

# Resource Pool Management: Reactive versus Proactive or Let's be Friends

Daniel Gmach<sup>a</sup>, Jerry Rolia<sup>a</sup>, Ludmila Cherkasova<sup>a</sup>, Alfons Kemper<sup>b</sup>

<sup>a</sup>Hewlett-Packard Laboratories, Palo Alto, CA, USA

<sup>b</sup>Technische Universität München, 85748 Garching, München

---

## Abstract

The consolidation of multiple workloads and servers enables the efficient use of server and power resources in shared resource pools. We employ a trace-based workload placement controller that uses historical information to periodically and proactively reassign workloads to servers subject to their quality of service objectives. A reactive migration controller is introduced that detects server overload and underload conditions. It initiates the migration of workloads when the demand for resources exceeds supply. Furthermore, it dynamically adds and removes servers to maintain a balance of supply and demand for capacity while minimizing power usage. A host-load simulation environment is used to evaluate several different management policies for the controllers in a time effective manner. A case study involving three months of data for 138 SAP applications compares three integrated controller approaches 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. Our study sheds light on the question of whether a reactive controller or proactive workload placement controller alone is adequate for resource pool management. The results show that the most tightly integrated controller approach offers the best results in terms of capacity and quality but requires more migrations per hour than the other strategies.

*Key words:* Virtualized Data Centres, Resource Pool Management, Enterprise Workload Analysis, Simulation

---

## 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 only end up using 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 chang-

---

*Email addresses:* daniel.gmach@hp.com (Daniel Gmach), jerry.rolia@hp.com (Jerry Rolia), lucy@viola.hpl.hp.com (Ludmila Cherkasova), alfons.kemper@in.tum.de (Alfons Kemper)

ing business conditions. For large enterprises, virtualization offers a solution for server and application consolidation in shared resource pools. 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 centres 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 centre 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 [1] 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 offer a model of application demands that are representative of future application behaviour. Traces are used to decide how to consolidate workloads to servers. In our past work, we assumed that the placement of workloads would be adjusted infrequently, e. g., weekly or monthly [1]. 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 [2] 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 [3, 4], 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 behaviour 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 exploit a host load simulation environment. The 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 simulator collects 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. This paper significantly enhances our recent work [5] by considering more advanced policies, by introducing new quality of service metrics, and by gaining new insights using an analysis of variance (ANOVA) upon data from more than ten times the previous number of simulation runs. The results of the ANOVA show that for our policies and case study data thresholds that define underload conditions have a greater impact on capacity and quality than those that define overload conditions. Finally, we found that the tight integration of controllers outperforms the use of the controllers in parallel or separately but in general causes more migrations per hour.

The remainder of this paper is organized as follows. Section 2 describes the workload placement and migration controllers, management policies, and metrics. The host load simulation 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* simulates 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 [6].

The workload placement controller is based on the Capman tool that is described further in [1, 5]. It supports both consolidation and load levelling exercises. Load levelling 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

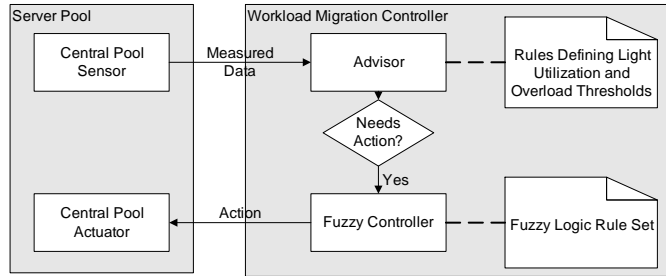


Figure 1: Architecture of the Workload Migration Controller.

aggregate demand. It is capable of supporting a different quality of service for each workload [7]. 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 exploit Capman’s multi-objective optimization functionality. Instead of simply finding the smallest number of servers needed to support a set of workloads, Capman 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  $t$  as a bound for the number of workloads that it is desirable to migrate. Limiting the number of migrations limits the migration overhead 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. A data centre operator could choose a value  $t$  based on experience regarding the overhead that migrations place upon network infrastructure and servers. The case study explores the sensitivity of capacity and quality metrics to the parameter  $t$ .

## 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 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. In an overload situation, first, a fuzzy controller determines a

workload to migrate away and then it 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 does not exist, we start up a new server and migrate the workload to the new one.

- An underload situation occurs whenever the CPU and memory usage averaged over *all* servers in the server pool drops below a specified threshold. While an overload condition is naturally defined with respect to a particular server, the underload situation is different. It is defined with respect to the average utilization of the overall system involving all the nodes. In this way, we try to avoid system thrashing: e.g., a new server generally starts with a small load and should not be considered immediately for consolidation. In an underload situation, first, the fuzzy controller chooses the least loaded server and tries to shut it down. For every workload on this server, the 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 4.4 evaluates 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 [8, 3].

### 2.3. Policies

The policies we consider evaluate whether a reactive controller or workload placement controller alone is adequate for resource pool management and whether the integration of controllers provides compelling benefits. Our study considers the following management policies:

- *MC*: migration controller alone;
- *WP*: workload placement controller operating periodically alone;
- *MC + WP*: workload placement controller operating periodically with the migration controller operating in parallel;
- *MC + WP on Demand*: migration controller is enhanced to invoke the workload placement controller on demand to consolidate workloads whenever the servers being used are lightly utilized; and,
- *MC + WP + WP on Demand*: workload placement controller operating periodically *and* the migration controller is enhanced to invoke the workload placement controller on demand to consolidate workloads whenever the servers being used are lightly utilized.

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 migrate workloads to alleviate overload and underload situations as they occur. The workload migration controller operates at the time scale that measurement data is made available. In this paper, the migration controller is invoked every 5 minutes.

With the *WP* policy, the workload placement controller uses historical workload demand trace information from the previous week that corresponds to the next control interval, e. g., the next four hours. In this way it periodically re-computes a globally efficient workload placement. The historical mode is most likely appropriate for enterprise workloads that have repetitive patterns for workload demands.

The *MC + WP* policy implements the *MC* and *WP* policies in parallel, i. e., the workload placement controller is executed for each control interval, e. g., every four hours, to compute a more effective workload placement for the next control interval. Within such an interval, the migration controller, independently, migrates workloads to alleviate overload and underload situations as they occur.

The *MC + WP on Demand* policy integrates the placement and migration controllers in a special way. Instead of running the workload placement controller after each workload placement control interval, the migration controller uses the workload placement algorithm to consolidate the workloads whenever servers being used are lightly utilized.

Finally, the *MC + WP + WP on Demand* policy is the same as *MC + WP on Demand* policy but also invokes the workload placement controller after every control interval, e. g., every four hours, to periodically provide a globally efficient workload placement.

#### 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 hours.

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 [9].

We define and introduce a new quality metric named *violation penalty* that is based on the number of successive intervals where a workload’s demands are not fully satisfied and the expected impact on the customer. Longer epochs of unsatisfied demand incur greater penalty values, as they are more likely to be perceived by those using applications. For example, if service performance is degraded for up to 5 minutes customers would start to notice. If the service is degraded for more than 5 minutes then customers may start to call the service provider and complain. Furthermore, larger degradations in service must cause greater penalties.

The quality of the delivered service depends on how much the service is degraded. If demands greatly exceed allocated resources then the utility of the service suffers more than if demands are almost satisfied. Thus, for each violation a penalty weight  $w_{pen}$  is defined that is based on the expected impact of the degraded quality on the customer. The violation penalty value  $pen$  for a violation with  $I$  successive overloaded measurement intervals is defined as  $pen = I^2 \max_{i=1}^I (w_{pen,i})$ , where  $w_{pen,i}$  is the penalty in the  $i$ th interval. Thus longer violations tend to have greater penalties than shorter violations<sup>1</sup>. The weight functions used for CPU and memory are given below. The sum of penalty values over all workloads over all violations defines the violation penalty for a metric.

Regarding CPU allocations, we estimate the impact of degraded service on a customer using a heuristic that compares the actual and desired utilization of allocation for the customer. An estimate is needed because we do not have measurements that reflect the actual impact on a customer. Let  $u_a$  and  $u_d < 1$  be the actual and desired CPU utilization of allocation for an interval. If  $u_a \leq u_d$  then we define the weight for the CPU penalty  $w_{pen}^{CPU}$  as  $w_{pen}^{CPU} = 0$  since there is no violation. If  $u_a > u_d$  then response times will be higher than planned so we must estimate the impact of the degradation on the customer. We define:

$$w_{pen}^{CPU} = 1 - \frac{1 - u_a^k}{1 - u_d^k}.$$

The penalty has a value between 0 and 1 and is larger for bigger differences and higher utilizations. The superscript  $k$  denotes the number of CPUs on the server. This formula is motivated by a formula that estimates the mean response time for the  $M/M/k$  queue [10], namely  $r = 1/(1 - u^k)$  estimates the mean response time for a queue with  $k$  processors and unit service demand [11]. The power term  $k$  reflects the fact that a server with more processors can sustain higher utilizations without impacting customer response times. Similarly, a customer that has a higher than desired utilization of allocation will be less impacted on a system with more processors than one with fewer processors.

Regarding memory allocations, we estimate the impact of degraded service on a customer using a heuristic that compares the actual allocation of memory  $l_a$  and desired allocation of memory  $l_d$  for a customer. If  $l_a \geq l_d$  then we define a memory penalty weight  $w_{pen}^{Mem} = 0$  since there is no violation. If  $l_a < l_d$ , we define  $w_{pen}^{Mem} = 1 - hr$ , where  $hr$  is the memory hit ratio. In our simulation, the hit ratio is measured as the percentage

---

<sup>1</sup>We note that such penalties may be translated to monetary penalties in financially driven systems and that monetary penalties are likely to be bounded in such systems.

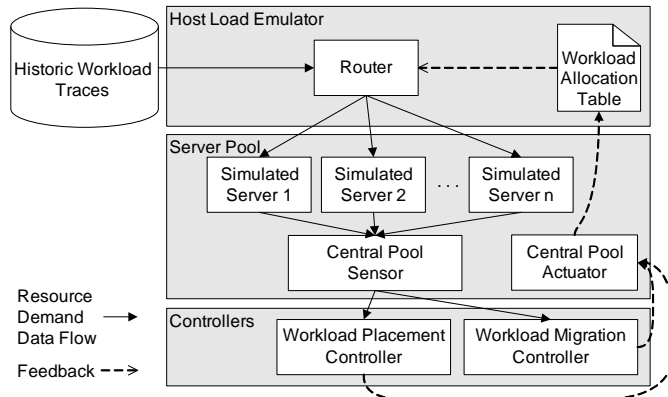


Figure 2: Architecture of the Host Load Simulator.

of satisfied memory demands in bytes.

To summarize, the CPU and memory violation penalties reflect two factors. They reflect the severity and length of the violation. The severity of the violation is captured by a weight function. Two weight functions were introduced, but others could be employed as well.

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 simulated hours to facilitate interpretation.

### 3. Host Load Simulator

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

The architecture of the host load simulation environment is illustrated in Figure 2. The simulator 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 simulator without recompiling its code.

The controllers periodically gather accumulated metrics and make decisions about whether to cause workloads to migrate from one server to another. Migration is initi-

ated by a call from a controller to the central pool actuator. In our simulation environment this causes a change to the routing table that reflects the impact of the migration in the next simulated time interval. During the simulation process the metrics defined in Section 2.4 are gathered. Different controller policies cause different behaviours that we observe through these metrics.

#### 4. Case Study

This section evaluates the effectiveness of the proposed management policies using three months of real-world workload demand traces for 138 SAP enterprise applications. The traces are obtained from a data centre that specializes in hosting enterprise applications such as customer relationship management applications for small and medium sized businesses. Each workload was hosted on its own server so we use resource demand measurements for a server to characterize the workload’s demand trace. The measurements were originally recorded using *vmstat* [12]. Traces capture average CPU and memory usage as recorded every 5 minutes for a four month interval.

As many of the workloads are interactive enterprise workloads, a maximum utilization of 0.66 is desired to ensure interactive responsiveness. Hence, CPU demands in the historical workload traces are scaled with a factor of 1.5 to achieve a target utilization of 0.66. The resource pool simulator operates on this data walking forward in successive 5 minute intervals. In addition to the three months of the real-world demand traces we used data from the previous month to initially fill the demand buffers of the central pool sensor. This enables the integrated management services to access prior demand values at the start of a simulation run.

We consider the following resource pool configuration<sup>2</sup>: each server consists of 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. Each server consumes 695 Watts when idle and 1013 Watts when it is fully utilized.

Section 4.1 gives a workload characterization for the SAP workloads considered in the study. Section 4.2 begins our study at workload placement by considering the impact of migration overhead. Section 4.3 considers the question: how much capacity and power can be saved by periodically consolidating workloads? The section assumes perfect knowledge about future workload demands and its results give a baseline for capacity savings that is used for the rest of the case study. Sections 4.4 and 4.5 do not assume perfect knowledge of future demands. They consider tuning of migration controller parameters and compare the capacity savings offered by the migration controller with the workload placement controller and three scenarios where the controllers are integrated. Finally, Section 4.6 presents results that demonstrate that the number of migrations caused by the workload placement controller can be significantly reduced without a major impact on CPU capacity or violation penalty.

---

<sup>2</sup>Service providers can use the proposed approach for evaluating different hardware platforms. For example, in [5] we made recommendations regarding server and blade based resource pool configurations.

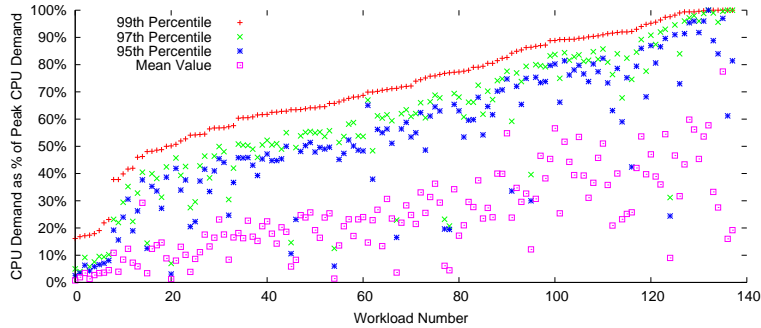


Figure 3: Top Percentile of CPU Demand for Applications under Study.

#### 4.1. Workload Characteristics

Use of virtualization technology enables the creation of shared server pools where multiple application workloads share each server in the pool. Understanding the nature of enterprise workloads is crucial to properly designing and provisioning current and future services in such pools.

Existing studies of internet and media workloads [13, 14] indicate that client demands are highly variable (“peak-to-mean” ratios may be an order of magnitude or more), and that it is not economical to overprovision the system using “peak” demands. Do enterprise workloads exhibit similar properties? We present results that illustrate the peak-to-mean behaviour for 138 enterprise application workloads. Understanding of burstiness for enterprise workloads can help to choosing the right trade-off between the application quality of service and resource pool capacity requirements. This section analyses burstiness and access patterns of the enterprise application workloads under study. It shows percentiles of demands, the maximum durations for contiguous demands beyond the 99<sup>th</sup> percentile, and a representative demand trace for an interactive enterprise application.

Figure 3 gives the percentiles of CPU demand for the 138 applications over the period of four months. The illustrated demands are normalized as a percentage with respect to their peak values. Several curves are shown that illustrate the 99<sup>th</sup>, 97<sup>th</sup>, and 95<sup>th</sup> percentile of demand as well as the mean demand. The workloads are ordered by the 99<sup>th</sup> percentile for clarity. The figure shows that more than half of all studied workloads have a small percentage of points that are very large with respect to their remaining demands. The left-most 60 workloads have their top 3% of demand values between 10 and 2 times higher than the remaining demands in the trace. Furthermore, more than half of the workloads observe a mean demand less than 30% of the peak demand. These curves show the bursty nature of demands for most of the enterprise applications under study. Consolidating such bursty workloads onto a smaller number of more powerful servers is likely to reduce the CPU capacity needed to support the workloads.

The corresponding percentiles for the memory demands of the 138 applications are shown in Figure 4. Again, the illustrated demands are normalized as percentage with respect to the peak memory demand. The curves show that the average memory demand of an application is closer to its peak demand than it is observed for CPU.

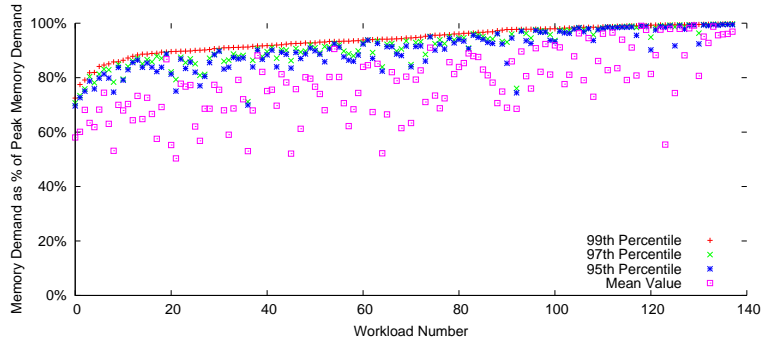


Figure 4: Top Percentile of Memory Demand for Applications under Study.

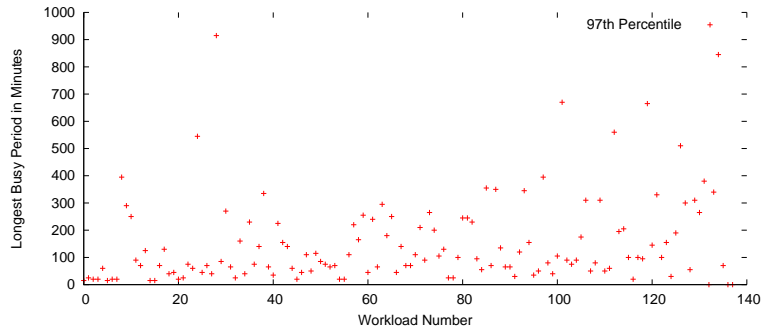


Figure 5: Longest Busy Periods above 99<sup>th</sup> Percentile of Demand for Studied Applications.

45% of the workloads exhibit a mean demand above 80% of their peak demands. Thus, in a memory bound infrastructure the potential resource savings from resource sharing is expected to be smaller than in CPU bound systems.

An additional and complementary property for a workload is the maximum duration of its contiguous application demands. While short bursts in demand may not significantly impact a workload’s users, a system must be provisioned to handle sustained bursts of high demand. However, if an application’s contiguous demands above the 99<sup>th</sup> percentile of demand are never longer than 10 minutes then it may be economical to support the application’s 99<sup>th</sup> percentile of demand and allow the remaining bursts to be served with degraded performance [7]. We have analysed the maximum duration of bursts of CPU and memory demands for the workloads. Figure 5 shows the duration of each workload’s longest burst in CPU demand that is greater than its corresponding 99<sup>th</sup> percentile of demand.

From the figure, we see that:

- 83.3% of workloads have sustained bursts in CPU demand that last more than 15 minutes; and,
- 60% of workloads have sustained bursts in CPU demand that last more than 30 minutes.

These are significant bursts that could impact an end user’s perception of performance. A similar analysis for memory demands shows that:

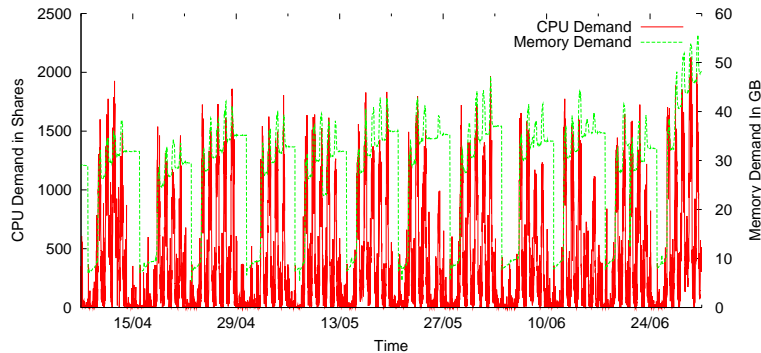


Figure 6: CPU and Memory Demands for a User Interactive Workload.

- 97.8% of workloads have sustained bursts in memory demand that last more than 15 minutes; and,
- 93.5% of workloads have sustained bursts in memory demand that last more than 30 minutes.

The numbers show that the length of the bursts matters. This justifies our use of the quality metric that takes the number of successive intervals where a workload's demands are not satisfied into account.

The analysis also shows that the CPU demands are much more variable than memory demands. CPU demands fluctuate with user load. Memory demands tend to increase then periodically decrease due to some form of memory garbage collection. For the applications in our case study, garbage collection appeared to occur each weekend. Figure 6 illustrates the behaviour of a typical workload.

Finally, a workload pattern analysis (following the methodology introduced in [4]) is conducted. Figure 7 gives a summary of the pattern lengths for the 138 workloads. The pattern analysis discovered patterns with lengths between three hours and seven weeks:

- 67.6% of the workloads exhibit a weekly behaviour; and,
- 15.8% of the workloads exhibit a daily behaviour.

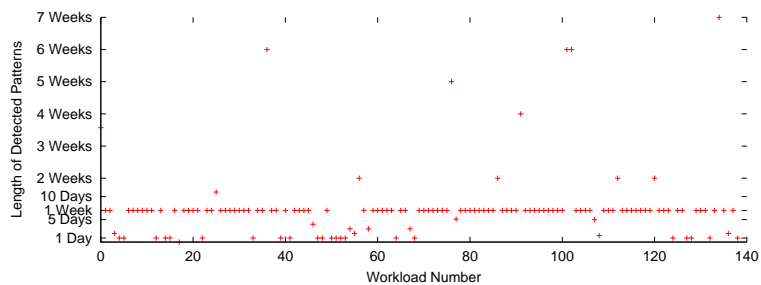


Figure 7: Lengths of Workload Demand Patterns.

To summarize, this section has shown that there are significant bursts in demand for the workloads and there is a greater opportunity for CPU sharing than for memory

sharing. A workload pattern analysis shows that most of the enterprise workloads exhibit strong weekly or daily patterns for CPU usage. Memory usage tends to increase over a week then decrease suddenly.

#### 4.2. Impact of Migration Overhead with a Workload Placement Controller

This section considers the impact of CPU overhead caused by migrations on required CPU capacity and on CPU violation penalty per hour. We do not focus on memory violation penalties, as these values were typically small.

Many virtualization platforms incur virtualization overhead. Virtualization overhead depends on the type of the virtualization and its implementation specifics. A migration requires the memory of a virtual machine to be copied from the source server to a target server. Typically, the “amount” of CPU overhead is directly proportional to the “amount” of I/O processing [15, 16]. Supporting a migration causes CPU load on both the source and target servers. The simulator 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  $(C_{migr} \cdot 12 \text{ GB}) / 1 \text{ Gb}$ , 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 8 shows the results using migration overhead coefficient  $C_{migr}$  varied from 0 to 2. The figure shows several different metrics. These include the normalized CPU hours used, the normalized idle CPU hours, and the CPU violation penalty per hour.

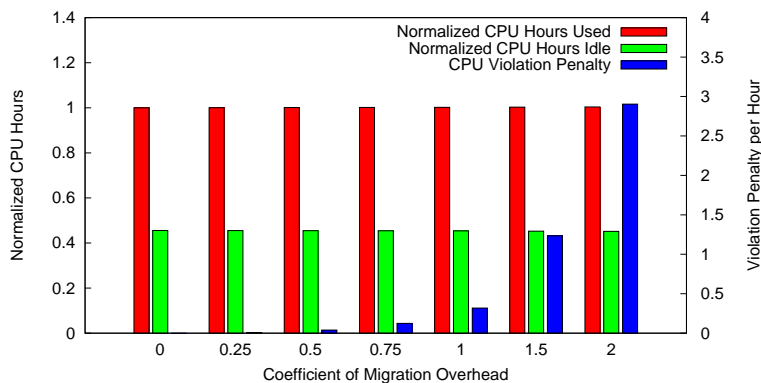


Figure 8: Migration Overhead.

A higher migration overhead requires more CPU resources. The impact on CPU hours used is only noticeable in Figure 8 when  $C_{migr} \geq 1$ . The CPU violation penalty

clearly increases for  $C_{migr} \geq 1$ . In general, we find our results to be insensitive to values of  $C_{migr}$  in the range between 0 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. There are many reports suggesting that such cards can be driven to 10Gbps bidirectional bandwidth while using 50% or less of a CPU, e. g., [17].

### 4.3. Performance, Quality, and Power Assuming Perfect Knowledge

In this section, we consider an *ideal* workload placement strategy. This approach assumes that we have perfect knowledge of future resource demands. It gives an upper bound for the potential capacity savings from consolidating workloads at different time scales. We use this bound later in the paper to determine how well our policies, that do not have perfect knowledge, perform compared to the ideal case.

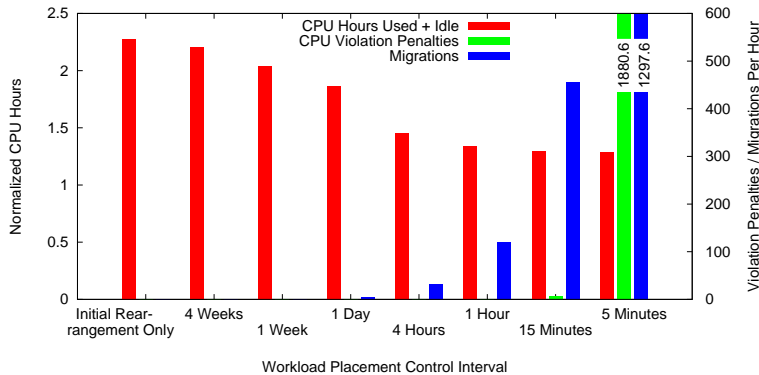


Figure 9: Simulation Results Assuming Perfect Knowledge.

Figure 9 shows the results of an simulation where we use the workload placement controller to periodically consolidate the 138 workloads to a small number of servers in the resource pool. For this scenario, for a given time period, the workload placement controller chooses a placement such that each server 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 4 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 19 servers. All the other scenarios also had peaks between 19 and 21 servers. For the 4 hour scenario, we note that the normalized server CPU hours used is approximately one-half of the idle CPU hours giving an average utilization close to 69% over the three month period with a negligible hourly CPU violation penalty value of 0.4. In subsequent subsections, we treat the results from the 4 hour ideal case as the baseline for capacity and quality.

Figure 9 shows that as expected as the control interval drops to the hourly, fifteen minute, and five minute levels the number of migrations per hour increases proportionally as most workloads are likely to be reassigned. The resulting migration overheads

increase the CPU quality violations. Table 1 gives a more detailed breakdown of the violations for the four hour control interval case.

Interval Duration	Total Number	Average Number
5 Minute	1090	13 per Day
10 Minute	161	1.9 per Day
15 Minute	14	1.2 per Week
20 Minute	3	1 per Month

Table 1: CPU Quality Violations Assuming Perfect Knowledge for the 4 Hour Control Interval.

The distribution of the Watts used is shown in Figure 10. We note that the power consumption of the 15 minutes, 1 hour, and 4 hour scenarios are pretty close to each other. For workload placement control intervals longer than 1 day, more servers are used resulting in higher power consumption.

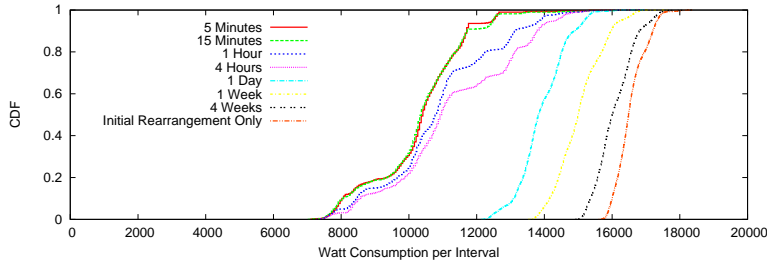


Figure 10: Power Consumption Assuming Perfect Knowledge.

In later subsections, we consider how much of these *ideal* advantages we are able to achieve in practice without assuming perfect knowledge. The workload placement controller control interval is chosen as four hours.

#### 4.4. Workload Migration Controller Thresholds

This section evaluates the effectiveness of the migration controller. The experiments start with an ideal workload placement for the first 4 hours and use the fuzzy logic based migration controller to maintain the resource access quality of the workloads. The advisor module of the controller is configured as follows: it triggers the fuzzy controller if either a server is overloaded or the system is lightly utilized. A server is considered overloaded if the CPU or memory utilization exceeds a given threshold. In that case, it triggers the fuzzy controller that tries to migrate one workload from the concerned server to a less loaded one. Furthermore, the advisor deems a server pool lightly utilized, if the average CPU and memory utilization over all servers fall below their given thresholds. Then, the fuzzy controller chooses the least loaded server, migrates all of its workloads to other servers, and shuts down the server.

To evaluate the impact of the feedback controller, the following levels for the thresholds are considered:

- $\alpha$ : The CPU threshold defining overloaded servers varies from 80%, 85%, 90%, 95%, and 99% CPU utilization.

- $\beta$ : The memory threshold defining overloaded servers varies from 80%, 85%, 90%, 95%, and 99% memory utilization.
- $\delta$ : The CPU threshold defining a lightly utilized resource pool varies from 30%, 40%, 50%, and 60% average CPU utilization of the server pool.
- $\gamma$ : The memory threshold defining a lightly utilized resource pool varies from 30%, 40%, 50%, 60%, 70%, and 80% average memory utilization of the server pool.

A three months simulation is conducted for each of the factor level combinations resulting in a total number of 600 experiments.

An ANOVA model [18] captures the effects of factor levels such as different values for thresholds on a metric, e. g., on the CPU violation penalty or CPU capacity metric. Each factor level has a numerical effect on the metric. The sum of each factor's effects adds to zero. The effect is defined as the difference between the overall mean value for the metric over all combinations of factor levels and the numerical impact of the factor level on the metric with respect to the overall mean value. Similarly, interactions between factor levels also have effects. An analysis of variance considers the sum of squares of effects. The sums of squares are variations for the metric. The analysis quantifies the impact of factors and interactions between factors on the total variation over all combinations of factor levels. When the assumptions of the ANOVA modelling approach hold, a statistical F-test can be used to determine which factors and interactions between factors have a statistically significant impact on the metric and to quantify the impact.

The assumptions of an ANOVA are:

- the effects of factors are additive;
- uncontrolled or unexplained experimental variations, which are grouped as experimental errors, are independent of other sources of variation;
- variance of experimental errors is homogeneous; and,
- experimental errors follow a Normal distribution.

The model we employ for the CPU capacity and CPU violation penalty metrics is illustrated with the following equation:

$$\begin{aligned} \text{Metric} = & \mu + \alpha_i + \beta_j + \gamma_k + \delta_l + (\alpha\beta)_{ij} + (\alpha\gamma)_{ik} + (\alpha\delta)_{il} + (\beta\gamma)_{jk} + (\beta\delta)_{jl} + (\gamma\delta)_{kl} + \varepsilon \\ & i \in \{80, 85, 90, 95, 99\}, j \in \{80, 85, 90, 95, 99\}, \\ & k \in \{30, 40, 50, 60\}, \text{ and } l \in \{30, 40, 50, 60, 70, 80\} \end{aligned}$$

In the model  $\alpha$ ,  $\beta$ ,  $\gamma$ , and  $\delta$  correspond to factors CPU and memory overload threshold and CPU and memory underload threshold, respectively. The model states that a metric, e. g., CPU violation penalty, is equal to a mean value over all experiments  $\mu$  plus an effect that is due to each factor's level plus an effect that is due to pair-wise interactions for factors, e. g., the  $i^{\text{th}}$  threshold for CPU overload *and* the  $k^{\text{th}}$  threshold for CPU underload. The error term  $\varepsilon$  includes the effects of higher level interactions, i. e., three and four factor interactions.

For the ANOVA models we consider, the factors have an additive impact on the metrics not a multiplicative impact. The experiments are fully controlled, experimental

errors are defined as higher order interactions, which from a detailed analysis have small effects. Visual tests suggest that the errors are homogeneous. The Normality assumption for errors is discussed next.

For the *CPU violation* penalty the results of a Kolmogorov-Smirnov test (K-S test) [18] for the 600 errors resulted in a D-value of 0.0627, which concludes that the error values are Normally distributed with  $\alpha = 0.01$ . However, after removing the ten largest of the 600 errors, the K-S test indicates that the remaining 590 errors are Normally distributed with significance  $\alpha = 0.2$ . This suggests that 20% of randomly generated Normally distributed data sets will have a greater difference from the Normal distribution than our experiment’s error data. Hence, we conclude that the error values are Normally distributed and the ANOVA model can be applied. For the *CPU capacity*, the K-S test yields a D-value  $D = 0.049122197$  suggesting that the 600 errors are Normally distributed with a significance level  $\alpha = 0.1$ . As with the CPU violation penalty model, removing a few extreme values dramatically increases the significance level.

Source	SS	SS in %	df	MS	F-Value	crit.	Conclusion
$\alpha$	32209682	0.17	4	8052420	50.17	2.39	significant
$\beta$	442121	0	4	110530.3	1	2.39	not significant
$\delta$	3952996127	20.96	3	1317665376	8209.6	2.623	significant
$\gamma$	9077000224	48.13	5	1815400045	11310.6	2.232	significant
$\alpha\beta$	2997479	0.02	16	187342	1.167	1.664	not significant
$\alpha\delta$	20512291	0.11	12	1709358	10.65	1.772	significant
$\alpha\gamma$	37595875	0.2	20	1879794	11.712	1.592	significant
$\beta\delta$	701391	0	12	58449.2	0.3	1.772	not significant
$\beta\gamma$	2969998	0.02	20	148500	0.925	1.592	not significant
$\delta\gamma$	5653318689	29.98	15	376887913	2348.2	1.687	significant
Error	78325016	0.41	488	160502.1			
Total:	18859068891	100	599				

Table 2: ANOVA Table for CPU Violation Penalty.

ANOVA results are presented in a table, e. g., Table 2. The columns identify the factor and factor interactions (Source), each source’s sum of squares of effects (SS) as an absolute value and as a percentage (SS in %) of the total SS over all sources, the number of degrees of statistical freedom that contribute to the SS, the mean square value (MS) which is the SS for a source divided by its number of degrees of freedom, the computed F-value for the mean square value, the critical value for the F-value that determines statistical significance, and finally the conclusion regarding whether a source has a statistically significant impact on the metric.

Table 2 gives the results of the ANOVA for the factors regarding CPU violation penalty. The table shows that factors  $\delta$  and  $\gamma$ , the CPU and memory thresholds for defining underloaded servers, and their interaction explains 99% of the variation in CPU violation penalty over the 600 experiments. Interestingly, factors  $\alpha$  and  $\beta$ , thresholds for overloaded CPU and memory, had little impact on quality. The bursts in demand for the traces under study were often larger than the headroom remaining on a server regardless of the chosen threshold level. The underload factors had a much

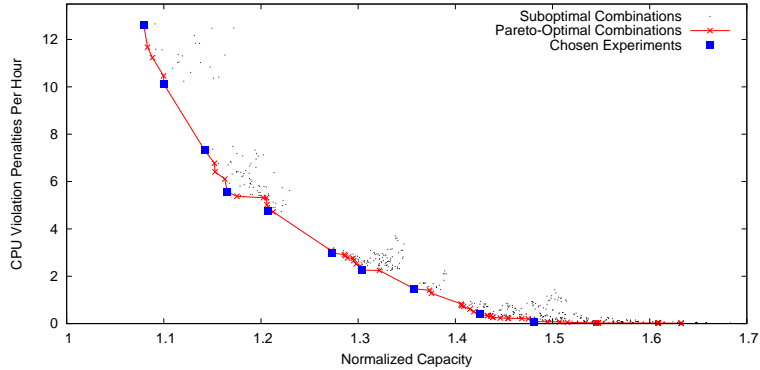


Figure 11: Chosen Combinations of Thresholds for Further Experiments.

bigger impact on quality. They guide consolidation. Lower threshold values limit consolidation so that the same applications use more servers and violations are less likely.

Source	SS	SS in %	df	MS	F-Value	crit.	Conclusion
$\alpha$	18451760914	2.95	4	4612940228	1689.8	2.39	significant
$\beta$	203448728	0.03	4	50862182	18.632	2.39	significant
$\delta$	93715380384	14.97	3	31238460128	11443.3	2.623	significant
$\gamma$	4.34742E+11	69.44	5	86948493051	31851.1	2.232	significant
$\alpha\beta$	101095635	0.02	16	6318477	2.315	1.664	significant
$\alpha\delta$	299339472	0.05	12	24944956	9.138	1.772	significant
$\alpha\gamma$	3502757297	0.56	20	175137865	64.157	1.592	significant
$\beta\delta$	66662589	0.01	12	5555216	2.035	1.772	significant
$\beta\gamma$	462381183	0.07	20	23119059	8.469	1.592	significant
$\delta\gamma$	73166116866	11.69	15	4877741124	1786.8	1.687	significant
Error	1332165080	0.21	488	2729846			
Total:	6.26044E+11	100	599				

Table 3: Analysis of Variance Table for CPU Capacity.

Table 3 gives the results of an ANOVA for the factors regarding the CPU capacity. Factor  $\gamma$ , the memory threshold for defining underloaded servers, has an even larger impact on capacity than on CPU violation penalty. Again, factors  $\delta$  and  $\gamma$  and their interaction explain nearly all, 97%, of the variation. Recognizing underload conditions is clearly an important aspect of policy for managing resource pools.

The results of the 600 simulations are shown in Figure 11 as small black dots. The figure illustrates CPU violation penalties versus normalized CPU capacity required under different policy configurations. *Normalized capacity* is defined as the sum of the total server CPU hours used and idle CPU hours divided by the sum of the total server CPU hours used and idle CPU hours used for the ideal case with a 4 hour workload placement interval.

Each of the 600 simulations represents a combination of factor levels. As expected, the figure shows that as workloads are consolidated more tightly and capacity is reduced there is an increase in the CPU violation penalties. The specific shape of this

curve is workload and resource pool specific and reflects the variability in workload demands.

A Pareto-optimal set of simulation runs is illustrated in Figure 11 using a red line. These combinations of factor levels provided lowest CPU violation penalties and/or the lowest normalized CPU capacity. Ten of the Pareto-optimal combinations are chosen representing the best behaviours of the migration controller and serve as a baseline for the remainder of the paper. A data centre operator could choose any one of these as a best behaviour depending on the quality versus capacity trade-off desirable for the data centre’s workloads. Migration controller thresholds for the ten cases are given in Table 4.

$\alpha$	$\beta$	$\gamma$	$\delta$
99%	99%	60%	80%
90%	95%	60%	80%
99%	99%	50%	80%
90%	95%	50%	80%
90%	90%	50%	70%
99%	95%	40%	80%
85%	99%	40%	80%
99%	95%	40%	60%
99%	90%	30%	80%
99%	99%	40%	40%

Table 4: Migration Controller Thresholds for Ten Pareto-Optimal Cases.

#### 4.5. Performance, Quality, and Power Achieved by Management Policies

We now consider the impact of integrated workload placement and workload migration controller policies for managing the resource pool. The management policies are described in Section 2.3. The management policies are each simulated for the ten Pareto-optimal sets of migration controller threshold values as illustrated in Figure 11. Figures 12 through 14 show simulation results for our baseline cases and the workload management policies we consider. The CPU metrics are discussed first followed by the memory and migration metrics.

The use of a workload migration controller alone policy MC is most typical of the literature [19, 20]. The MC policy does very well as a starting point. Figure 12 shows that when using approximately 8% more CPU capacity than the ideal case there is a CPU violation penalty per hour of 12.6. As the migration controller becomes less aggressive at consolidating workloads, i. e., using 50% more CPU capacity than the ideal case, the penalty drops to nearly zero.

The workload placement controller policy WP doesn’t use the migration controller. It operates with a control interval of four hours and consolidates to a given CPU and memory utilization, which is varied between 75% and 100%. In Figure 12 the 100% is omitted for visibility reasons. It incurred hourly CPU violation penalties of 84. The WP policy does well when the systems are overprovisioned because there is little likelihood of a CPU violation penalty. As the workloads become more consolidated, the CPU violation penalty per hour increases dramatically.

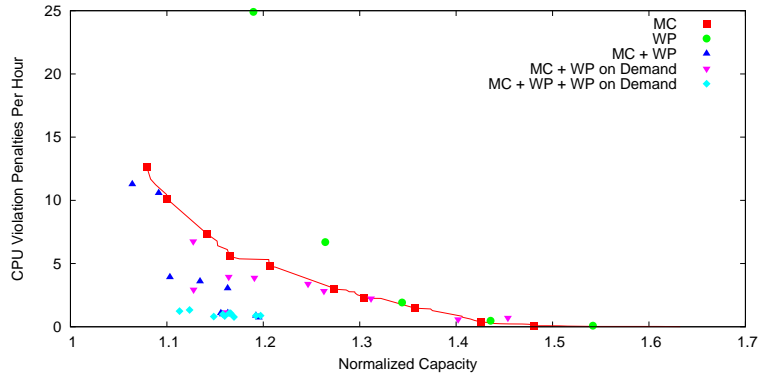


Figure 12: Comparison of Different Management Policies Regarding CPU.

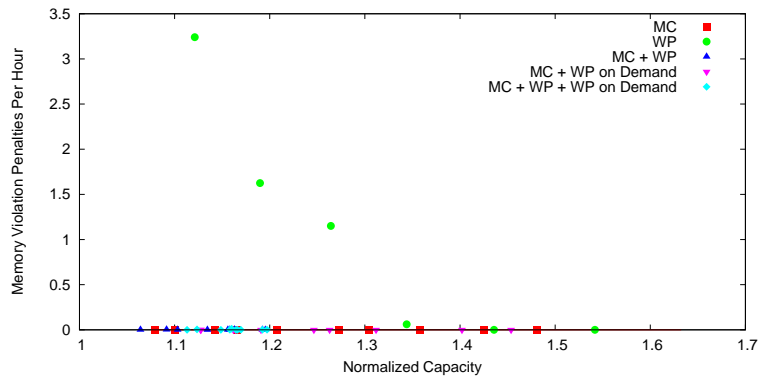


Figure 13: Comparison of Different Management Policies Regarding Memory.

The MC + WP policy is able to achieve much better CPU quality than either MC or WP alone while using much less CPU capacity. The periodic application of the workload placement controller globally optimizes the CPU usage for the resource pool. The migration controller alone does not attempt to do this. This policy and subsequent policies permit the workload placement controller to consolidate workloads onto servers using up to 100% CPU and memory utilization.

The MC + WP on Demand policy invokes the workload placement controller to consolidate the workloads whenever the resource pool is lightly loaded. It behaves better than MC alone but not as well as MC + WP because it does not periodically provide for a global optimization of CPU usage for the resource pool.

Finally, MC + WP + WP on Demand provides very good results from both a capacity and violation penalty point of view. It achieves nearly ideal CPU violation penalties while requiring only 10% to 20% more CPU capacity than the ideal case. We also note that the violation penalties for this approach are less sensitive to migration controller threshold values.

Figure 13 shows the capacity versus quality trade-off for the memory metric. All of the cases provide for very low memory violation penalties except for the WP policy. The WP policy has no ability to react to the case where the demand for memory exceeds

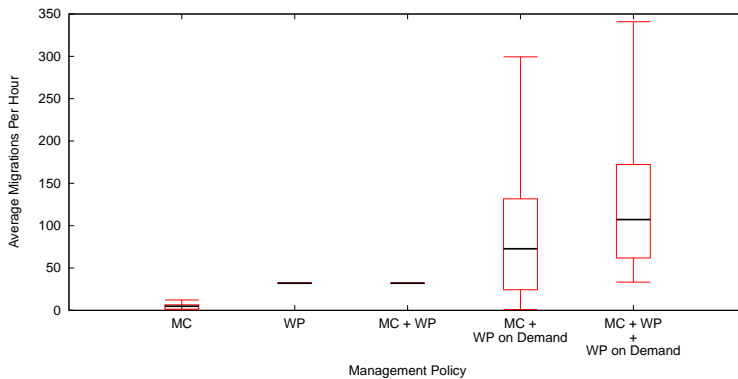


Figure 14: Number of Migrations for all Ten Chosen MC Threshold Cases.

the supply of memory. As a result WP can incur violations with many measurement intervals and hence large violation penalties.

Figure 14 shows the number of migrations for the different policies. For each policy the figure shows the minimum, first quartile<sup>3</sup>, median, third quartile and maximum number of migrations for all ten chosen MC thresholds cases illustrated in Figure 11. The workload placement controller causes more migrations than the migration controller alone. The on-demand policies can cause significantly more migrations when implementing very aggressive migration controller policies. However, these policies also result in the most significant capacity savings with low violation penalties. The next subsection presents the results of a method that reduces the number of migrations caused by a workload placement controller.

#### 4.6. Constraining Migrations For Workload Placement

This section applies a multi-objective approach for the workload placement controller, as described in Section 2.1 to one of the ten MC threshold cases. The approach constrains the number of migrations that the workload placement controller is permitted to recommend. Fewer migrations will cause lower migration overheads but also reduces the opportunity for consolidation. To evaluate the benefits of the approach we compare capacity, quality violations and migrations between the MC policy, which does not use the workload placement controller, and the MC + WP + WP on Demand policy. Figure 15 shows the results.

In the figure, we vary the percentage of workloads that it is desirable for the workload placement controller to migrate from 100%, i.e., no constraint on migrations, down to 5%. The results show that introducing the constraint causes much fewer migrations. Without a limit, the average number of migrations every hour was 116.6 for the on-demand case. This value is nearly 50 times larger than the number of migrations for the MC policy. With a 50% constraint, the migrations per hour drops below 12. With a 15% constraint, the number of migrations drops to 10.5 per hour using slightly less capacity as the MC case and yielding a significantly lower quality violation value. With a 5% constraint, the capacity increases slightly beyond the MC case because there

<sup>3</sup>The first and second quartile refer to the 25 and 50 percentiles.

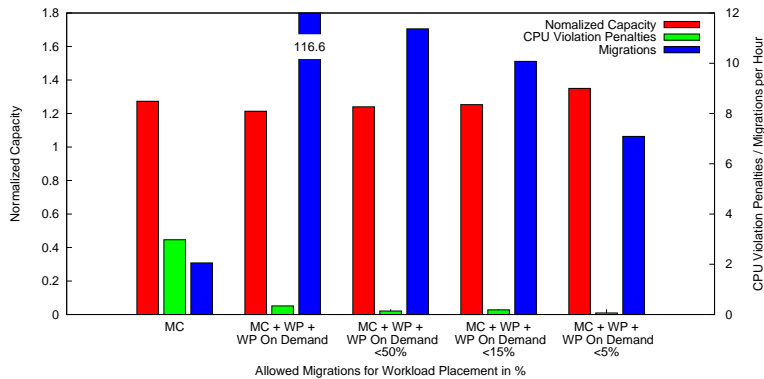


Figure 15: Constrained Migrations

are fewer gains from consolidation but the quality violation value decreases to nearly zero. This is achieved with the average number of migrations per hour being only four times greater than for the MC case. The peak number of migrations per hour for the unconstrained, 50%, 15%, and 5% cases are 1477, 231, 147, and 132, respectively. The peak values are high when the workload placement controller is triggered every five minutes.

## 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 [21, 22]. 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 centre of attention for many research and product groups.

In our work, we chose to represent application behaviour via workload demand traces. Many research groups have used a similar approach to characterize application behaviour and applied trace-based methods to support what-if analysis in the assignment of workloads to consolidated servers [23, 7, 8, 24, 25, 1, 3]. A consolidation analysis presented in [23] 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 [26, 27, 28, 29] 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 addresses issues including classes of

service and placement constraints. The approach is able 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 [24]. 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 [3] rather than making it possible for each workload to have an independently specified objective.

Recently, virtualization platforms such as VMware and Xen [2, 30] 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 [31, 32, 33]. In contrast, we focus on data centre environments with stringent quality of service requirements that necessitate design of highly responsive migration algorithms.

Wood et al. [20] 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 [34] 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.

1000 Islands Project [35] aims to provide an integrated capacity and workload management for the next generation data centres. In the paper, the authors evaluate one loose integration policy for different controllers, while our paper provides a detailed performance study evaluating outcome of the three different integration policies and uses a set of novel QoS metrics. The paper also considers the integration of a per-server workload manager and reports on some real system measurements whereas this paper does not.

Raghavendra et. al. [19] integrates 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 simulation 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. In [19], the authors conclude that 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 QoS 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.

Our approach is not application-centric. Commonly available resource demand traces are the basis for our management system. However, there are many research papers, e. g., [36, 37], which design dynamic provisioning systems for targeted classes of applications, e. g., multi-tier applications. There are excellent earlier works on load sharing systems that support batch-like workloads [38, 39]. These papers argue that simple adaptive techniques outperform static job placement policies and perform nearly as well at improving system performance as more complex optimization methods. Our results confirm these findings for complex enterprise applications with time varying demands and time varying resource pool size. However, for our complex scenario, by integrating two adaptive techniques that operate at different time scales we are able to significantly improve quality measures to a nearly optimal level without increasing the required capacity as compared to using the techniques separately.

This work is relevant to many related efforts on policy-based management. For example, in [40], the authors statically derive and then dynamically refine low-level service level specifications to meet given SLAs while maximizing business profit.

There is a new research direction that has emerged from studying server consolidation workloads using a multicore server design [41, 42]. The authors show, across a variety of shared cache configurations, that a commercial workload's memory behaviour 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

This paper describes an approach for evaluating the impact of policies for resource pool management on required capacity, resource access quality violations, and workload migrations. The evaluation takes into account the impact of different controllers that may operate at different timescales. The approach can be applied to different sets of workloads and different configurations of resource pools. The results can be used to select appropriate policy for a given resource pool scenario. We provide a detailed workload analysis of 138 SAP workloads that operate in an industrial data centre environment. The workloads have sustained bursts in demand that must be taken into account during resource pool management. We propose a resource access quality violation penalty metric that reflects both the duration of violations and the expected impact of the violations on end customers.

Migration and workload placement controllers are studied in detail. A formal analysis is conducted that explores the impact of migration controller threshold values on metric capacity and violation penalty. The analysis quantifies the impact of the threshold values and show the particular importance of underload threshold values. Such results can be used to guide data centre operators in their choice of thresholds.

Over 600 simulation experiments are conducted to assess the impact of combinations of migration controller threshold parameters. We use a Pareto-optimal subset of

these results as a baseline to further evaluate management policies. The set provides a range of quality versus capacity trade-offs that a resource pool operator could choose from. The migration and workload placement controllers are evaluated alone, in parallel, and in an integrated manner. We found that the integrated controllers had the best quality versus capacity trade-off for our resource pool scenario. The tightest integration had the most benefits, but caused a high workload migration rate. Finally, a version of the workload placement controller is employed that minimizes migrations thereby significantly reducing the migration rate with little impact on capacity and quality.

We conclude that a reactive migration controller or proactive workload placement controller alone is not adequate for effective resource pool management. A reactive migration controller does not exploit resource saving opportunities from global optimizations while a workload placement controller is unable to reduce violation penalties caused by bursts in demands and overload conditions between changes in workload placements. In addition, it does not take advantage of short term opportunities to remove servers in the same way as the migration controller does.

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] J. Rolia, L. Cherkasova, M. Arlitt, A. Andrzejak, A Capacity Management Service for Resource Pools, in: Proc. of the 5<sup>th</sup> Int. Workshop on Software and Performance (WOSP), Palma, Illes Balears, Spain, 2005, pp. 229–237.
- [2] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, Live Migration of Virtual Machines, in: Proc. of the 2<sup>nd</sup> Symposium on Networked Systems Design and Implementation (NSDI), Boston, MA, 2005, pp. 273–286.
- [3] S. Seltzsam, D. Gmach, S. Krompass, A. Kemper, AutoGlobe: An Automatic Administration Concept for Service-Oriented Database Applications, in: Proc. of the 22<sup>nd</sup> Int. Conf. on Data Engineering (ICDE), Industrial Track, Atlanta, Georgia, USA, 2006.
- [4] D. Gmach, J. Rolia, L. Cherkasova, A. Kemper, Workload Analysis and Demand Prediction of Enterprise Data Center Applications, in: Proc. of the IEEE Int. Symposium on Workload Characterization (IISWC), Boston, MA, USA, 2007, pp. 171–180.
- [5] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, A. Kemper, An Integrated Approach to Resource Pool Management: Policies, Efficiency and Quality Metrics, in: Proc. of the 38<sup>th</sup> IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN), Anchorage, Alaska, USA, 2008.
- [6] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

- [7] L. Cherkasova, J. Rolia, R-Opus: A Composite Framework for Application Performability and QoS in Shared Resource Pools, in: Proc. of the Int. Conf. on Dependable Systems and Networks (DSN), Philadelphia, USA, 2006.
- [8] D. Gmach, S. Krompass, A. Scholz, M. Wimmer, A. Kemper, Adaptive Quality of Service Management for Enterprise Services, ACM Transactions on the Web (TWEB) 2 (1).
- [9] D. Economou, S. Rivoire, C. Kozyrakis, P. Ranganathan, Full-System Power Analysis and Modeling for Server Environments, in: Workshop on Modeling, Benchmarking, and Simulation (MoBS), 2006.
- [10] L. Kleinrock, Queueing Systems, Volume 1: Theorie, John Wiley & Sons, New York, USA, 1975.
- [11] J. Rolia, Predicting the Performance of Software Systems, Ph.D. thesis, University of Toronto (1992).
- [12] H. Ware, F. Frdrick, Linux Man Page: vmstat(8), <http://linux.die.net/man/8/vmstat> (1994).
- [13] M. F. Arlitt, C. L. Williamson, Web Server Workload Characterization: The Search for Invariants, in: Proc. of the ACM SIGMETRICS Int. Conf. on Measurement and Modeling of Computer Systems, ACM, Philadelphia, PA, USA, 1996, pp. 126–137.
- [14] L. Cherkasova, M. Gupta, Characterizing Locality, Evolution, and Life Span of Accesses in Enterprise Media Server Workloads, in: Proc. of the 12<sup>th</sup> Int. Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), ACM, New York, NY, USA, 2002, pp. 33–42.
- [15] L. Cherkasova, R. Gardner, Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor, in: ATEC '05: Proc. of the USENIX Annual Techn. Conf., USENIX Association, Anaheim, CA, 2005, pp. 24–24.
- [16] D. Gupta, L. Cherkasova, R. Gardner, A. Vahdat, Enforcing Performance Isolation Across Virtual Machines in Xen, in: Proc. of the ACM/IFIP/USENIX 7<sup>th</sup> Int. Middleware Conf., Melbourne, Australia, 2006.
- [17] NetXen, Power and Cost Savings Using NetXen's 10GbE Intelligent NIC, White Paper, [http://www.netxen.com/technology/pdfs/Power\\_page.pdf](http://www.netxen.com/technology/pdfs/Power_page.pdf) (2008).
- [18] R. Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, John Wiley & Sons, New York, NY, USA, 1991.
- [19] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, X. Zhu, No "Power" Struggles: Coordinated Multi-Level Power Management for the Data Center, in: ASPLOS XIII: Proc. of the 13<sup>th</sup> Int. Conf. on Architectural Support for Programming Languages and Operating Systems, New York, NY, USA, 2008, pp. 48–59.

- [20] T. Wood, P. Shenoy, A. Venkataramani, M. Yousif, Black-box and Gray-box Strategies for Virtual Machine Migration, in: Proc. of the 4<sup>th</sup> USENIX Symposium on Networked Systems Design & Implementation, Cambridge, MA, USA, 2007, pp. 229–242.
- [21] HP, HP Virtual Server Environment,  
<https://h30046.www3.hp.com/campaigns/2007/promo/VSE/index.php> (2008).
- [22] J. Jann, L. M. Browning, R. S. Burugula, Dynamic reconfiguration: Basic building blocks for autonomic computing on IBM pSeries servers, IBM Systems Journal 42 (1) (2003) 29–37.
- [23] A. Andrzejak, , M. Arlitt, J. Rolia, Bounding the Resource Savings of Utility Computing Models, Tech. Rep. HPL-2002-339, HP Labs (2002).
- [24] B. Urgaonkar, P. Shenoy, T. Roscoe, Resource overbooking and application profiling in shared hosting platforms, ACM SIGOPS Operating System Review 36, Special Issue: Cluster Resource Management, (2002) 239–254.
- [25] J. Rolia, X. Zhu, M. Arlitt, A. Andrzejak, Statistical Service Assurances for Applications in Utility Grid Environments, Performance Evaluation 58 (2+3) (2004) 319–339.
- [26] VMware, VMWare Capacity Planner,  
[http://www.vmware.com/products/capacity\\_planner/](http://www.vmware.com/products/capacity_planner/) (2008).
- [27] HP, HP Integrity Essentials Capacity Advisor,  
<http://h71036.www7.hp.com/enterprise/cache/262379-0-0-0-121.html> (2008).
- [28] IBM, Tivoli Performance Analyzer,  
<http://www.ibm.com/software/tivoli/products/performance-analyzer/> (2008).
- [29] TeamQuest, TeamQuest – IT Service Optimization,  
<http://www.teamquest.com> (2008).
- [30] VMware, VMware VMotion,  
<http://www.vmware.com/products/vi/vc/vmotion.html> (2008).
- [31] L. Grit, D. Irwin, A. Yumerefendi, J. Chase, Virtual Machine Hosting for Networked Clusters: Building the Foundations for "Autonomic" Orchestration, in: Proc. of the 1<sup>st</sup> Int. Workshop on Virtualization Technology in Distributed Computing (VTDC 2006), IEEE Computer Society, Tampa, FL, USA, 2006, p. 7.
- [32] P. Ruth, J. Rhee, D. Xu, R. Kennell, S. Goasguen, Autonomic Live Adaptation of Virtual Computational Environments in a Multi-Domain Infrastructure, in: Proc. of the 3<sup>rd</sup> IEEE Int. Conf. on Autonomic Computing (ICAC), Dublin, Ireland, 2006.

- [33] A. I. Sundararaj, A. Gupta, P. A. Dinda, Increasing Application Performance In Virtual Environments Through Run-time Inference and Adaptation, in: Proc. of the 14<sup>th</sup> IEEE Int. Symposium on High Performance Distributed Computing (HPDC), 2005, pp. 47–58.
- [34] VMware, VMware Dynamic Resource Scheduler, <http://www.vmware.com/products/vi/vc/drs.html> (2008).
- [35] X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D. Gmach, R. Gardner, T. Christian, L. Cherkasova, 1000 Islands: Integrated Capacity and Workload Management for the Next Generation Data Center, in: Proc. of the 5<sup>th</sup> IEEE Int. Conf. on Autonomic Computing (ICAC'08), Chicago, IL, USA, 2008.
- [36] B. Urgaonkar, P. J. Shenoy, A. Chandra, P. Goyal, T. Wood, Agile Dynamic Provisioning of Multi-tier Internet Applications, *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 3 (1).
- [37] D. Carrera, M. Steinder, I. Whalley, J. Torres, E. Ayguade, Enabling Resource Sharing Between Transactional and Batch Workloads Using Dynamic Application Placement, in: Proc. of the 9<sup>th</sup> ACM/IFIP/USENIX Int. Conf. on Middleware (Middleware), Leuven, Belgium, 2008.
- [38] D. L. Eager, E. D. Lazowska, J. Zahorjan, Adaptive Load Sharing in Homogeneous Distributed Systems, *IEEE Trans. on Software Engineering* 12 (5) (1986) 662–675.
- [39] D. L. Eager, E. D. Lazowska, J. Zahorjan, The Limited Performance Benefits of Migrating Active Processes for Load Sharing, *ACM SIGMETRICS Performance Evaluation Review* 16 (1) (1988) 63–72.
- [40] I. Aib, R. Boutaba, On Leveraging Policy-Based Management for Maximizing Business Profit, *IEEE Trans. on Network and Service Management* 4 (3) (2007) 25–39.
- [41] N. E. Jerger, N. Vantrease, M. Lipasti, An Evaluation of Server Consolidation Workloads for Multi-Core Designs, in: *IEEE Int. Symposium on Workload Characterization (IISWC 2007)*, IEEE Computer Society, Boston, MA, USA, 2007, pp. 47–56.
- [42] M. R. Marty, M. D. Hill, Virtual Hierarchies to Support Server Consolidation, in: Proc. of the 34<sup>th</sup> Annual Int. Symposium on Computer Architecture (ISCA), ACM, San Diego, California, USA, 2007, pp. 46–56.