Cluster Comput
DOI 10.1007/s10586-008-0067-6

1000 islands: an integrated approach to resource management for virtualized data centers

Xiaoyun Zhu · Donald Young · Brian J. Watson · Zhikui Wang · Jerry Rolia · Sharad Singhal · Bret McKee · Chris Hyser · Daniel Gmach · Robert Gardner · Tom Christian · Ludmila Cherkasova

Received: 22 September 2008 / Accepted: 9 October 2008

Abstract Recent advances in hardware and software virtualization offer unprecedented management capabilities for the mapping of virtual resources to physical resources. It is highly desirable to further create a "service hosting abstraction" that allows application owners to focus on service level objectives (SLOs) for their applications. This calls for a resource management solution that achieves the SLOs for many applications in response to changing data center conditions and hides the complexity from both application owners and data center operators. In this paper, we describe an automated capacity and workload management system that integrates multiple resource controllers at three different scopes and time scales. Simulation and experimental results confirm that such an integrated solution ensures efficient and effective use of data center resources while reducing service level violations for high priority applications.

Keywords Data center · Virtualization · Resource management · Control · Optimization · Integration

X. Zhu (✉) · D. Young · B.J. Watson · Z. Wang · J. Rolia · S. Singhal · B. McKee · C. Hyser · R. Gardner · T. Christian · L. Cherkasova
Hewlett-Packard Laboratories, Palo Alto, CA 94304, USA
e-mail: Xiaoyun.Zhu@hp.com

D. Young
e-mail: deyoung@tiworks.com

B.J. Watson
e-mail: brian.j.watson@hp.com

Z. Wang
e-mail: Zhikui.Wang@hp.com

J. Rolia
e-mail: Jerry.Rolia@hp.com

S. Singhal
e-mail: Sharad.Singhal@hp.com

B. McKee
e-mail: Bret.McKee@hp.com

C. Hyser
e-mail: Chris.Hyser@hp.com

R. Gardner
e-mail: Robert.Gardner@hp.com

T. Christian
e-mail: Tom.Christian@hp.com

L. Cherkasova
e-mail: Lucy.Cherkasova@hp.com

D. Gmach
Technical University of Munich (TUM), Munich, Germany
e-mail: daniel.gmach@in.tum.de

1 Introduction

Data centers are inexorably growing more complex and difficult for humans to manage efficiently. Although virtualization provides benefits by allowing consolidation and driving higher levels of resource utilization, it also contributes to this growth in complexity. Data centers may include both hardware- and software-level virtualization, such as HP's Virtual Connect [1] network virtualization technology, as well as the hypervisor-based VMware ESX Server [2], Citrix XenServer [3], Microsoft Hyper-V [4], and Virtual Iron [5] products. Each technology offers different "control knobs" for managing the mapping of virtual and physical resources. Continually adjusting these knobs in response to changing workloads and data center conditions can minimize hardware and energy costs while meeting the service level objectives (SLOs) specified by application owners. This activity should be automated to help avert the coming complexity crisis in data center resource management and more fully realize the benefits of virtualization.

The purpose of our work is to enable both application owners and data center operators to focus on service policy settings, such as response time and throughput targets, and not worry about the details of where an application is hosted or how it shares resources with others. These details are handled by our resource management solution, so that system administrators can "set it and forget it".

This paper describes three key contributions. First, we propose the 1000 Islands solution architecture that supports automated resource management in a virtualized data center. It exploits multiple control knobs at three different scopes and time scales: short-term allocation of system-level resources among individual workloads on a shared server, medium-term live migration of virtual machines (VMs) between servers, and long-term organization of server clusters and groups of workloads with compatible long-term demand patterns. This architecture integrates multiple resource controllers that are designed using different analytical techniques including control theory, bin packing, trace-based analysis and other optimization methods. The innovation is in leveraging each of these independently and then combining their power. Second, we define specific interfaces for coordinating the individual controllers at run time to eliminate potential conflicts. This includes interfaces for sharing policy information, so that policies do not have to be duplicated among controllers, as well as application resource demands. Finally, we validate the effectiveness of the integrated solution through a simulation study, as well as experimental evaluation on a testbed built from real systems.

Section 2 presents the 1000 Islands solution architecture, and explains how its three controllers are integrated. Section 3 describes the simulation environment and the experimental testbed used to validate the architecture. The performance evaluation results from two case studies are shown in Sect. 4. Section 5 discusses related work. In Sect. 6, we conclude and discuss future research directions.

2 Our solution

The 1000 Islands architecture (shown in Fig. 1) consists of three individual controllers operating at different scopes and time scales:

- On the shortest time scale (seconds), *node controllers* dynamically adjust resource allocations to the virtual machines (VMs) running on each *node* (physical server). Each VM hosts a *workload* (WL), which is an application or its component. The node controller aims to satisfy the SLOs of individual applications in spite of changes in workload demands.
- On a longer time scale (minutes), *pod controllers* manage *pods* (workload migration domains consisting of multiple nodes) by adjusting the placement of workloads on nodes within a pod, in response to changing pod conditions.
- On the longest time scale (hours to days), *pod set controllers* study the resource consumption history of many workloads. This controller determines whether the data center has enough resource capacity to satisfy workload demands, places compatible workloads onto nodes, and groups nodes into pods. A *pod set* can consist of multiple non-overlapping pods.

The next three subsections describe the three individual controllers, and the last subsection presents how the three controllers are integrated.

2.1 Node controller

A node controller is associated with each node in a pod. It manages the dynamic allocation of the node's resources to each individual workload running in a VM. Next, we describe two implementations of the node controller: the HPL node controller and the TUM node controller. The former is used for the measurement testbed and the latter in our workload emulation environment.

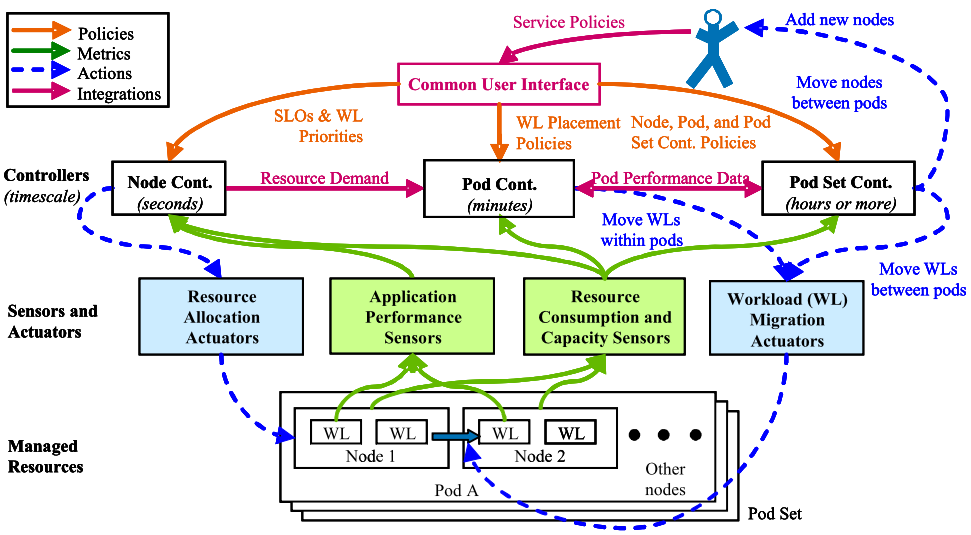

Fig. 1 The 1000 Islands solution architecture consisting of node, pod, and pod set controllers

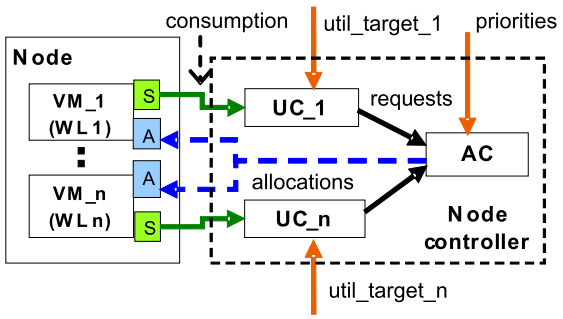

Fig. 2 Node controller architecture

HPL Node Controller The HPL node controller consists of two layers: a set of utilization controllers (UCs) for the individual VMs, and an arbiter controller (AC) for the node. Figure 2 shows this layered structure. For this implementation, we use the term *resource* to refer to a specific type of system resource (e.g., CPU) on a node, although the algorithm can be generalized to handle multiple resources (such as memory, disk I/O or network bandwidth) at the same time.

A utilization controller collects the average resource consumption of each VM from a sensor (see "S" in Fig. 2), and determines the required resource allocation to the VM such that a specified utilization target can be achieved. This is done periodically with a control interval of seconds. We define a VM's *utilization* as the ratio between its resource consumption and resource allocation. For example, if a VM's measured average CPU consumption in a control interval is 0.3 of a CPU, and the specified utilization target is 75%, then the utilization controller will drive the CPU allocation for the VM towards 0.4 of a CPU in subsequent control intervals. The utilization target for a workload is driven by its need to meet its application-level SLO. For example, in order for an interactive Web application to meet an average response time goal of 1 second, the average CPU utilization of the VM running the application may have to be maintained at 60%. A feedback controller was presented in [6] to automatically translate application-level SLOs to VM-level utilization targets. This enhanced controller will be integrated into the 1000 Islands architecture later.

All utilization controllers feed the desired resource allocation for each VM (referred to as a *request*) into the arbiter controller, which determines the actual resource allocations (referred to as an *allocation*) to the VMs. If the sum of all the requests is less than the node's capacity, then all the requests are granted. In addition, the excess capacity is distributed among the VMs in proportion to their requests. On the contrary, if the sum of all the requests exceeds the node's capacity, the arbiter controller performs *service level differentiation* based on workload priorities defined in service policies by data center operators. In our current implementation, each workload is assigned a priority level and a weight within that level. A workload with a higher priority level always has its request satisfied before a workload with a lower priority level does. Workloads at the same priority level receive a percentage of their requests in proportion to their weights. Finally, the allocations determined by the arbiter controller are fed into the *resource allocation actuators* (see Fig. 1 or "A" in Fig. 2) for the next control interval.

TUM Node Controller The TUM node controller is used in the trace-driven emulation environment. The time scale it emulates depends on the sampling interval used for collecting the traces, in this case, five minutes. The controller emulates the relationship between resource utilization and allocation as described above. Utilization targets for CPU and memory are used to scale resource demands in traces to allocations. The arbiter operates on the allocation values. The assumption is that a real node controller, such as the HPL node controller, that operates at shorter timescales would yield allocation requests that approach the TUM controller's allocation estimates when observed at the five minute timescale. Unsatisfied allocations are carried forward to the next control interval.

2.2 Pod controller

The pod controller provides automatic, policy-driven rearrangement of workloads within a pod in response to changes in workload demands and infrastructure. Its primary purpose is to react to aggregate resource requests exceeding a node's capacity, exploiting the fact that they will rarely exceed total pod capacity. The pod controller uses live migration of VMs [7] to move workloads between nodes to mitigate node overload.

In our experiments, moving a 512 MB VM takes slightly over one minute from migration initiation to completion, with only sub-second actual downtime. This makes live migration effectively transparent to the workload inside the migrated VM, though the nodes experience transient CPU and LAN overheads. Similar to the node controller, we use the HPL pod controller for the measurement testbed and the TUM pod controller for the workload emulation environment.

HPL Pod Controller The HPL pod controller [8] consists of a simulated annealing algorithm that periodically searches for VM to node mappings in accordance with a node overload avoidance and mitigation policy. Sensor information on each VM's resource consumption and workload placement policies specified by data center operators, such as consolidation to the fewest physical nodes, are included in the fitness function of the SA algorithm that is used to guide the search for potential mappings. Candidate mappings are generated by modeling the effects of a succession of random VM migrations, and are evaluated using a cost

function that penalizes mappings that lead to overload conditions on any node. A node is defined as overloaded when the total CPU request or memory consumption of its VMs, plus the hypervisor and the DOM-0 overheads, exceeds the available capacity of the node. Mappings with some headroom are favored to avoid overload or SLO violations. To mitigate overloads that do occur, mappings with fewer high priority VMs per node enable more effective service level differentiation by the node controller. This is done with a nonlinear penalty on the count of high priority VMs per node. The best mapping is turned into a migration list and fed into the *workload migration actuators* (see Fig. 1).

This pod controller implementation has the following three advantages. First, the optimization approach used, in contrast with [17], simplifies adding new constraints such as those representing cooling efficiency of different nodes, into the problem. The objective then is to minimize the total weighted penalties associated with soft to semi-hard constraint violations and maximize weighted rewards associated with desirable arrangements. A unique element of this algorithm is a penalty for the overhead of live migration. This can be a function of prior migration history to provide critical hysteresis, such that minor changes in metrics will only infrequently trigger migrations thus permitting an occasional minor optimization.

Second, stability is a fundamental consideration, and is defined here such that bounded input changes must result in bounded controller reactions. Migration of a running VM can have transient effects on that machine's performance (e.g., catching up on queued, re-tried network packets), as well as overhead on the source and target nodes during the migration. The combination of migration cost hysteresis and a transient-damping settling time prevent bounded input changes from resulting in unbounded migrations. With migrations taking approximately one minute from start to completion, a one minute settling time is used.

A third advantage of this algorithm is the creation and execution of a migration plan whenever an atomic set of multiple migrations are deemed desirable. For example, a failure in the cooling system or large changes in several VM workloads could trigger this response. As part of the selection of a new arrangement, a migration plan is created and evaluated, with migrations parallelized and strictly ordered as needed to transiently satisfy or minimally violate constraints and policies.

Note that what would be considered hard constraints in a typical optimization problem are treated as *soft constraints*, and handled using proportional penalties in the objective function. This allows the algorithm to find "less bad" arrangements in response to unplanned events like infrastructure failures.

TUM Pod Controller The TUM pod controller [9] uses a fuzzy logic feedback control loop. It continuously monitors the nodes' resource consumptions for values that are too low or too high. In our experiments, a node is overloaded whenever its CPU or memory consumption exceeds 99% or 95%, respectively. Furthermore, a pod is lightly utilized if the average CPU or memory consumption of all nodes drops below 40% or 60%, respectively. Our justification for these thresholds is beyond the scope of this paper, but appears in related work [10]. After detecting a lightly utilized or overloaded situation, the pod controller identifies actions to remedy the situation, considering the load situation of all affected nodes and workloads. If a node is overloaded, it first determines a workload on the node that should be migrated, and then searches for another node to receive the workload. These rules also initiate the shutdown and startup of nodes to help reduce power usage within a pod.

2.3 Pod set controller

A pod set controller determines whether a data center pod set has enough resource capacity to satisfy all workloads, and periodically determines compatible sets of workloads to place onto nodes within each pod. Our pod set controller [11] supports capacity planning for pod sets, as well as objectives that consolidate workloads to a small number of nodes or balance workloads across nodes. To accomplish this, it studies the historical resource demands of each workload and assumes that future demand patterns will be similar to past demand patterns. The pod set controller simulates the future resource demand behavior of alternative workload placements and uses an optimization heuristic to determine a placement expected to take best advantage of statistical multiplexing among time-varying workload demands.

2.4 Controller integration

One of our contributions is identifying necessary interfaces between these three controllers, so that they can work well together to achieve fully automated capacity and workload management at a data center scale. The red arrows in Fig. 1 indicate these integration points. First, the node controllers must provide estimated resource demands to the pod controller. Otherwise, the pod controller might estimate resource demands that do not agree with the node controllers. If the pod controller's estimates are too low, then it will pack too many workloads onto a node, possibly causing application-level SLO violations. On the other hand, estimates that are too high could trigger an excessive number of overload conditions, and would reduce the power savings that could be achieved by consolidating workloads onto as few nodes as possible.

Second, the pod controller must provide pod performance data to the pod set controller so that the latter can improve

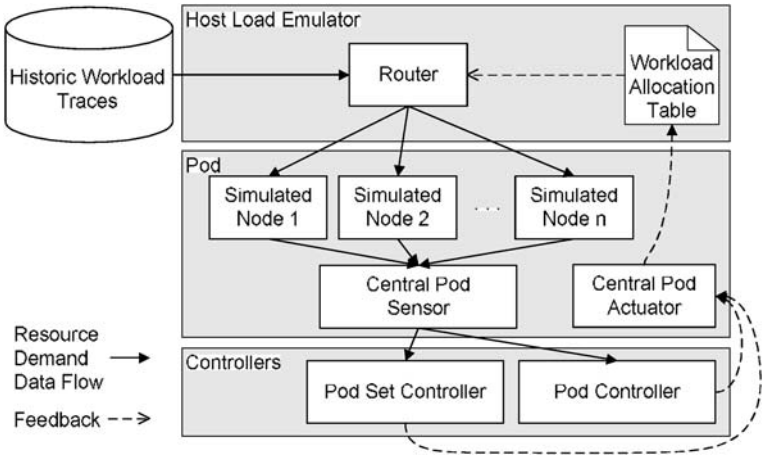

Fig. 3 Simulation environment setup

the compatibility of workloads it places in each pod, and react to pod overload or underload situations by adding or removing nodes. Third, the pod set controller should provide hints to the pod controller about what will happen in the near future. If a workload's resource demand is expected to increase significantly at a particular time, then the pod controller can prepare in advance by placing that workload on a lightly loaded node. Finally, all three controllers must be configurable through a single user interface, and they must consider the other controllers' configuration parameters. For example, the pod controller needs to know the workload priorities used by the node controllers, so that it does not group too many high priority workloads onto the same node, thus preventing effective service level differentiation. When properly integrated, these controllers automate resource allocation and hide the complexity of resource management from data center operators.

3 Validation of the solution

In order to validate the design of the proposed architecture and to demonstrate the merits of the integration approach, we have built both a host load emulator and an experimental testbed to perform workload consolidation case studies using real-world resource consumption traces from enterprise data centers. For the work described here, we have used the experimental testbed for evaluating the integration of the HPL pod and node controllers in a small-scale pod, and the emulator for evaluating the integration of the pod set and TUM pod controllers in a large-scale pod with a larger number of workloads. This section describes the setup of these two environments.

3.1 Host load emulator

Predicting the long term impact of integrated management solutions for realistic workloads is a challenging task. We employ a simulation environment to evaluate a number of management policies in a time effective manner.

The architecture for the host load emulator is illustrated in Fig. 3. The emulation environment takes as input historical workload resource consumption traces, per-workload utilization targets for CPU and memory, per-workload priority and weight, node resource capacity descriptions, pod descriptions, and the management policy. The node resource capacity descriptions include numbers of processors, processor speeds, and physical memory size. A routing table directs the historical time-varying resource consumption data for each workload to the appropriate simulated node, which then determines how much of its aggregate workload demand can be satisfied and shares this time varying information through the *central pod sensor*. The management policy determines how controllers are invoked. Pod and pod set controllers periodically poll the sensor and decide whether to migrate workloads from one node to another. Migration is initiated by a call to the *central pod actuator*. In our emulation environment this changes the routing table and adds an estimated migration overhead to both the source and destination nodes for the duration of the migration.

Our emulator gathers various statistics, including the frequency and length of CPU and memory saturation periods, node capacity used in terms of CPU hours, and the number of workload migrations. Different controller policies have different behaviors that we observe through these metrics.

3.2 Experimental testbed

Our experimental testbed consists of four VM hosts, as well as several load generator and controller machines, all interconnected with Gigabit Ethernet. Each VM host is an HP Proliant server consisting of dual 3.2 GHz Pentium D processors with 2 MB of L2 cache, 4 GB of main memory, and SLES 10.2 with a Xen-enabled 2.6.16 kernel. Storage for the VMs is provided by an HP StorageWorks 8000 Enterprise Virtual Array, and the nodes connect to the array via

Qlogic QLA2342 Fiber Channel HBA. Each VM is configured with 2 virtual processors and 512 MB of memory, and runs SLES 10.2 for best interoperability with the Xen hosts.

We use an Apache Web server (version 2.2.3) as the test application inside each Xen VM. It serves CGI requests, each doing some random calculation and returning the result in HTML. Eight other physical machines are used to generate workload demands on the VMs. These "driver" machines are mostly dual AMD Opteron servers with 1 MB of L2 cache and 8 GB of main memory, each running Redhat AS4. Each driver machine hosts two instances of a modified version of httperf [12], which can continuously generate a variable number of concurrent HTTP sessions. Each session consists of a series of CPU-intensive CGI requests. In order to reproduce the CPU consumption from the real-world resource consumption traces, we first ran experiments to calibrate the average CPU time used by a CGI request, and then we calculated the CGI request rate to produce a given level of CPU consumption.

The Xen hypervisor interface exposes counters that accumulate the CPU time (or cycles) consumed by individual VMs. The counters are sampled at fixed intervals, effectively yielding a sensor for CPU consumption (i.e., *resource consumption sensor* in Fig. 1). Information on the completed transactions, like URLs and response times, is collected on the client side. Xen also exposes interfaces in Dom-0 that allow run time adjustment of scheduler parameters such as the CPU share for each VM (i.e., *resource allocation actuator* in Fig. 1). In our experiments, we use the Credit Scheduler as the actuator for CPU allocation, operated in the non-work-conserving mode, which means that a VM cannot use more than its share of the total CPU time, even if there are idle CPU cycles. However, as noted earlier in Sect. 2.1, the arbiter in the node controller always allocates all the node's resource capacity to the VMs even if the node is underutilized, so this does not result in unnecessary throttling of the workloads. In addition, this capped mode of the scheduler provides a straightforward guarantee on the CPU time allocated to a VM and provides performance isolation among workloads hosted by different VMs. Live VM migration in Xen uses a bounded iterative pre-copy of VM memory from node to node, followed by a stop and copy of remaining or recently dirtied pages [7]. This increases the time between migration initiation and completion, in favor of minimizing VM down time when network connections might be lost.

4 Results from case studies

The following subsections discuss performance evaluation results from two case studies. In the first study, we used the host load emulation environment to evaluate the pod set, TUM pod, and TUM node controllers. The second case study was done on our experimental testbed using the HPL node and HPL pod controllers.

4.1 Emulation results

In this study, we focus on the use of all three controllers within a single pod. The evaluation used real-world load traces for 138 SAP enterprise applications. The load traces captured average CPU and memory consumption as recorded every 5 minutes for a three month period. Each workload has a utilization target of 66% for CPU to ensure interactive responsiveness and a utilization target of 95% for memory. All workloads have equal priority and weight. The host load emulator walked forward through this data in successive 5 minute intervals. The nodes in the emulated pod had 8 2.93-GHz processor cores, 128 GB of memory, and two dual 10 Gigabit Ethernet network interface cards for network traffic and for virtualization management traffic, respectively.

Migration overheads were emulated in the following way. For each workload that migrated, a CPU overhead was added to the source and the destination nodes. The overhead was proportional to the estimated transfer time based on the workload's memory size and the network interface card bandwidth. In general, we found our results to be insensitive to proportions in the range of 0.2–1. Therefore, we chose a factor of 0.5 of a CPU to be used throughout the transfer time.

Figure 4 shows the results of an emulation where we used the pod set controller alone to periodically rearrange the 138 workloads to minimize the time-varying number of active nodes. For this scenario, we assumed the pod set controller

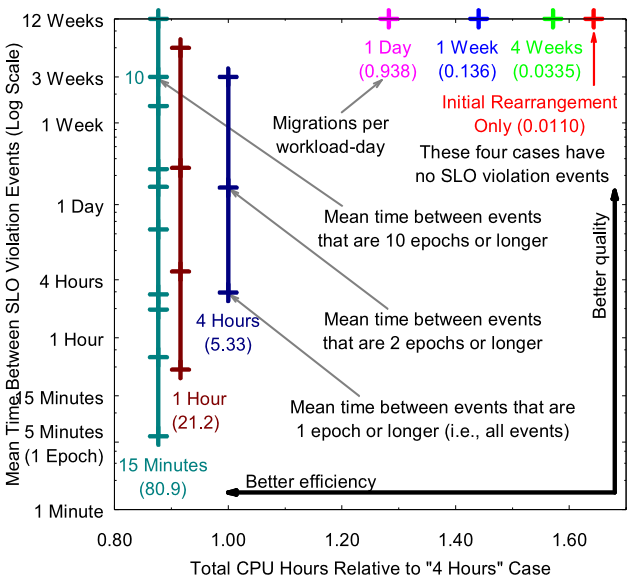

Fig. 4 CPU quality vs. rearrangement periods for pod set controller only (with perfect knowledge of future demands)

had perfect knowledge of the future and chose a workload placement such that each node was able to satisfy the peak of its aggregate workload CPU and memory demands, which gives us a theoretical baseline for comparison with algorithms that have realistic assumptions. Figure 4 shows the impact on capacity requirements of using the pod set controller once at the start of the three months (i.e., Initial Rearrangement Only) and for cases where the workload placement is recomputed every 4 Weeks, 1 Week, 1 Day, 4 Hours, 1 Hour, and 15 Minutes, respectively. The x-axis shows the Total CPU Hours used relative to the 4 Hours case. A smaller value indicates lower CPU usage. CPU Hours includes busy time and idle time on nodes that have workloads assigned to them. The cases with more frequent migrations incur greater CPU busy time, due to migration overhead, but may have lower total time if fewer nodes are needed. Shown in parentheses for each case is the average number of migrations per workload per day. The Initial Rearrangement Only case has migrations only at the very beginning, so it has the smallest count. As the rearrangement period decreases, the migration overhead increases. The figure shows that re-allocating workloads every 4 Hours captures most of the capacity savings that can be achieved. It requires 39% less CPU hours than the Initial Rearrangement Only case (1.00 vs. 1.64) and 22% less CPU hours than rearranging on a daily basis (1.00 vs. 1.28). It uses 9% and 14% more CPU hours than rearranging every hour (1.00 vs. 0.92) and 15 minutes (1.00 vs. 0.88), respectively, but it has much better CPU quality, as we discuss in the next paragraph. That is why we selected the 4 Hours case as our baseline.

Even though we assume perfect knowledge of workload demands, we did not include the CPU overhead of migrations when conducting our workload placement analysis. For this reason, even the ideal cases can have time intervals when a node's CPU is saturated. However, there was no memory overhead for migrations, so there was no occurrence of memory saturation for these cases. Because CPU saturation can result in SLO violations in the hosted workloads, we use the term *SLO violation event* to refer to *a time interval where a workload is allocated less CPU than what it demands*. Figure 4 shows the frequency of SLO violation events using a vertical bar for each case. The y-axis is a logarithmic scale for the mean start-to-start time between SLO violation events, which is calculated by dividing 12 weeks by the number of events. The bottom tick on a bar corresponds to a violation event of one epoch (5 minutes) or longer (i.e., all violation events). Each tick upwards corresponds to two epochs (10 minutes) or longer, three epochs or longer, as so forth.

For the 4 Hours case, there are SLO violation events lasting five minutes or longer every three hours, ten minutes or longer every day and a half, and fifteen minutes or longer every three weeks. This is aggregated over all 138 workloads. One of the ticks in the 15 Minutes case of Fig. 4 is annotated with a "10" to indicate that it corresponds with events lasting 10 epochs (50 minutes) or longer. The careful reader will observe that it is actually the ninth tick from the bottom. This is because this case has no SLO violation events that are 9 epochs long, so the tick for 9 epochs or longer would be in the same position as the tick for 10 epochs or longer.

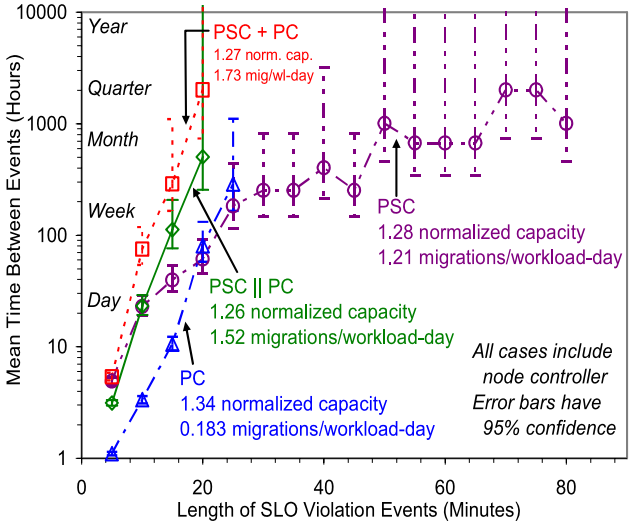

Fig. 5 Emulation results for four different combinations of controller policies

We now describe several policies for integrating the controllers. Note that all of the policies use the TUM node controller. Policy names and descriptions only highlight the relationship between the pod set and pod controllers. Four policies are considered in this study:

- PC: the pod controller is used alone.
- PSC: the pod set controller is used alone.
- PSC||PC: The pod set controller operates periodically with the pod controller operating in parallel.
- PSC + PC: The pod controller is enhanced to invoke the pod set controller on demand to consolidate workloads whenever the servers being used are lightly utilized. This is a tighter integration of controllers than when they operate in parallel.

Figure 5 shows the SLO violation events for these four policies. The events are characterized by two numbers. The x-axis shows the length of violations (in minutes), and the mean start-to-start time (in hours) between events of a given length are on the y-axis, which uses a log scale to better show the dramatically different interarrival times. The data points are discrete, due to the five minute granularity of the emulator, and the points representing events of various lengths for each case are connected by lines with different colors, styles, and markers to guide the eye. Also shown are error bars representing 95% confidence intervals.

The four test cases are indicated by labels, which include the total CPU capacity, normalized relative to the 4 Hours baseline, and the number of VM migrations executed per workload per day by the pod set and pod controllers. For example, the PC policy used 34% more capacity than the baseline, and it migrated each workload 0.183 times per day on average.

The use of a pod controller alone, i.e., PC, is most typical of the literature [13]. Figure 5 shows that the use of a pod controller alone for managing the resource pool incurs a 5 minute SLO violation event approximately each hour on average, and a 10 minute violation about every 3 hours on average. It has a normalized capacity of 1.34, which means that it used 1.34 times as many total CPU hours as Fig. 4's ideal case with a 4 hour placement interval. While these are good results, the remaining policies have much less frequent 5, 10, and 15 minute violations and required lower normalized capacity.

PSC reduced normalized capacity to 1.28 by globally consolidating workloads. However, it is the only policy to have longer SLO violations because it is unable to adapt to SLO violations between its 4 hour pod set controller control intervals. It had SLO violations that were 100 and 200 minutes in length, which were omitted from Fig. 5 for clarity. PSC‖PC overcomes the quality issues while simultaneously reducing normalized capacity. The integrated controller policy PSC + PC improves on quality further. This is clear for 5 and 10 minute SLO violations, and is likely the case for longer SLO violations according to the 95% confidence intervals. However it used slightly more normalized capacity, 1.27 instead of 1.26 for the PSC‖PC case.

We note that the PSC + PC case offered better quality than the 4 hour ideal case from Fig. 4. This is only possible because it uses 1.27 times as much capacity as the ideal case. From detailed results, the ideal case has 5 minute SLO violations every 3.2 hours on average and 10 minute violations every 1.6 days on average. PC had 5 minute violations every 5 hours, but had 10 minute violations every 0.96 days. PSC‖PC had 5 minute violations every 3.1 hours, likely due to increased use of migrations by the pod set controller, and 10 minute violations also every 0.96 days. PSC + PC, the tightly integrated controllers, had 5 minute violations every 5.4 hours and 10 minute violations every 3.1 days. It achieved the best quality while using only slightly more capacity than PSC‖PC but less capacity than PC.

Finally, we note that policies employing the pod set controller cause up to 10 times more migrations per workload day than the PC policy. While this is not an issue for the resource pools we considered, it may be an issue in bandwidth constrained environments.

4.2 Experimental results

Another case study was done on our experimental testbed described in Sect. 3.2 to validate the effectiveness of the integration between the HPL node controller and the HPL pod controller. We ran 16 Apache Web servers in 16 Xen VMs on 4 physical nodes in a pod. The workloads were driven using CPU consumption traces from 4 Web servers, 10 e-commerce servers and 2 SAP application servers from various enterprise sites. For the purpose of this study, we ignore the potential heterogeneity in the servers where the original applications were run, and map CPU consumption of $x\%$ in the original trace to a workload demand of $x\%$ of a CPU on our test server.

The workloads are associated with two classes of service, where eight of them belong to the High Priority- (HP-) class and the other eight belong to the Low Priority- (LP-) class. We start with a semi-random initial placement of workloads, where each node hosts four workloads, two in the HP-class and two in the LP-class. We consider a resource utilization target of 70% and 80% for HP-class and LP-class workloads, respectively, to provide service level differentiation between the two classes. During resource contention on a node, the resource requests of HP-class workloads are satisfied, if possible, before the workloads in the LP-class get their shares.

We compare three workload management policies in this experimental study:

- *Fixed Allocation* (*no control*): Each VM (incl. Dom-0) has a fixed 20% allocation of its node's CPU capacity. There are no VM migrations.
- PC‖NC (*independent control*): The HPL pod and node controllers run in parallel without integration.
- PC + NC (*integrated control*): The HPL pod and node controllers run together with integration.

The first policy is one without dynamic resource control, and it simply provides a baseline for the study. The control intervals for the pod and node controllers are 1 minute and 10 seconds, respectively.

Figure 6 shows a comparison of the resulting application performance from using the three policies. From the client side, a mean response time (MRT) is computed and logged every 10 seconds over the duration of each experiment (5 hours). To better illustrate the results, we consider a 2-s MRT target for the HP-class workloads and a 10-s target for the LP-class workloads, as indicated by the two vertical lines in the figure. For the *Fixed Allocation* policy, the cumulative distribution function (CDF) of the MRT across all 16 workloads is represented by the dashed line in Fig. 6. No class of service was considered in this policy. All the workloads achieve the 2-s target 68% of time and the 10-s target 90% of time. For PC‖NC, or the *independent control policy*, the solid line with a triangle marker represents the CDF of the MRT for the HP-class workloads, and the solid line with an 'x' corresponds to the LP-class workloads. As we can see, the HP-class workloads achieve the 2-s target

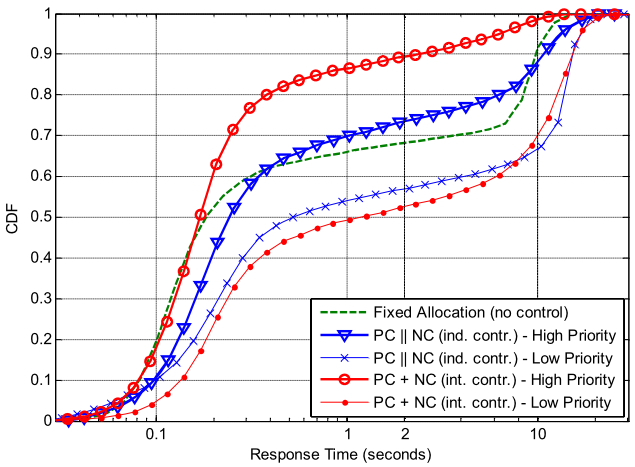

Fig. 6 Cumulative distributions of 10-second mean response times for all the workloads from using three workload management policies—no control, independent control, and integrated control

Table 1 Probability of SLO satisfaction for different response time targets, different workload management policies, and different priority classes

	MRT ≤ 2 s		MRT ≤ 10 s	
	HP	LP	HP	LP
Fixed allocation	68%		90%	
PC‖NC	73%	57%	88%	67%
PC + NC	90%	53%	98%	70%

73% of time (a 5% improvement over Fixed Allocation), but the LP-class workloads achieve the 10-s target only 67% of time. For PC + NC, or the *integrated control policy*, the MRT distributions for the HP-class and the LP-class workloads are represented by the two solid lines with a circle or a dot marker, respectively. We see that the HP-class workloads achieve the 2-s target 90% of time, an improvement of 22% and 17% over the no control and independent control policies, respectively. The relative improvements in these two cases are 32% (22/68) and 23% (17/73), respectively. The LP-class workloads achieve the 10-s target 70% of time, similar to the no integration case. The values of these probabilities are also summarized in Table 1 for easy reference.

To explain the observed difference between the two controller policies, we recall that without integration, the pod controller estimates workload resource demand based on the observed resource consumptions only. In contrast, when the two controllers are integrated, the node controller determines the resource allocation each workload needs to satisfy its performance goal, and this information is provided to the pod controller as an input. The results in Fig. 6 and Table 1 clearly show that this integration enables the pod controller to take into account the performance-driven resource demands of all the workloads, and therefore make

Table 2 Comparison of migration events and unsatisfied demand with and without integration

	No. of migration events		Unsatisfied demand (% of total demand)	
	HP	LP	HP	LP
PC‖NC	17	14	15	12
PC + NC	13	22	9	15

better workload placement decisions such that the HP-class workloads have higher probabilities of achieving their service level objectives.

In addition, we computed the statistics of system-level metrics from the controller logs to see if they demonstrate similar trends as seen in the response time data. Table 2 shows a comparison of the two controller policies in terms of two metrics: the total number of VM migrations that occurred and the total unsatisfied demand (resource request) as a percentage of total demand, for both the HP-class and LP-class workloads. As we can see, the HP-class workloads experienced a smaller number of migrations using integrated control (13) than using independent control (17). This is consistent with our previous explanation that when resource requests are considered instead of measured consumptions, the HP-class workloads are less likely to be migrated and consolidated, leading to better performance. Similarly, using the integrated control policy resulted in a lower percentage of unsatisfied demand (9%) compared to using the independent control policy (15%). Both statistics are consistent with the observed response time data shown in Fig. 6.

In Fig. 7, we demonstrate the impact of controller integration on a particular HP-class workload. The top two figures show the measured CPU consumption, the CPU request computed by the utilization controller, and the actual CPU allocation determined by the arbiter controller for this workload over a 10 minutes interval. In particular, Fig. 7(a) shows the results from using independent node and pod controllers, and Fig. 7(b) represents the integrated control policy. Figure 7(c) shows a comparison of the resulting response times from both control policies.

As we can see, for the independent controllers case, the actual allocation is below the request most of the time (see Fig. 7(a)), causing the VM hosting this workload to be overloaded resulting in a high response time of approximately 10 seconds for most of the 10 minutes interval (see Fig. 7(c)). This is likely due to the pod controller placing too many workloads onto this node causing the shared node to be overloaded. Note that the response time drops off at around 300 seconds, which is due to the reduced resource demand as we can see from Fig. 7(a). In a few sampling intervals that follow, the allocation is below the request but above the average consumption, which means that the VM is less overloaded

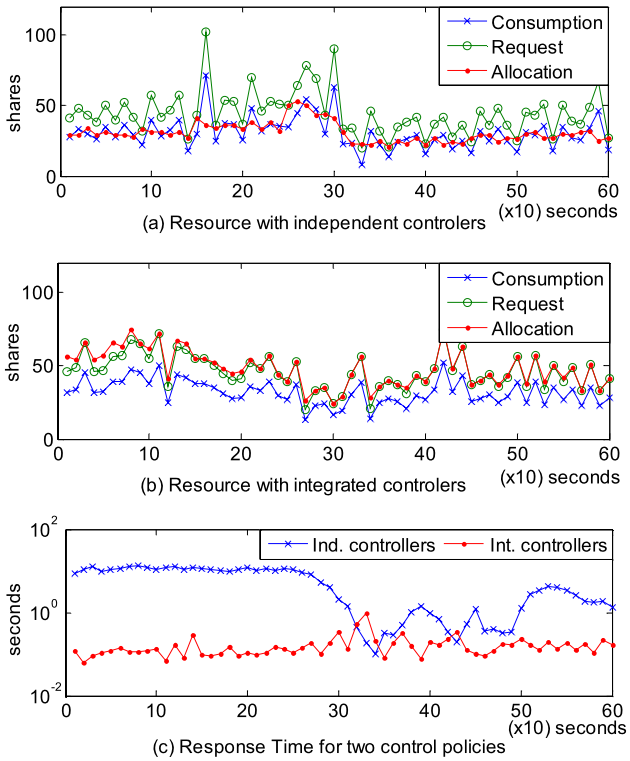

Fig. 7 Time series of the resource consumption, request, and allocation with independent control (**a**) and integrated control (**b**), as well as measured mean response time with both control policies (**c**) for an HP-class workload

and has a much lower response time. With the integrated control policy, the CPU request is satisfied for most of the interval (see Fig. 7(b)). This leads to a much lower response time that remains under 1 second (see Fig. 7(c)). The advantage of the integrated control policy is that it made more informed placement decisions that did not subject this HP-class workload to an overload situation.

5 Related work

VMware's VirtualCenter and DRS products [2] and the management infrastructure from Virtual Iron [5] provide alternatives to parts of our solution. Each offers a degree of pod control for workloads in hypervisor-based VMs. Our approach considers additional metrics, like application service level metrics (e.g., response time and throughput), and uses long-term historical usage trends to predict future capacity requirements. The commercial products could possibly be integrated into our architecture.

Other researchers have studied potential conflicts that can arise when running multiple automation policies independently without coordination. In [14], Kephart et al. studied the scenario where a performance manager that dispatches workloads to a set of blades runs in parallel to a power manager that controls processor frequency and demonstrated that oscillations can occur in both autonomic managers. The paper also showed how this problem can be fixed by explicit communication between the two managers. In [15], Heo et al. identified the incompatibility between a DVFS adaptation policy and a server on/off policy in a server farm when they are not coordinated, and presented a co-adaptation approach that can resolve such conflicts. Neither study dealt with resource management in virtualized data centers as is considered in this article.

Xu et al. presented a two-layered approach to managing resource allocation to virtual containers sharing a server pool in a data center and evaluated the scheme on a testbed running VMware ESX Server [16]. The local and the global controllers together offer a solution similar to the node controller studied in this article, while using fuzzy logic instead of feedback control. The solution does not explore other resource management mechanisms such as workload migration.

Khana et al. solved the dynamic VM migration problem using a heuristic bin-packing algorithm, evaluated on a VMware-based testbed [13]. Wood et al. considered black and grey box approaches for managing VM migration using a combination of node and pod controllers in a Xen-based testbed [17]. They only considered resource utilization for the black box approach, and added OS and application log information for the grey box approach. They found that the additional information helped make more effective migration decisions. Neither work took advantage of long-term demand patterns as we do using the pod set controller.

Raghavendra et al. integrated various sophisticated policies for power and performance management at the node and the pod levels [18]. It presented a simulation study that optimizes with respect to power while minimizing the impact on performance. The simulation results for integrated control suggest that between 3% and 5% of workload CPU demand is not satisfied, but unsatisfied demands were not carried forward between simulation periods. Our host emulation approach carries forward demands and focuses more on the length of events where performance may be impacted.

Control theory has recently been applied to performance management in computer systems [19] through admission control [20, 21] or resource allocation [22–24], including dynamic resource allocation in virtualized environments [6, 25, 26]. Compared with these prior solutions that only dealt with individual non-virtualized or virtualized systems, we have proposed an integrated solution for capacity and workload management in a virtualized data center through a combination of dynamic resource allocation, VM migration, and capacity planning.

6 Conclusion and future work

In this paper, we introduce the 1000 Islands solution architecture that integrates islands of automation to the benefit of their managed workloads, as well as our first steps toward an implementation of this architecture.

While all of the controllers achieve their goals independently using different analytic techniques, including control theory, meta-heuristics, fuzzy logic, trace-based analysis, and other optimization methods, there is power in leveraging each controller independently and then combining them in this unified architecture. In the emulations, the integrated pod set and pod controllers resulted in CPU and memory quality that improved upon the ideal case, while using only 27% more capacity. The testbed showed that the integration of pod and node controllers resulted in performance improvements of 32% over the fixed allocation case and 23% over the non-integrated controllers, as well as reduced migrations for high priority workloads. In addition, service level differentiation can be achieved between workload classes with different priorities.

As a next step, we plan to scale our testbed to a larger number of physical nodes so that they can be divided into multiple pods. This will allow us to evaluate the complete solution architecture that consists of node, pod, and pod set controllers on real systems, as well as study consolidation scenarios with a much larger number of workloads.

We will also integrate with power [18] and cooling [27] controllers, to better share policies and to offer a more unified solution for managing both IT and facility resources in a data center. For example, our node controller can be extended to dynamically tune individual processor P-states to save average power, our pod controller can consider server-level power budgets, and the thermal profile of the data center can guide our pod set controller to place workloads in areas of lower temperature or higher cooling capacity.

Ultimately, data center operators would like application-level service level objectives (SLOs) to be met without having to worry about system-level details. In [28], workload demands are partitioned across two priorities to enable workload-specific quality of service requirements during capacity planning and runtime phases. This can be integrated with node and pod controllers. In [29], application-level SLOs are decomposed into system-level thresholds using performance models for various components being monitored. These thresholds can potentially be used to drive our utilization controllers at the VM level. However, this decomposition is done over longer time scales (minutes). In [6], we have developed a feedback controller for translating SLO-based response time targets into resource utilization targets over shorter time scales (seconds). These approaches can be incorporated in our next round of integration. Finally, a distributed management framework is being developed for integrating all of these components in a scalable manner, such that they can potentially manage a data center of 10,000 nodes.

References

1. HP Virtual Connect Enterprise Manager: http://h18004.www1.hp.com/products/blades/components/ethernet/vcem/index.html
2. VMware ESX Server: http://vmware.com/products/vi/esx/
3. Citrix XenServer: http://www.citrixxenserver.com/products/Pages/XenEnterprise.aspx
4. Microsoft Hyper-V: http://www.microsoft.com/windowsserver2008/en/us/hyperv.aspx
5. Virtual Iron: http://www.virtualiron.com/products/
6. Zhu, X., Wang, Z., Singhal, S.: Utility-driven workload management using nested control design. In: Proc. of the American Control Conference (ACC'06), June 2006
7. Clark, C., Fraser, K., Hand, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live migration of virtual machines. In: Proc. of the 2nd Symposium on Networked Systems Design and Implementation (NSDI'05), May 2005
8. Hyser, C., Mckee, B., Gardner, R., Watson, B.J.: Autonomic virtual machine placement in the data center. HP Labs Technical Report HPL-2007-189, February 2007
9. Seltzsam, S., Gmach, D., Krompass, S., Kemper, A.: AutoGlobe: An automatic administration concept for service-oriented database applications. In: Proc. of the 22nd Intl. Conference on Data Engineering (ICDE'06), Industrial Track, April 2006
10. Gmach, D., Rolia, J., Cherkasova, L., Belrose, G., Turicchi, T., Kemper, A.: An integrated approach to resource pool management: policies, efficiency and quality metrics. In: Proc. of IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'08), June 2008
11. Rolia, J., Cherkasova, L., Arlitt, M., Andrzejak, A.: A capacity management service for resource pools. In: Proc. of the 5th Intl. Workshop on Software and Performance (WOSP'05), Spain, July 2005
12. Mosberger, D., Jin, T.: Httperf—A tool for measuring Web server performance. In: Proc. of the Workshop on Internet Server Performance, June 1998
13. Khana, G., Beaty, K., Kar, G., Kochut, A.: Application performance management in virtualized server environments. In: Proc. of the IEEE/IFIP Network Operations & Management Symposium (NOMS'06), April 2006
14. Kephart, J., Chan, H., Das, R., Levine, D., Tesauro, G., Rawson, F., Lefurgy, C.: Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs. In: Proc. of the 4th IEEE Int. Conf. on Autonomic Computing (ICAC'07), June 2007
15. Heo, J., Henriksson, D., Liu, X., Abdelzaher, T.: Integrating adaptive components: An emerging challenge in performance adaptive systems and a server farm case-study. In: Proc. of the 28th IEEE Int. Real-Time Systems Symposium (RTSS'07), December 2007
16. Xu, J., Zhao, M., Fortes, J., Carpenter, R., Yousif, M.: Autonomic resource management in virtualized data centers using fuzzy logic-based approaches. Cluster Comput. J. **11**, 213–227 (2008)
17. Wood, T., Shenoy, P., Venkataramani, A., Yousif, M.: Black-box and gray-box strategies for virtual machine migration. In: Proc. of the 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI'07), April 2007

18. Raghavendra, R., Ranganathan, P., Talwar, V., Wang, Z., Zhu, X.: No power struggles: Coordinated multi-level power management for the data center. In: Proc. of the 13th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS'08), March 2008
19. Hellerstein, J., Diao, Y., Parekh, S., Tilbury, D.: Feedback Control of Computing Systems. Wiley-IEEE Press, New York (2004). ISBN: 0-471266-37-X
20. Kamra, A., Misra, V., Nahum, E.: Yaksha: A self-tuning controller for managing the performance of 3-tiered web sites. In: Proc. of the Int. Workshop on Quality of Service (IWQoS'04), June 2004
21. Karlsson, M., Karamanolis, C., Zhu, X.: Triage: Performance differentiation for storage systems using adaptive control. ACM Trans. Storage **1**(4), 457–480 (2005)
22. Abdelzaher, T., Shin, K., Bhatti, N.: Performance guarantees for web server end-systems: A control-theoretical approach. IEEE Trans. Parallel Distrib. Syst. **13**, 80–96 (2002)
23. Lu, Y., Abdelzaher, T., Saxena, A.: Design, implementation, and evaluation of differentiated caching services. IEEE Trans. Parallel Distrib. Syst. **15**(5), 440–452 (2004)
24. Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P.: Dynamic provisioning of multi-tier internet applications. In: Proc. of the Int. Conf. on Autonomic Computing (ICAC'05), June 2005
25. Wang, Z., Zhu, X., Singhal, S.: Utilization and SLO-based control for dynamic sizing of resource partitions. In: Proc. of the 16th IFIP/IEEE Distributed Systems: Operations and Management (DSOM'05), October 2005
26. Padala, P., Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., Salem, K., Shin, K.: Adaptive control of virtualized resources in utility computing environments. In: Proc. of the EuroSys'07, March 2007
27. Bash, C.E., Patel, C.D., Sharma, R.K.: Dynamic thermal management of air cooled data centers. In: Proc. of the Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITHERM'06), May 2006
28. Cherkasova, L., Rolia, J.: 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'06), June 2006
29. Chen, Y., Iyer, S., Liu, X., Milojicic, D., Sahai, A.: SLA decomposition: Translating service level objectives to system level thresholds. In: Proc. of the 4th IEEE Int. Conf. on Autonomic Computing (ICAC'07), June 2007

Xiaoyun Zhu is a senior scientist at Hewlett-Packard Labs in Palo Alto, California. Her research interests are in applying control theory, optimization, algorithms, statistical analysis and simulation to IT systems and services management, data center management and cloud computing. She received her Ph.D. in Electrical Engineering from California Institute of Technology in 2000, and her dual B.S. in Automation and Applied Mathematics from Tsinghua University in 1994. She is a member of the IEEE and ACM.

Donald Young has managed HP Labs projects focused on applying formal control theory and machine learning to IT automation. His current interests continue earlier automation and ecommerce themes; this time applying them to the integration of internet business and medical monitoring services. He holds a Masters in Engineering Administration from George Washington University and a BSEE from Northeastern University.

Brian J. Watson is a research scientist in Hewlett-Packard's Sustainable IT Ecosystem Laboratory. His interests include models and controllers for maximizing the utilization of server resources while ensuring that performance, availability, and other requirements are satisfied. He is also interested in how IT can help automate and reduce the resource needs of other sectors of the economy. His diverse career has spanned computer science, aerospace engineering, and physics.

Zhikui Wang received the B.S. degree in Automation in 1995 from Tsinghua University, M.S. degree in Industrial Automation in 1998 from the Chinese Academy of Sciences both at Beijing, China, and Ph.D. degree in Electrical Engineering in 2005 from UCLA, Los Angeles, CA, USA. He is now a research scientist at Hewlett-Packard Laboratories, Palo Alto, CA, USA. His research interests are in the control and optimization of networks, computers and data centers.

Jerry Rolia is a Principal Scientist in the Automated Infrastructure Laboratory of Hewlett- Packard Labs. His research interests include resource pool management, software performance engineering, and utility and cloud computing. Jerry received his Ph.D. from the University of Toronto in 1992, was an Associate Professor in the department of Systems and Computer Engineering at Carleton University in Ottawa, Canada until 1999, and has been with HP Labs since.

Sharad Singhal is a Distinguished Technologist at Hewlett Packard Laboratories, Palo Alto, CA. His current research interests include application of control theory to systems management, policy-based system management, and large-scale management architectures. He received the 2003 Joel S. Birnbaum prize for innovation at HP, and the Harding Bliss prize for his graduate work at Yale University. He obtained his B. Tech degree from the Indian Institute of Technology, Kanpur, and his M.S. and Ph.D. degrees from Yale University. He is a member of the IEEE and the Acoustical Society of America.

Bret McKee Bret McKee is a senior engineer at Hewlett-Packard in Fort Collins, Colorado. His current career interests include virtualization and networking protocols for file sharing. He received B.S. and M.S. degrees in Computer Science from Iowa State University in 1986 and 1988.

Chris Hyser is a senior researcher at Hewlett Packard Labs based in Rochester, New York.

Daniel Gmach is a Ph.D. student at the database group of the Technische Universität München. Before joining the chair in 2003, he studied computer science at the University of Passau. His current research interests are in adaptive resource pool management of virtualized enterprise data centers, performance measurement and monitoring, hosting large-scale enterprise applications, database systems, and software engineering principles.

Robert Gardner has been an engineer with Hewlett Packard for 25 years, with a broad variety of R&D experience. He currently works for HP's Storage Works Division as the team leader for the high availability NFS kernel subsystem. He received his B.S. and M.S. degrees in Computer Science from Clarkson College of Technology (now Clarkson University) in 1983.

Tom Christian is a Principal Scientist in the Sustainable IT Ecosystem Lab at Hewlett-Packard Laboratories and is currently working on the Data Center Synthesizer component of the Sustainable Data Center project. Tom joined Hewlett-Packard in 1978, having previously worked at Ball Aerospace and the University of Colorado at Boulder. While at HP, Tom has contributed to the development of networking, programming language, operating system, artificial intelligence and software engineering products for HP computer systems.

Ludmila Cherkasova is a senior scientist in the Storage and Information Management Platforms Laboratory at HP Labs, Palo Alto. Before joining HP Labs, she was a senior researcher at Institute of Computing Systems, Russia, and adjunct associate professor at Novosibirsk State University. Her current research interests are in distributed systems, internet technologies and networking, performance measurement and monitoring, characterization of next generation system workloads and emerging applications in the large-scale enterprise data centers.