholds are (50%,80%). These are used for both blade and server infrastructure scenarios.

4.4 Performance, Quality, and Power Assuming Perfect Knowledge

In this section, we consider an *ideal* workload placement strategy. Figure 6 shows the results of an emulation where we use the workload placement controller to periodically consolidate the 138 workloads to a small number of blades in the resource pool. For this scenario, we assume the workload placement controller has perfect knowledge of the future, for a given time period, and chooses a workload placement such that each blade is able to satisfy the peak of its workload CPU and memory demands. The figure shows the impact on capacity requirements of using the workload placement controller once at the start of the three months, and for cases with a control interval of 3 weeks, 1 week, 1 day, 4 hours, 1 hour, and 15 minutes. The figure shows that re-allocating workloads every 4 hours captures most of the capacity savings that can be achieved, i. e., with respect to reallocation every 15 minutes. The 4 hour and 15 minute scenarios required a peak of 30 servers and the other scenarios had peaks between 30 and 34 servers. For the 4 hour scenario, we note that the normalized server CPU used and idle are approximately equal giving an average utilization close to 50% over the three month period.

The CPU penalty in Figure 6 illustrate the hourly quality metric for CPU. Recall, that the values are shown using a logarithmic scale. For the 4 hour control interval, the hourly CPU quality value was 2.2. The penalties are due the migration overheads, which, even in this ideal scenario, were not considered by the workload placement controller when deciding workload placements. The total CPU quality value over the three months was 2250. Table 1 gives a detailed breakdown of the violations.

The distribution of the Watts used is shown in Figure 7. We note that the power consumption of the 15 minutes, 1 hour,

Interval Duration	Total Number	Average Number
5 Minute	658	7.8 per Day
10 Minute	206	2.4 per Day
15 Minute	49	4 per Week
20 Minute	9	3 per Month
25 Minute	3	1 per Month
30 Minute	3	1 per Month

Table 1. CPU Quality Violations for Blades Assuming Perfect Knowledge

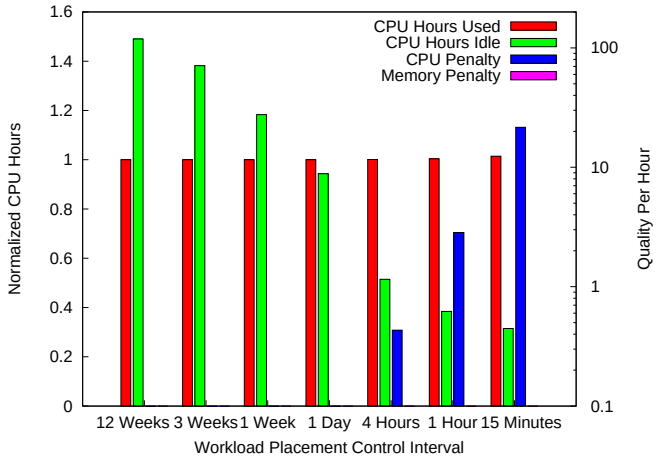

Figure 8. Results for Server Pool Assuming Perfect Knowledge

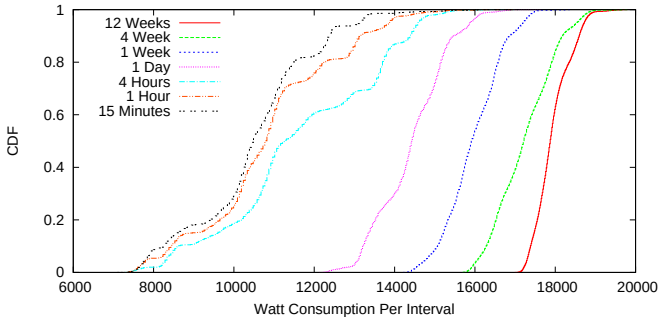

Figure 9. Power Consumption for Servers Assuming Perfect Knowledge

and 4 hour scenarios are pretty close to each other. The 15 minute case consumes more power than the 1 hour simulation even though the workloads are packed more densely and fewer servers are used on average. Re-shuffling the workloads every 15 minutes significantly increases the number of migrations and hence migration CPU overhead and power consumption. For workload placement control intervals longer than 1 day, more servers are used resulting in higher power consumption.

Figure 8 shows the results of an emulation for the server resource pool assuming perfect knowledge. Again the 4 hour workload placement control interval provides a good trade-off between capacity and quality. It provides for an average CPU utilization of 66% with a nearly perfect hourly CPU quality pentalty value of 0.4. The outcome for the server configuration is interesting: it is able to utilize the CPU more efficiently than the blade configuration. This outcome is due to the server configuration having twice the memory, i. e., 128GB instead of 64GB. SAP applications are very memory-intensive. Servers with larger memory sizes enable the consolidation of more applications to a smaller number of servers. The blade configuration is memory limited and as a result is less able to make full use of its CPU capacity, i. e., nearly 47% on average. While CPU quality and CPU utilization are higher for the server pool than for the blade pool, Figure 9 shows that more power is needed for the server pool. From the perspectives of efficiency and quality, the server based pool has an advantage. However, blades appear to have a power advantage. We consider these trade-offs in more detail in the next subsection along with how much of these *ideal* advantages we are able to achieve in practice without assuming perfect knowledge.

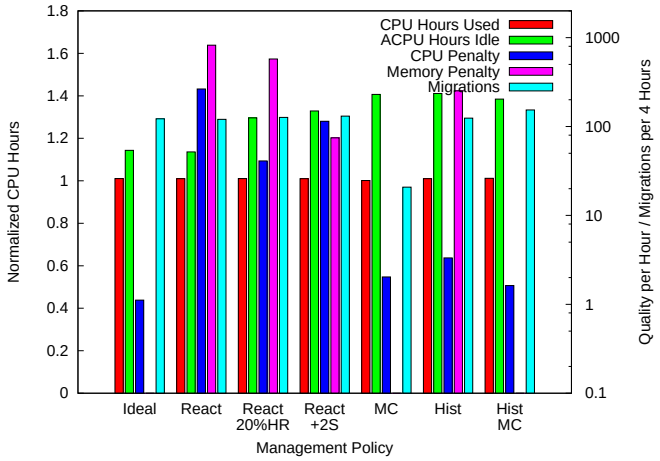

Figure 10. Management Policies for Blade Pool

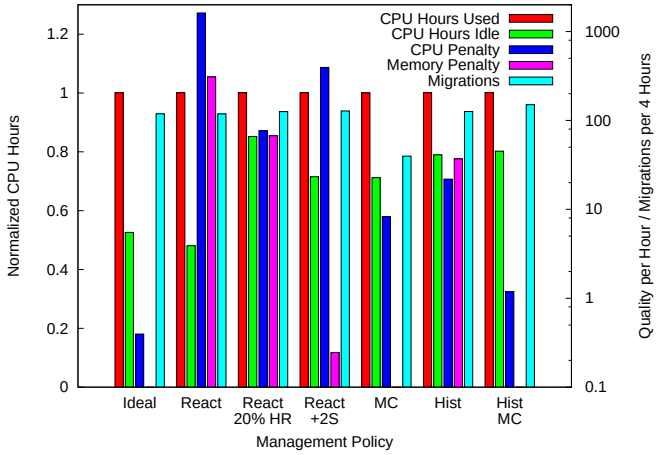

Figure 11. Management Policies for Server Pool

4.5 Performance, Quality, and Power Achieved by Management Policies

We now consider the impact of several different workload management and workload migration controller policies for managing the resource pool. Figure 10 shows results for emulations of the blade infrastructure. The ideal scenario corresponds to the 4 hour results from Figure 6 that shows ideal capacity and quality behavior. We always consider 4 hour intervals for the workload placement controller. The management policies are described in Section 2.3.

The use of a workload migration controller alone (MC) is most typical of the literature [19][30]. The MC policy does very well as a starting point. Figure 10 shows that the use of a workload placement controller in a reactive mode does not provide any advantage with respect to the workload migration controller. This is because it fails to anticipate the large variance in workload demands, for details see [8], even if we add a 20% CPU headroom or two extra servers. The historical workload placement controller does better as it anticipates changes typical of historical demand patterns, but it still uses more capacity and offers lower quality. However, by integrating the historical workload placement and the migration controllers, the hourly CPU quality is improved from a penalty of 3.3 and 2.0 for the workload placement and migration controllers separately, respectively, to 1.6 for the integrated controllers. This is an improvement of 20% over the migration controller alone. The corresponding ideal case, i. e., assuming perfect knowl-

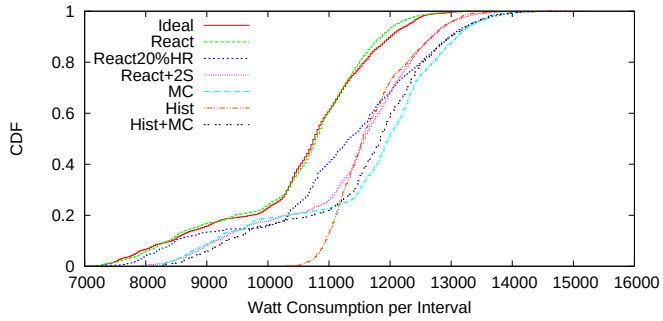

Figure 12. Power Consumption for Blades

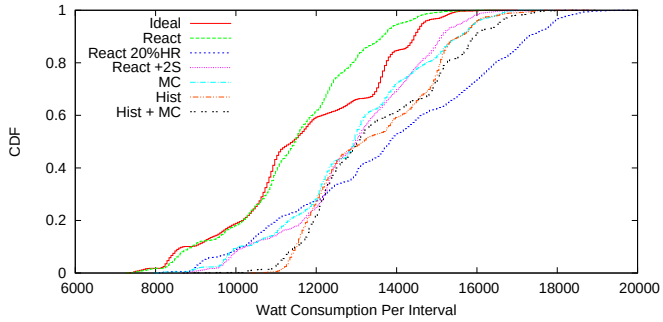

Figure 13. Power Consumption for Servers

edge and a 4 hour control interval, had an hourly CPU quality of 1.1. The integrated controllers also use slightly less resources than either controller separately and they use only 11% more CPU used and idle capacity than the corresponding ideal case. The minimum and maximum numbers of servers used during the emulated period was 25 and 33. The ideal case used between 16 and 30 servers.

Figure 11 shows the same policies for the server pool. The integrated approach provides even better results. It improves hourly CPU quality penalty from 21.9 and 8.3 for the workload placement and migration controllers separately, respectively, to 1.2 for the integrated case. The hourly CPU quality penalty for the 4 hour control interval for the ideal server pool case was 0.4. CPU quality is a bigger issue for the server pool as compared to the blade pool because the servers are not as memory bound. For this case, the integrated controllers use slightly more resources than either controller alone. During the emulated period they used between 14 and 22 servers while in the ideal case between 9 and 19 servers were used. They used a total of only 18% more CPU used and idle capacity than the corresponding ideal case.

Finally, we note that when assuming perfect knowledge, the *Hist+MC* policy offered an average CPU utilization of 47% and 66% for the blade and server pool scenarios, repectively. Without perfect knowledge we are able to achieve similar quality values with CPU utilizations of approximately 42% and 56%, respectively. Figures 12 and 13 show the distribution of aggregate power usage for the blades and servers in the pools, respectively. The *React* case has power usage closest to the ideal scenario, but has very poor CPU quality. The peak

Scenario	Peak	Average
Blade ideal(4 hour)	13740	10560
Blade *Hist+MC*	14616	11592
Server ideal(4 hour)	17016	11760
Server *Hist+MC*	18960	13536

Table 2. Power Usage in Watts per 5 Minute Interval

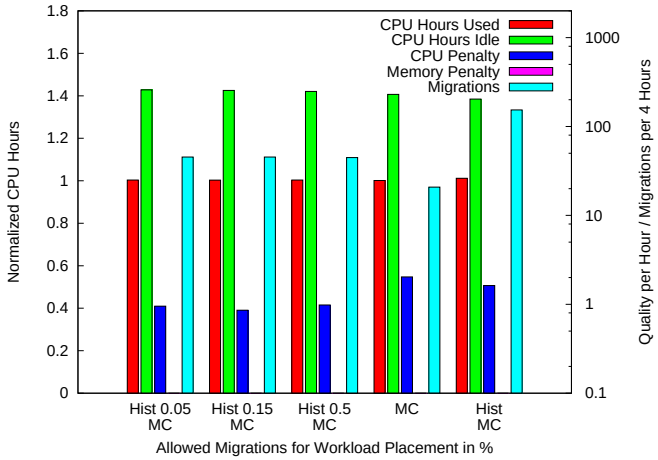

Figure 14. Desired Migrations per Workload Placement for Blade Pool

and average power usage for the ideal and integrated controller cases are summarized in Table 2.

We note that additional infrastructure is still needed for these environments. An additional two Ethernet switches are needed to connect the blade enclosures with 10Gb/s links. An additional 5 Ethernet switches are needed to connect the servers with 10Gb/s links. In total we estimate the peak power requirements for the blade and server alternatives, using the *Hist+MC* management policy, to be 16800W and 24500W, respectively, and the average power requirements to be 13800W and 19000W, respectively. The approximate hardware street prices for such hypothetical alternatives, at the time of writing, was 100 thousand US dollars more for the blade case, not including factors such as software, support and management costs. The blade infrastructure costs a little more but uses only two thirds the peak power and 70% of the average power. The server infrastructure, while using more power, may be more flexible as it is less memory bound.

4.6 Limiting Migrations For Workload Placement

In the previous simulations we didn't aim to reduce the number of migrations for the workload placement controller. We now apply the new multi-objective approach for Capman, as described in Section 2.1, to limit migrations and observe the impact on capacity, quality, and migration metrics. For this study we use the *Hist+MC* management policy as it appears to be the best policy. Figure 14 shows the results for the blade resource pool. In the figure we vary the percentage of workloads that it is desirable to migrate from 5% to 100%. The results show that introducing any desired limit at all causes much fewer migrations. Without a limit, the average number of migrations every 4 hours was 153.8. When limiting the number of migrations this drops to 45.3, including both the placement and migration controllers, which is much more comparable to the migration controller alone which had 20.8 migrations. We note that by limiting migrations the capacity used increases only slightly, due to less efficient workload placements, but the CPU quality improves further to 0.95, where the ideal was 1.1. Note that emulation in the ideal scenario were permitting far more migrations per 4 hour control interval. The increase in CPU quality is due to the lower impact of migration CPU overheads.

5 Related Work

Server consolidation is becoming an increasingly popular approach in enterprise environments to better utilize and manage systems. Manufacturers of high-end commercial servers have long provided hardware support for server consolidation such as a logical partitioning and dynamic domains [14][16]. Although virtualization has been around for more than three decades, it has found its way into the mainstream only recently with a variety of solutions – both commercial and open source – that are now available for commodity systems. Many enterprises are beginning to exploit shared resource pool environments to lower their infrastructure and management costs. The problem of efficient workload placement and workload management in such environments is in a center of attention for many research and product groups.

In our work, we chose to represent application behavior via workload demand traces. Many research groups have used a similar approach to characterize application behavior and applied trace-based methods to support what-if analysis in the assignment of workloads to consolidated servers [2][26][20][21][23][5]. A consolidation analysis presented in [2] packs existing server workloads onto a smaller number of servers using an Integer Linear Programming based bin-packing method. Unfortunately the bin-packing method is NP-complete for this problem, resulting in a computation intensive task. This makes the method impractical for larger consolidation exercises and on-going capacity management. There are now commercial tools [12][15][3][25] that employ trace-based methods to support server consolidation exercises, load balancing, ongoing capacity planning, and simulating placement of application workloads to help IT administrators improve server utilization.

We believe the workload placement service we employ has advantages over other workload placement services described above. It is supported by a genetic algorithm that improves upon greedy workload placement solutions. Furthermore, our workload placement methods go further than the other methods by addressing issues including classes of service and placement constraints. For this paper, we extended the approach to minimize migrations over successive control intervals. Some researchers propose to limit the capacity requirement of an application workload to a percentile of its demand [26]. This does not take into account the impact of sustained performance degradation over time on user experience as our required capacity definition does. Others look only at objectives for resources as a whole [23] rather than making it possible for each workload to have an independently specified objective.

Recently, virtualization platforms such as VMware and Xen [6][28] provide the ability to dynamically migrate VMs from one physical machine to another without interrupting application execution. They have implemented "live" migration of VMs that results in extremely short downtimes ranging from tens of milliseconds to a second. VM migration has been used for dynamic resource allocation in Grid environments [22][24][9]. In contrast, we focus on data center environments with stringent quality of service requirements that necessitate design of highly responsive migration algorithms.

Wood et al. [30] present Sandpiper, a system that automates the task of monitoring virtual machine performance, detecting hotspots, and initiating any necessary migrations. Sandpiper implements heuristic algorithms to determine which virtual machine to migrate from an overloaded server, where to migrate it, and a resource allocation for the virtual machine on the target server. Sandpiper implements a black-box approach that is fully OS- and application-agnostic and a gray-box approach that exploits OS- and application-level statistics. Sandpiper is closest to the migration controller presented in our paper though they implement different migration heuristics.

VMware's Distributed Resource Scheduler [27] also uses migration to perform automated load balancing in response to CPU and memory pressure. DRS uses a user space application to monitor memory usage similar to Sandpiper, but unlike Sandpiper, it does not utilize application logs to respond directly to potential application service level violations or to improve placement decisions.

Raghavendra et. al. [19] integrate sophisticated aspects of power and performance management for resource pools. They present a simulation study that optimizes with respect to power while minimizing the impact on performance. The results from simulations suggest that for integrated controllers between 3% and 5% of workload CPU demand units are not satisfied with their approach. Unsatisfied demands are not carried forward in their simulation. With our host emulation approach, we carry forward demands and focus more on per-workload quality metrics that characterize epochs of sustained overload. With our experiments, more than 99.9% of workload demands were satisfied for all cases except *React*, which was 99.8%. In [19], the authors conclude that the 3% to 5% performance degradation is acceptable to save power. We concur, but suggest this is only true in exceptional circumstances when access to power is degraded. Otherwise workload quality of service must be maintained to satisfy business objectives.

In our work, to further improve efficiency and application quality of service, we manage workloads by integrating the workload placement approach with a workload migration controller. Our simulation results show that such integrated approach provides unique performance and quality benefits.

There is a new research direction that has emerged from studying server consolidation workloads using a multicore server design [17][18]. The authors show, across a variety of shared cache configurations, that a commercial workload's memory behavior can be affected in unexpected ways by other workloads. In our work, we do not consider impact of cache sharing, while it is an interesting direction for future research.

6 Conclusions and Future Work

In this paper we considered the integration of workload placement and workload migration controllers to support resource pool management. We enhanced the Capman workload placement tool to better act as a controller. A new host emulation environment was developed to help evaluate the long term impact of management policies and combinations of controllers on various quality metrics using historical workload demand traces from real enterprise workloads. A new quality metric based on the duration of sustained overloads has been introduced to better evaluate the impact of resource sharing on workloads. The approach was applied to both emulated blade and server resource pool infrastructures.

Our results show that the integration of controllers outperforms the use of either controller separately. For the blade

pool, the integrated controllers offered CPU quality that was 20% better than for either controller separately, while using only 11% percent more capacity than the corresponding ideal case. For the server resource pool, the hourly CPU quality penalty was nearly 7 times better than either controller separately, while using 18% percent more capacity than the corresponding ideal case. Because the blades were memory bound for the workloads in our study, they had lower CPU quality penalties per hour. In fact, the migration controller did a very good job on its own for the blade pool, reacting mostly to memory quality violations. The impact of the integrated controllers was most clear for the server pool where load was more equally balanced between CPU and memory and where rapidly changing CPU contention was responsible for more migrations.

We showed that for the workloads under study, the blade based resource pool would use only 2/3 the peak power of the server based resource pool and about 30% less power in total. Based on street prices at the time of writing, the blade pool would cost approximately one hundred thousand US dollars more but would have lower power costs than the server pool. However, the server based resource pool is less memory bound and is more flexible in terms of resource sharing and supporting workloads with larger memory requirements.

The multi-objective enhancements to Capman helped to reduce workload placement controller initiated migrations by a factor of 3.4. The resulting workload placement controller initiated only 33% more migrations than the migration controller.

Our future work includes evaluating other instances of controllers and management policies, and to develop management policies that react well to more kinds of workloads and different kinds of simulated failures. Finally, we also plan to consider a greater variety of workloads.

References

[1] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimisation: NSGA-II. Proc. of the 6th Intl. Conf. on Parallel Problem Solving from Nature, London, UK, 2000.

[2] A. Andrzejak, J. Rolia, and M. Arlitt. Bounding Resource Savings of Utility Computing Models. HPLabs Technical Report, HPL-2002-339.

[3] http://www.aogtech.com/

[4] L. Cherkasova, and R. Gardner. Measuring CPU overhead for I/O processing in the Xen virtual machine monitor. Proc. of USENIX Annual Technical Conf., Apr 2005.

[5] L. Cherkasova, and J. Rolia: R-Opus: A Composite Framework for Application Performability and QoS in Shared Resource Pools. Proc. of the Intl. Conf. on Dependable Systems and Networks (DSN), Philadelphia, USA, 2006.

[6] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. Proc. of the 2nd Symp. on Networked Systems Design and Implementation (NSDI), Boston, MA, May 2005.

[7] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan. Full-System Power Analysis and Modeling for Server Environments. Workshop on Modeling, Benchmarking, and Simulation (MoBS), June 2006.

[8] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper: Workload Analysis and Demand Prediction of Enterprise Data Center Applications. Proc. of the 2007 IEEE Intl. Symposium on Workload Characterization (IISWC), Boston, MA, USA, September, 2007.

[9] L. Grit, D. Irwin, A. Yumerefendi, and J. Chase. Virtual Machine Hosting for Networked Clusters: Building the Foundations for Autonomic Orchestration. In Proc. VTDC'06.

[10] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. Enforcing Performance Isolation Across Virtual Machines in Xen. Proc. of the ACM/IFIP/USENIX 7th Intl. Middleware Conf., Melbourne, Australia, 2006.

[11] J. H. Holland. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, 1975.

[12] HP Integrity Essentials Capacity Advisor. http://h71036.www7.hp.com/enterprise/cache/262379-0-0-0-121.html

[13] HP Virtual Connect Enterprise Manager. http://h18004.www1.hp.com/products/blades/components/ethernet/vcem/index.html

[14] HP Virtual Server Environment. https://h30046.www3.hp.com/campaigns/2007/promo/VSE/index.php

[15] IBM Tivoli Performance Analyzer. http://www.ibm.com/software/tivoli/products/performance-analyzer/

[16] J. Jann, L. M. Browning, and R. S. Burugula. Dynamic reconfiguration: Basic building blocks for autonomic computing on IBM pSeries servers. IBM Systems Journal , 42(1), 2003.

[17] N. Jerger, Vantrease, and M. Lipasti An Evaluation of Server Consolidation Workloads for Multi-Core Designs. Proc. of the 2007 IEEE Intl. Symposium on Workload Characterization (IISWC), Boston, 2007.

[18] M. Marty and M. Hill. Virtual hierarchies to support server consolidation. Proc. of the 34th Intl. Symposium on Computer Architecture (ISCA), 2007.

[19] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No power struggles: Coordinated multi-level power management for the data center. To appear in Proc. of ASPLOS 2008.

[20] J. Rolia, X. Zhu, M. Arlitt, and A. Andrzejak, Statistical Service Assurances for Applications in Utility Grid Environments. Performance Evaluation Journal, vol. 58, 2004.

[21] J. Rolia, L. Cherkasova, M. Arlitt, and A. Andrzejak. A Capacity Management Service for Resource Pools. Proc. of the 5th Intl. Workshop on Software and Performance (WOSP), Spain, 2005.

[22] C P. Ruth, J. Rhee, D. Xu, R. Kennell, and S. Goasguen. Autonomic Live Adaptation of Virtual Computational Environments in a Multi-Domain Infrastructure. In Proc. of the 3nd IEEE Intl. Conf. on Autonomic Computing (ICAC), June 2006.

[23] S. Seltzsam, D.Gmach, S. Krompass, and A. Kemper. AutoGlobe: An Automatic Administration Concept for Service-Oriented Database Applications. Proc. of the 22nd Intl. Conf. on Data Engineering (ICDE), Industrial Track, Atlanta, GA, 2006.

[24] A. Sundararaj, A. Gupta, and P. Dinda. Increasing Application Performance in Virtual Environments through Run-time Inference and Adaptation. In Proc. HPDC'05.

[25] TeamQuest http://www.teamQuest.com

[26] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource Overbooking and Application Profiling in Shared Hosting Platforms. Proc. of the 5th Symp. on Operating System Design and Implementation (OSDI), Boston, MA, December, 2002.

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

[28] VMware VMotion. http://www.vmware.com/products/vi/vc/vmotion.html

[29] M. Wimmer, V. Nicolescu, D. Gmach, M. Mohr, A. Kemper, and H. Krcmar. Evaluation of Adaptive Computing Concepts for Classical ERP Systems and Enterprise Services. *Proc. of the Joint Conf. on E-Commerce Technology and Enterprise Computing, E-Commerce and E-Services*, San Francisco, CA, 2006.

[30] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Black-box and gray-box strategies for virtual machine migration. In Proc. of the 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI), April, 2007.