J Supercomput (2014) 68:1088–1112
DOI 10.1007/s11227-014-1141-x

Adaptive global power optimization for Web servers

Leonardo Piga · Reinaldo A. Bergamaschi · Mauricio Breternitz · Sandro Rigo

Published online: 11 March 2014

Abstract This work investigates power and performance trade-offs for Web servers on a state-of-the-art, high-density, power-efficient SeaMicro SM15k cluster by AMD. We relied on the concept of virtual power states (VPSs), a combination of CPU utilization rate to the P/C power states available in modern processors, and on our global optimization algorithm called *Slack Recovery*, to deploy an adaptive global power management system in a production environment. The main contributions of this paper are twofold. First, it presents the *Slack Recovery* algorithm deployed on a real cluster, composed of 25 SeaMicro nodes. The algorithm finds a P-state and a utilization rate for each CPU node to minimize power under a minimum performance requirement. Second, it proposes a novel mechanism to control utilization rates in each server, a key aspect on our power/performance optimization system which enables the implementation of the VPS concept in practice. Experimental results show that our *Slack Recovery*-based system can reduce up to 6.7 % of the power consumption when compared to policies usually deployed in SeaMicro production systems.

L. Piga (✉) · R. A. Bergamaschi · S. Rigo
Institute of Computing, University of Campinas (UNICAMP), Av. Albert Einstein, 1251, Cidade Universitaria, Campinas, SP 13083-852, Brazil
e-mail: lpiga@ic.unicamp.br

R. A. Bergamaschi
e-mail: rberga@ic.unicamp.br

S. Rigo
e-mail: sandro@ic.unicamp.br

M. Breternitz
Advanced Micro Devices, 7171 Southwest Pkwy, Austin, TX 78735, USA
e-mail: Mauricio.Breternitz@amd.com

Keywords Power management · High-density servers · Web server · Power optimization · Cluster

1 Introduction

The shift towards increasing demand in computational resources has forced companies to build facilities hosting hundreds of thousands of computers called data centers. A variety of services, from search (e.g., Google and Yahoo), to e-commerce (e.g., Amazon and e-Bay), to stock trading (e.g., Fidelity and e-Trade) and media streaming (e.g., YouTube) rely on the computing capabilities of data centers. This comes at a price of higher operational costs, which include the management of these installations, the electricity costs, and environmental impacts due to the increased power consumption.

The power associated with the IT equipment in a data center includes the power of the servers, the power required by the cooling, and auxiliary equipment (e.g., power distribution units, switching, back up power). The power consumption of the actual server drives the power needs for auxiliary equipment and cooling; thus, reducing server power has a direct effect on reducing data center power as a whole. Furthermore, saving energy implies in reducing costs as the total cost of ownership of a data center is proportional to its power consumption.

Data centers efficiency can be measured using a metric called power usage effectiveness (PUE), which is calculated by dividing the total electric power used in a facility by the power brought to the computing units (e.g., computers, networking equipment). In 2009, Schulz claimed that 50 % of electrical power was spent on cooling in a typical data center [31]. By 2006, the PUE of 85 % of data centers was about 2, that is, for each watt used for computation, other additional two watts were spent by cooling and auxiliary equipments [24]. By the end of 2010, Google's data centers achieved an overall PUE of 1.1 [30], a huge improvement.

In 2005, the energy consumption of total servers corresponded to about 0.6 % of total electricity consumed in the USA. If auxiliary equipment was taken into account, this share increases to 1.2 % [20]. In 2010, the electricity used in US data centers accounted for between 1.7 and 2.2 % of total electricity use [21].

In this context, power-aware computing has emerged as a concern in data centers and, in this article, we follow-up on the work introduced in Bergamaschi et al. [4], which presented the theoretical concept of virtual power states (VPSs), a combination of utilization rate to the P/C power states available in modern processors, and the *Slack Recovery* Algorithm. Now, we develop power optimization techniques that can show significant improvements over best-of-breed, production-level dense data centers, using the latest CPU architectures. We deploy them to a modern cluster architecture composed of 25 nodes of SeaMicro servers [3], a state-of-the-art server architecture by AMD. In order to do this, we develop a novel mechanism to control utilization rates in each server, a key aspect for our power/performance optimization heuristic.

The main focus of this article is to show how a *Slack Recovery*-based system can be deployed in practice using a production, state-of-the-art, SeaMicro SM15k cluster, instead of simulation. It shows how a utilization rate control mechanism can

be integrated to such a system. We compare our result to policies usually deployed in SeaMicro production systems (i.e., the Linux governors [7,26]). The reader may find more details about the optimization problem in our previous work [4]. To the best of our knowledge, this is the first work to implement and evaluate power/performance optimization algorithms in such high-density cluster architectures (e.g., the SeaMicro SM15k). We believe the results are general and applicable to a wide range of data center architectures.

This paper is organized as follows: Sect. 2 presents the related work and highlights the main differences of our work. Section 3 gives an overview of the SeaMicro cluster architecture used in this work; Sect. 4 introduces basic concepts on CPU power and performance trade-offs. Section 5 presents the overall organization of our approach and describes our experimental methodology, including the benchmarks used and the algorithms. Section 6 explains how the theoretical concept of VPS is deployed in the cluster. Section 7 presents the results and, finally, Sect. 8 draws our conclusions.

2 Related work

Over the last decade, there have been several research approaches tackling the problem of power and performance trade-off and optimization in processors and data centers. Most of these works presented algorithms for either turning processors on and off, or applying dynamic voltage and frequency scaling (DVFS). We split the related work into six categories that are explained next.

Server provision Filani et al. [15] used a closed-loop control algorithm to perform power management, by acting on processor states and memory, to comply with certain policy directives dictated by the upper level software. Their work focuses on increasing the compute density by changing the server provision policy.

Performance optimizations under a power cap Other works intend to maximize performance, given a power cap. Rajamani and Lefurgy [28] investigated the problem of scheduling service requests among servers in a cluster to minimize energy consumption known as Power-Aware Energy Distribution (PARD). The authors evaluated the influence of the system-workload context on energy-saving schemes using a simple on/off model for estimating the energy consumption of the cluster. In Chen et al. [9], on/off node optimizations were applied in a multiple-application data center. The goal was to determine how many servers and their operation frequencies should be used for each application. All servers ran the same application at the same frequency.

Kant et al. [19] developed a simple task model based on QoS requirements and presented Willow, a simple adaptive control scheme for energy-adaptive computing (EAC) that considered power and thermal constraints simultaneously. They discussed three scenarios for applying their model: (1) Cluster EAC, where clients submit requests that required significant computation on the cluster side; (2) Client–Server EAC, where observing QoS was an important requirement; and (3) Peer-to-Peer EAC, where devices changed information through a network.

Winter et al. [35] presented scheduling and power management algorithms for heterogeneous many-core architectures. Cochran et al. [10] presented a control technique to choose CPU voltage and frequency states in an optimal way to maximize

performance under a power budget. Shen et al. [32] developed an operating system feature that enabled request level power management. Meisner et al. [25] evaluated the effects of switching CPU states to idle states for a short period of time on online data-intensive services (e.g., Web search). The problem was more complicated than other Web workloads, since it required fast response time and the data were distributed across the nodes unabling nodes shutdown.

Microarchitectural power optimizations The in-core power management algorithms in Isci et al. [18] dynamically control the processor parameters (frequency, voltage and fetch throttling) to optimize power and/or performance according to on-the-fly workloads. They report up to 38 % of power savings with 17.7 % in performance degradation.

Leverich et al. [23] point that per-core power gating can be used as an additional mechanism on multi-core processors. They show that by using their technique they can reduce energy consumption in up to 40 %, maintaining the same performance level. They also combined DVFS and per-core power gating and reported energy savings of almost 60 %.

Local optimizations Linux has some built-in performance and power algorithms, which are readily available on Linux and used in production systems. For this reason, we consider them as a good baseline to our technique. One of them is *performance* governor [7], where the CPU frequency is kept at the maximum when the CPU is executing any instruction. When in idle, it is automatically put in power-safe states (also known as C-States) by the CPU hardware [29].

Other algorithm is the Linux *ondemand* governor [7,26]. The governor samples the CPU usage at small intervals (typically 10 ms) and decides which frequency to set the CPU based on its utilization rate. When the CPU utilization rate is greater than the threshold (typically 95 %) in an interval, the core frequency is set to the maximum frequency. When the utilization is lower than the threshold, the core frequency is adjust to a frequency state that is able to keep the CPU utilization rate at 80 %.

Server-level optimizations A survey of power management techniques developed to explore and optimize the power/performance trade-off in data centers can be found in Bianchini and Rajamony [6].

The work in Chase et al. [8] developed an agent-based approach, implemented in the middleware, to turn off servers under low-load conditions while maintaining the expected (or contracted) service level agreements (SLAs). Their approach is based on turning servers on and off and distributing the work among the powered-up servers. They could reduce energy in up to 29 % for a Web workload from the 2000s.

In Kusic et al. [22], dynamic resource provisioning is used in a virtualized computing environment to reduce power consumption while maintaining SLA. This work accounts for the switching costs incurred while provisioning (turning on/off) virtual machines and explicitly encodes the corresponding risk in the optimization problem. They report 26 % of power savings.

In Elnozahy et al. [12], five cluster-wide power management policies are evaluated. The authors apply DVFS and node on/off techniques to reduce power consumption during periods of reduced workload. The policies assume that the workload is balanced across cluster nodes. Policies include: independent voltage scaling, where each node manages its own power consumption; coordinated nodes' voltage scaling actions;

turning nodes on/off, so that the minimum number of servers required by the workload is kept active; and combinations of these techniques. The authors concluded that between 33 and 50 % of cluster energy can be saved by applying the combined policy, when compared to a cluster that is not power managed.

In their follow-up research [13], the authors use DVFS and request batching (where the servicing of incoming packets is delayed until a specified batching timeout is reached) management mechanisms to propose three policies to reduce energy consumption in Web servers. They show that DVFS is better suited for moderately intense workloads, while batching is better for low-intensity workloads. They also propose a combined policy that reaches 17 to 42 % energy savings in all workloads, compared to a base model with no optimization. A feedback-driven control framework is used to adjust policy parameters.

Bertini et al. [5] present an optimal linear programming-based solution to the problem of dynamic cluster configuration combined with the use of feedback control theory to control the quality of service (QoS) and dynamically select which servers turn on/off or their operating frequencies. Two control theory schemes are compared, single-input single-output (SISO) controller and single-input multiple-output (SIMO) controller. The authors show that the SISO approach does not scale as an online solution, so they apply a table-based off-line solution. On the other hand, the SIMO approach runs with N-independent controllers, at a cost of loss of optimality. They report 40 % in power reduction.

Abbasi et al. [2] propose TACOMA (two-tier architecture for cooling-computing energy managements), a two-tier Internet data center scheme. The first tier adjusts the number of active servers; the second tier predicts the workload arrival rate. The algorithm evaluation is based on web traces and they reported energy savings of up to 40 % considering compute and cooling power.

Several of these works reported significant savings (30 % or more). Their contributions were undoubtedly relevant for the CPU architectures and data center power management techniques at the time. However, the state-of-the-art today is very different. CPU power (both active and idle) in modern architectures (e.g., Intel Ivy Bridge) is considerably lower than it was 5 years ago; thus, reducing significantly the gains that simple on/off or DVFS techniques can achieve.

The approach in Bergamaschi et al. [4] used a simulation model of a data center to evaluate the algorithms. Several practical implementation aspects are simplified in a simulation environment. In this work, we implemented everything in an actual cluster; thus, demonstrating not only that the approach is scalable, but can achieve significant savings even considering the extremely power-efficient baseline starting point, such as the SeaMicro cluster using Intel's Xeon processors (Ivy Bridge architecture), running Linux *ondemand* governor power management.

3 SeaMicro cluster overview

We deploy our Power and Performance Optimizations on an AMD's SeaMicro SM15000 (Fig. 1a) family of Fabric Compute Systems (SM15k) [3]. The SM15k is a high-density cluster composed of compute nodes, networking, and storage on a single

Fig. 1 AMD's SeaMicro SM15000 family of fabric compute systems. **a** SM15k front picture. **b** Freedom-based motherboard

10 Rack Unit (RU). This system amortizes the power overhead of fixed system components such as power supplies and fans among the cluster by sharing them among the cluster nodes. Our SM15k has 64 server cards and consumes between 3.0 and 3.5 kW.

A server card is logically described in Fig. 1b. Each server card is composed of DRAM, CPU+Chipset, and the Freedom ASIC. The latter includes network interfaces, which removes the need of network adapters, cables, and switches, resulting in a high-density and energy efficient cluster. It also implements an I/O virtualization technology, which virtualizes the disks to the server nodes. Each node accesses the disk as a virtual SATA disk. This feature reduces power and space without requiring any special software and driver. In our configuration, we used server cards composed of one Intel Xeon E3-1265Lv2 processor, 32 GB ECC DRAM, 8 $\times$ 1 Gbit network interface card, connected with one virtual SATA disk. Each virtual disk was assigned to one physical disk.

The Freedom ASIC enables interconnection of servers in a 3D torus topology with 1.28 Tbit bandwidth and less than 6 μs communication delay between any two server nodes. Our SeaMicro SM15k processors are from the Intel Ivy Bridge family, and feature special registers that provide estimates to the CPU power consumption. We took advantage of the fast interconnection and the embedded power estimators when implementing our power and performance optimization algorithms.

4 Background on power and performance trade-offs

In our previous work [4], we used Integer Linear programming (ILP) to find an optimal solution to the problem of minimizing power by selecting discrete frequencies and voltage levels (P-states) while maintaining performance above a minimum threshold. This approach, however, did not scale for large number of nodes, because of the complexity of the ILP problem. To overcome this limitation, we also developed a heuristic algorithm called *Slack Recovery* and implemented it on the simulation environment.

In this work, we took the *Slack Recovery* algorithm and adapted it to a real cluster environment. We used it to select the P-states in each CPU node to optimize the overall Web server cluster power, while satisfying a given level of total performance (i.e., a performance threshold).

Current processor cores, such as the Intel Xeon E3-1265Lv2, have internal mechanisms in hardware and firmware to change its P/C-state according to its load and power. According to ACPI tables found in operating systems, Intel Xeon E3-1265Lv2 has 11 P-states, or 11 different levels of frequency and voltage.

In our implementation, we are considering only the CPU power, since it is the most important share of the power consumption. Moreover, it is the only device in our system that provides mechanisms to trade-off power and performance, such as DVFS. We based our choice on a characterization of Web servers done previously, which found that disk, memory, and network power components do not vary too much comparing to the CPU when a Web server workload is running; therefore, they can be considered as a constant value [27] for the purpose of power optimizations. These findings are also corroborated by Economou et al. [11] who showed that disk, network, disk, and memory have about the same power consumption when running Web applications (i.e., SPECweb) and when in idle.

In the SeaMicro environment, the disk and the network cards are virtualized. This means that they are shared by computing nodes using the SeaMicro ASIC, which amortizes their share of the total power. Power supplies and fans are also shared. As a result, their power component is lower than on regular servers. Even though memory power is an important component with a share of 20 to 25 % server power consumption, power reduction on DRAM has been targeted only on the DRAM device or compute system (e.g., reduce refresh rate and refresh power) [34] lacking power-aware mechanisms, such as DVFS, whose *Slack Recovery* is based on.

Intel Xeon E3-1265Lv2 processors feature Running Average Power Limit (RAPL) interfaces [17,29] which, among other capabilities, provide a power metering interface. We developed a Linux kernel module that reads these power registers and provides a power estimate for the CPU. According to Hackenberg et al. [16], RAPL interfaces are suitable for low-resolution power consumption measurements, which is the case of this paper.

Our performance metric is billions-of-instructions-per-second ($BIPS$). We developed an additional piece of software that uses libpfm-4.3 [1], to monitor the CPU performance counters, allowing the measurement of number of instructions executed as well as user, system, I/O, and idle times.

In our optimization problem, we observed that the CPU presents different power consumption and different performance levels (measured in $BIPS$), under different utilization rates. We define utilization rate as the ratio of the time that the CPU is doing useful work (i.e., the CPU is not in idle mode) over the total amount of time in the observation window.

We characterized one server node of our SeaMicro SM15k when running the Olio benchmark to derive the Pareto Frontier shown in Fig. 2. We set all CPU cores to a given P-state and ran the benchmark for an increasing number of concurrent users until we found a number of users that makes the CPU operate at 100 % of utilization. At this point, we ran the benchmark measuring power and instructions to determine the

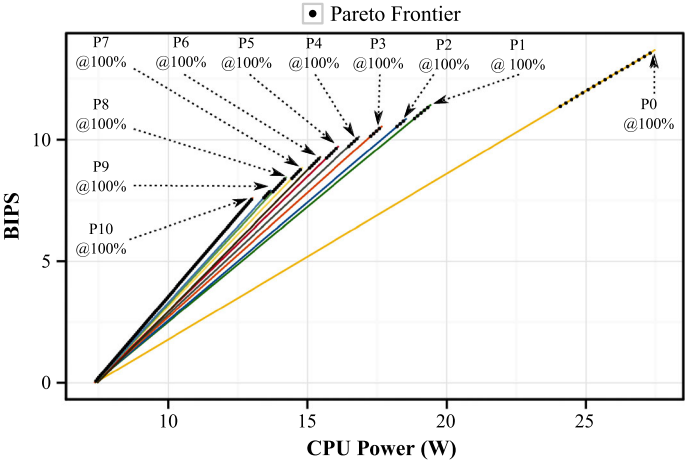

Fig. 2 Pareto Frontier of P-states Intel Xeon E3-1265Lv2

$BIPS$ at maximum utilization rate and power at maximum utilization rate for each P-state. We measured the power at idle for all P-states to obtain the idle power ratio. For this CPU, the idle power is around 7.3 W for all P-states.

To better understand Fig. 2, consider a performance level of ten $BIPS$. This requirement can be achieved in five different configurations consuming from 16.7 to 22 W as follows: P4-state under 100 % utilization rate, P3-state under 95 % utilization rate, P2-state under 91 % utilization rate, P1-state under 86 % utilization rate, and P0-state under 77 % utilization rate.

This observation leads us to add the utilization rate as an extra dimension to the optimization problem. Therefore, our problem is to find a P-state and a utilization rate for each CPU that minimizes power under a minimum performance requirement.

In Fig. 2, we derived an envelope curve, which is the Pareto Frontier of states. A point in the Pareto Frontier will provide the power consumption and the performance of the CPU under a P-state and a utilization rate. Moreover, there is no other state and utilization rate that would result in higher performance or lower power. The union of the Pareto Frontier states and the idle C-states constitutes the set of VPSs for a core. These VPSs are the states used by our optimization algorithm.

5 Experimental methodology

We organized our system as follows: a Web server cluster, which handles HTTP requests, and a power manager and load-balancer node that configures the cluster to an optimal power state and distributes the load accordingly, as illustrated in Fig. 3. Each Web server is composed of a back-end that runs MySQL 5.5.20, and a front-end that executes the Nginx 1.0.10 Web server with PHP 5.3.5. The back-end and the front-end run on the same node to fully utilize the CPU. The Web server application is the CloudSuite Web server benchmark [14] running Olio, a Web 2.0 Web-based social calendar. The power manager is the implementation of the power and performance

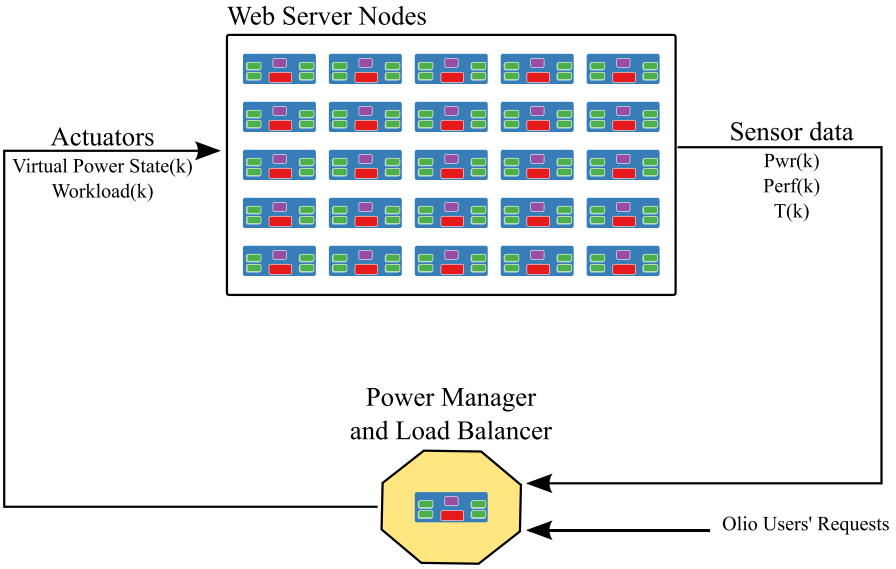

Fig. 3 High-level description of system organization to deploy our global optimization algorithm

policy (e.g., *Slack Recovery*, Linux *ondemand* governor). The load balancer is the HAProxy 1.5 [33], which is a fast, reliable, and open-source proxy solution.

The Web server cluster produces the sensor data, that is, readings of the power and performance values of each node in the cluster. The manager then takes into account the power optimization policy and the input workload to determine, on-the-fly, the best configuration of power states for all CPU nodes (i.e., which CPU node needs to have its power state or utilization rate changed). It then passes this new configuration information and the new workload distribution back to the cluster, which are then reconfigured while the workload is running.

The reconfiguration process is done in a fixed time window that needs to be tuned to the variation of the workload. At the same time, it depends on the overall time that it takes to do the following procedures: (1) gather the power and performance information, (2) run the algorithm, and 3) configure the new power states and CPU utilization. Assuming that this computation time is small (in the order of 250 ms for the SeaMicro), the time window can be driven by the variation on the system workload. Workloads that vary rapidly need a shorter time window, therefore the system can reconfigure itself for the new loads. If the overall workload varies gently, then longer workloads can be used.

In order to recreate a realistic workload stream, we obtained the load distribution over time for a specific server cluster hosting the chat room from a large Internet provider. This load is represented in Fig. 4, which shows the intra-day variation (for 37 h) of the load on the servers (where the load is represented as a percentage of the maximum load supported by the whole server cluster). We modified CloudSuite Web server benchmark [14] to follow this trend line creating a synthetic load representing the same variation with the same characteristics and same relative load percentage with respect to each experiment.

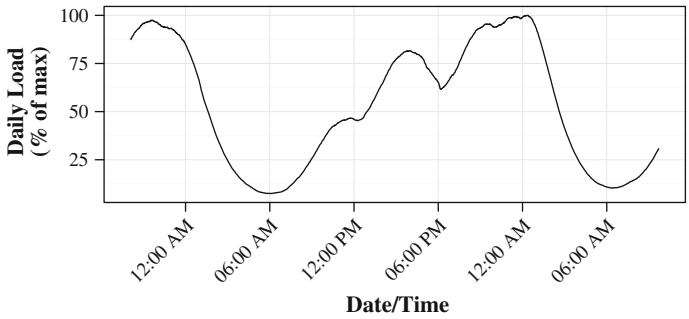

Fig. 4 Intra-day variation of the load on the server

The remainder of this section describes our test bed, the software tools used to evaluate the *Slack Recovery* implementation on a SM15k, the `Slack Recovery` algorithm, and the methodology for predicting performance.

5.1 Scaling-out Web server benchmark

We used 63 server nodes configured as follows: 25 nodes running the Web servers; 37 nodes running as client machines; and one node executing the power manager and the load balancer.

Although HAProxy is one of the fastest known proxy implementations, its TCP stack saturated and the operating system quickly ran out of TCP ports when we made all the client–server traffic pass through it. Therefore, we had to adapt the benchmark to alleviate the proxy server traffic. This was accomplished by developing a mechanism to emulate a reconfigurable network switching fabric. In this mechanism, the client requests a connection to the proxy (I in Fig. 5). The proxy forwards the connection to a server from the pool, selected based on a weighted round-robin policy (II in Fig. 5). During the next 5 s, the follow-up requests coming from the same client are directly addressed by the server selected in the first connection call (III in Fig. 5). Then, this connecting process is repeated.

On a large data center, it is possible to probe reconfigurable network switching fabrics that perform load balancing to implement our power management technique. Therefore, our requirements are the provision of a mechanism to change the weights of a round-robin load balancer and a metric that reports the amount of traffic redirected for each server.

5.2 Slack recovery algorithm

This section describes the *Slack Recovery* algorithm, which performs global power optimization. It was introduced in our previous work [4]. This heuristic algorithm is based on the idea of slack recovery (in power) to determine a near optimal solution. At first, it assigns power states to all CPU nodes to a state of the largest slack possible, that is, they are set to the highest performance virtual state, so there will be performance

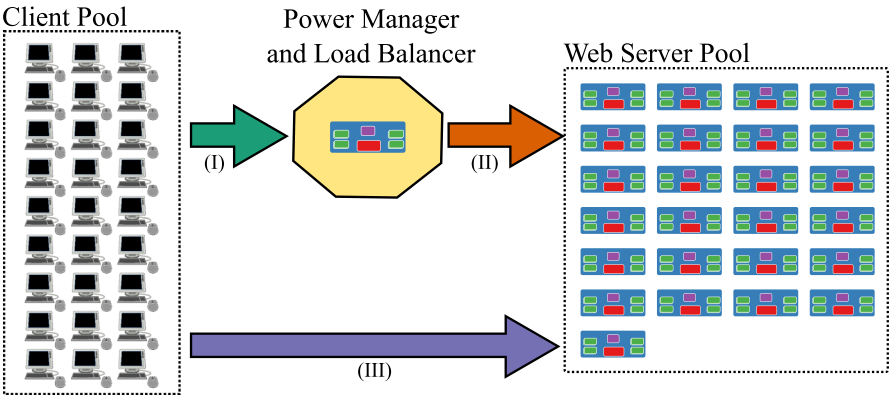

Fig. 5 System configuration to avoid proxy saturation

slack to be exchanged for power. In this case, the algorithm will switch the states of certain CPU nodes to decrease power and consequently lowering performance up to a threshold.

The algorithm runs over a cluster model to find the optimal configuration, and at the end it assigns the VPS to the physical cluster. The cluster model is a set of node models, which are implementations of the Pareto Frontier described in Sect. 4. The implementation needs a model, because it requires the estimation of power consumption and performance for different configurations. The model also reduces significantly the number of virtual state transitions.

For the sake of completeness in this text, we present the basic steps of the *Slack Recovery* algorithm. For further details, the reader should refer to Bergamaschi et al. [4]. Figure 6 lists the *Slack Recovery* pseudocode configured to minimize the power, given a minimum performance. The algorithm starts by finding an initial configuration state (line 5) that is able to sustain the minimum performance requirement. Next, it executes the slack routine, where it tries to decrease the power by moving to a lower performance state (lines 9 to 38). The routine iterates over all model nodes (lines 11 to 33), checks if the node could be moved to a higher VPS, which means a state with lower performance and power consumption (line 14). The algorithm provisions the performance (lines 16 and 17) in the cluster model and checks if it is greater than the performance threshold (line 19). If it satisfies the performance constraint, it provisions power and checks if it is lower than the minimum configuration so far (lines 22 to 24), storing this node. After iterating over all nodes, the algorithm moves the selected node (if any) to a higher VPS (lines 34 to 37). Finally, it assigns the VPS to the actual cluster nodes (line 40).

Example 1 To understand how `Slack Recovery` compares to the Linux *ondemand* governor, consider the following example: Assume a cluster with five Web servers composed of Intel Xeon E3-1265Lv2 processors. The power and performance curve of this CPU is shown in Fig. 2. Suppose that the performance requirement is 54 $BIPS$. By using these input parameters to the *Slack Recovery* algorithm, a configuration described in Table 1 is returned.

```
bool SlackRecoveryPerNode::minimizePower(
                float minBIPS, float powerCap,
                float actualPower, float actualBIPS) {

  if (!findInitialStateOnMinimizePower(minBIPS,currPower,currBIPS))
    return false; // No feasible solution

  // Now we are ready to execute the slack recovery algorithm
  do {
    nodeSel = NULL;
    for all nodes n do
      // There is slack: attempt to decrease power slack by moving
      // to a lower power/performance state
      if (n.getCurrentVirtualState() < NumVirtualStates -1) {
        // Check if we meet the constraint for the next state
        bipsChg = currBIPS  -
            provisionBIPS(currBIPS,n.getCurrentVirtualState(),
                n.getCurrentVirtualState()+1);
        if (currBIPS - bipsChg >= minBIPS) {
          // Provision power for the next state
          nextSttPower =
              provisionPower(currPower,
                             n.getCurrentVirtualState(),
                             n.getCurrentVirtualState()+1);
          powerChg = currPower - nextSttPower;
          if (powerChg > bestPowerChg) {
            nodeSel = n; // Remember this node
            bestPowerChg = powerChg;
            bestBipsChg = bipsChg;
          }
        }
      }
    }
    if (nodeSel != NULL) {
      currBIPS = currBIPS - bestBipsChg;
      nodeSel->oneStateMoveUp();
    }
  } while (nodeSel != NULL);
  // Do the assignment in the actual nodes
  assignVirtualStates();
```

Fig. 6 *Slack Recovery* algorithm pseudocode

Table 1 Server configuration when *Slack Recovery* is used

Server	A	B	C	D	E
P-state	P1	P2	P1	P1	P1
Utilization rate	96	100	95	95	95
Power (W)	18.96	18.43	18.84	18.84	18.84
$BIPS$	10.97	10.76	10.86	10.86	10.86

The configuration shown in Table 1 is able to sustain a performance of 54 $BIPS$, while the total power would be 93.9 W.

Figure 7 shows a pseudocode of the Linux *ondemand* governor implementation [26]. The algorithm works as follows: On every period of time, typically 10 ms, pool the

```
1 void ondemand() {
2   every X milliseconds {
3     for cpu in {CPU in the system} {
4       u = utilization_since_last_check(cpu)
5       if (u > UP_THRESHOLD)
6         increase_freq_to_max(cpu)
7       if (u < DOWN_THRESHOLD)
8         set_freq_80_perc_busy(cpu)
9     }
10  }
11 }
```

Fig. 7 Linux *Ondemand* governor pseudocode

CPUs in the system (line 2). If the utilization of the current CPU (line 5) is greater than a threshold (typically 95 %), it increases the current CPU frequency to the maximum (line 6). If the current utilization is lesser then the threshold (typically 95 %), jumps directly to the lowest frequency that can sustain a CPU load of 80 %.

Since the cluster is homogeneous (i.e., servers' CPUs are all the same), when running the workload with the Linux *ondemand* governor, we set the weights of the load balancer to the same value so that the load distribution among the server follows a round-robin policy. Therefore, all the servers have about the same load in the steady state.

If all servers have about the same load, for sustaining 54 $BIPS$, each server needs to keep a throughput of 10.8 $BIPS$. By observing the power and performance curve (Fig. 2), we can see that the only state that can keep the CPU 80 % busy, for a given $BIPS$ of 10.8 is P0. In this state, the CPU power is 23.1 W, and the total power for all five servers is 115.5 W.

As shown, the `Slack Recovery` is able to configure the cluster to a performance state that can sustain 54 $BIPS$ dissipating 93.9 W, while the the Linux *ondemand* governor dissipates 115 W to the same performance requirement.

5.3 Predicting the demanded performance

Slack Recovery configures the cluster in time window Π_t to sustain the load that will come in next time window Π_{t+1}. Therefore, we need to implement a mechanism that predicts the demanded performance for the next cycle. This performance is estimated based on the number of sessions opened by the HAProxy, that is, the rate of start connection requests to the proxy (I in Fig. 5) per second. On a big cluster, the session rate could be read from the network switching fabric and this number could be translated to the performance metric (i.e., $BIPS$).

Let us define session rate as the average number of sessions opened per time window. Differently from any microarchitectural performance metrics (e.g., $BIPS$), the session rate will always increase proportionally to the load even if some nodes are saturated (i.e., operating at a 100 % utilization rate). Figure 8 shows the relation of session rate to $BIPS$ for different benchmark loads (i.e., number of users). We can see that, even if the cluster is saturated, the session rate increases while $BIPS$ tends to be constant

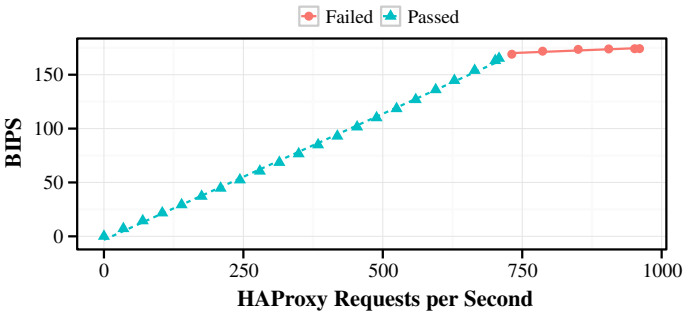

Fig. 8 Total cluster $BIPS$ versus HAProxy requests per second

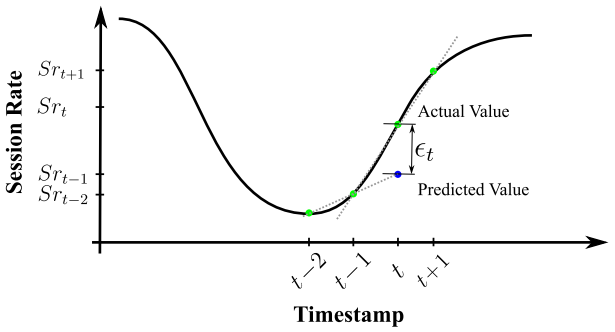

Fig. 9 Variables used to predict the value of the next session rate, Sr_{t+1}

at the maximum utilization. We computed a linear regression of the points at which the benchmark passes (i.e., the cluster is not saturated) to translate a given session rate into $BIPS$.

The predictor is developed using simple linear extrapolation as follows: let Sr_{t-1} be the session rate in time window Π_{t-1}, and Sr_t be the session rate in time window Π_t. To estimate the session rate Sr_{t+1} in time window Π_{t+1}, we find a tangent line as by using Eq. 1. Figure 9 illustrates the extrapolation method used by our predictor.

$$\begin{aligned} Sr_{t+1} &= \alpha \cdot (t+1-t) + Sr_t \\ &= \alpha + Sr_t \end{aligned} \tag{1}$$

where α is the slope of the line given by Eq. 2.

$$\begin{aligned} \alpha &= \frac{Sr_t - Sr_{t-1}}{t-(t-1)} \\ &= Sr_t - Sr_{t-1} \end{aligned} \tag{2}$$

Now replacing Eq. 2 in 1, Sr_{t+1} can be calculated as follows:

$$\begin{aligned} Sr_{t+1} &= \alpha + Sr_t \\ &= Sr_t - Sr_{t-1} + Sr_t \\ &= 2 \cdot Sr_t - Sr_{t-1} \end{aligned} \tag{3}$$

We also add an estimator error, ϵ_t, to amortize the prediction errors, calculated as follows:

$$\epsilon_t = \begin{cases} 0 & \text{if } t = 0 \\ Sr_{t_{\text{Actual}}} - Sr_{t_{\text{Predicted}}} & \text{if } t > 0 \end{cases} \tag{4}$$

Finally, the predicted session rate for the next time window is given by adding Eq. 3 to 4 as follows:

$$Sr_{t+1} = 2 \cdot Sr_t - Sr_{t-1} + \epsilon_t \tag{5}$$

The performance prediction is done based on the variation of the system-wide workload. We empirically observed that the total workload, when measured at discrete time intervals (i.e., 10 s in this work), tends to change smoothly with very few inversions. In addition, even when the load variation changes direction, the prediction may get it wrong for one or two intervals at most, before correcting itself.

After having predicted the performance for the next time window Π_{t+1}, we used this information as an input parameter for *Slack Recovery*. The algorithm returns a set of VPS, and we set each node to the corresponding VPS. The next Section, discusses the implementation of VPSs in our environment.

6 Virtual power state implementation

A VPS is a tuple containing a utilization rate and a P-state. While one can easily set a node's P-state; controlling each individual CPU utilization rate is more difficult. This section discusses our approach to control the CPU utilization rate on a Web server cluster.

In order to follow the remainder of this section, let us first define a number of variables used in the ensuing formulation. Table 2 presents the variables and their corresponding definitions.

Each server will handle a certain number of users until a point at which the CPU utilization rate reaches 100 %. Thus, one way to control the CPU utilization is to change the number of users connected to a server. Figure 10 shows the relation of the users connected to a server and its corresponding utilization rate for the Olio benchmark when the server is operating at highest frequency (P0-state). If the server is operating at other P-states, we normalize the utilization to P0-state using Eqs. 7 and 8.

The CPU utilization rate (Ψ) could be used to estimate the number of connected users (μ) to a Web server using a second-order degree polynomial regression using

Table 2 Variables used in the remainder of this section

Symbol	Definition
k	P-state index
s	Server index
μ	Number of users
χ_s	Number of connected users to the server s
ν_s	Number of new users to the server s
Ψ	Utilization rate
$T_{s,k}$	Target utilization rate for server s running at P_k-state
$K_{s,k}$	Current utilization rate for server s running at P_k-state
ϕT_s	Normalized target utilization rate for server s
ϕK_s	Normalized current utilization rate for server s
Δ_s	$\phi T_s - \phi K_s$
ϵ_s	Error in number of users
η_s	Number of expected users to the next iteration

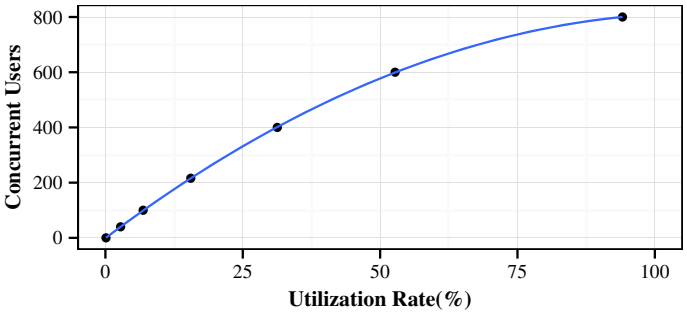

Fig. 10 Number of benchmark users (μ) versus utilization rate (Ψ). We use this relation to estimate the number of users that should be connected to a given server node

least square method from the points in Fig. 10, as given in Eq. 6.

$$\mu = f(\Psi) = -0.98 + 15.03 \cdot \Psi - 0.07 \cdot \Psi^2 \tag{6}$$

As we have shown in Sect. 5.1, the HAProxy will redirect a benchmark user to a Web server following a weighted round-robin policy, and this user-server assignment lasts 5 s. Therefore, we can control the number of users that will be redirected to a given machine by changing the server weight.

Therefore, if we want to reduce the utilization rate of a server (s), we need to reduce the number of new users (ν_s) arriving at the server and wait until the number of connected users (χ_s) drops, which means that we need to wait until some connected users finish their requests, entering into the connection phase again (where they request another server to the proxy). Figure 11 shows the variation of the CPU utilization rate over time when a server has 800 connected users, and we suddenly stop binding new users to it. We can see that the CPU utilization rate over time can be modeled as a step function, which facilitates the system modeling.

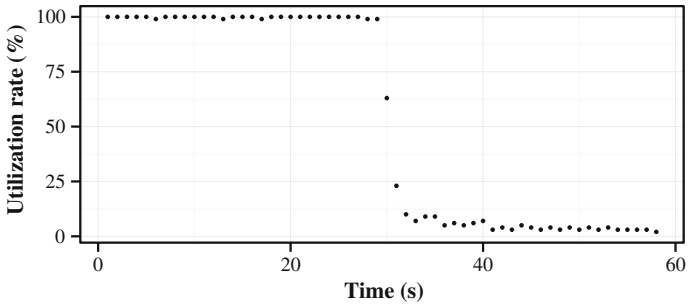

Fig. 11 Behavior of the utilization rate on time when server stops receiving new users

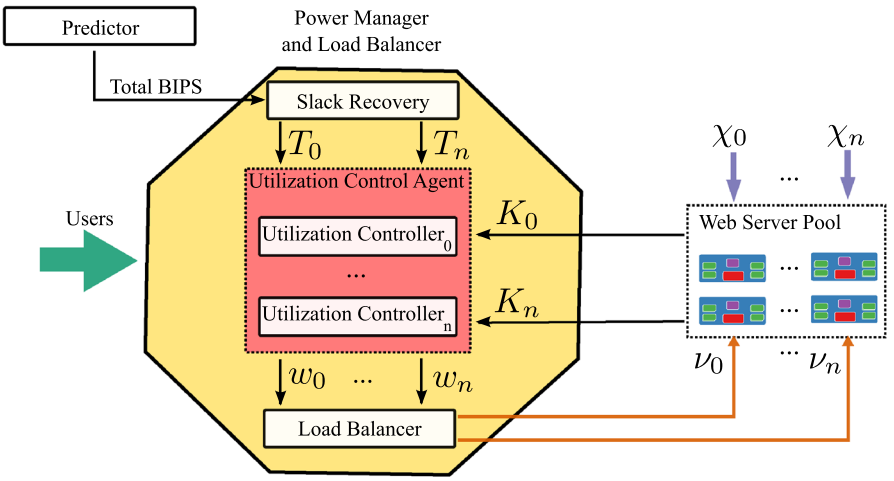

Fig. 12 An overview of the components involved on the implementation of the utilization controller mechanism

By using this methodology, we developed a utilization control agent (UCA) to control the CPU utilization rate of each server on the Web server cluster. Figure 12 illustrates the elements involved in this implementation.

Each server (s) running at P_k-state is assigned to a target utilization rate ($T_{s,k}$) by the *Slack Recovery* algorithm. The UCA receives from each server (s) its corresponding current P-state (k) and its current utilization rate ($K_{s,k}$). $T_{s,k}$ and $K_{s,k}$ are normalized to P_0-state, since the performance at 100 % utilization rate is different among P-states. This is accomplished using the information in Fig. 2, which displays the maximum performance ($BIPS_{100\,\%}$) for each P-state. Thus, the UCA calculates the normalized target utilization rate, (ϕT_s) and the normalized current utilization rate (ϕK_s) as stated by Eqs. 7 and 8.

$$\phi T_s = T_{s,k} \cdot \frac{BIPS_{100\,\%@P_k}}{BIPS_{100\,\%@P0}} \tag{7}$$

$$\phi K_s = K_{s,k} \cdot \frac{BIPS_{100\,\%@P_k}}{BIPS_{100\,\%@P0}} \tag{8}$$

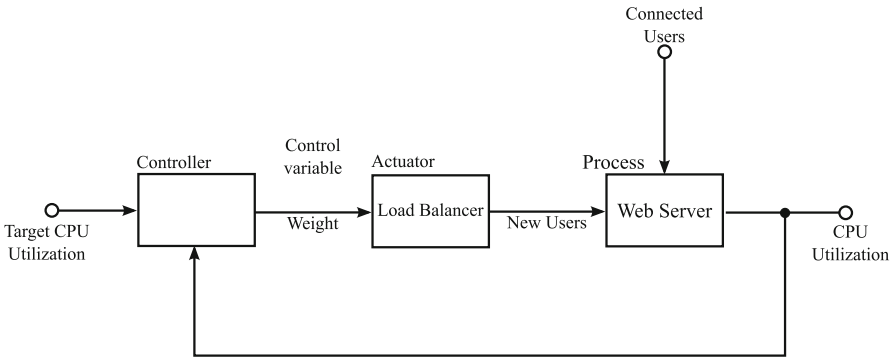

Fig. 13 Control loop approach for enforcing a CPU utilization rate

We develop a utilization controller as illustrated in Fig. 13. The controller box in this figure is the HAProxy, which will have changed its weights associated to each server. This will be translated to an increment or decrement to the number of new users. The number of new users plus the number of connected users will impact the utilization rate. The measured utilization rate of the server is compared to the target utilization rate and will be translated to a new weight closing the loop.

Each CPU utilization controller is modeled as a simple feedback control system from the Modern Control Theory. In such systems, the variable being controlled (i.e., CPU utilization rate) is measured and fed back to the controller to influence the controlled variable. To develop our controller, we first derive equations to describe our system and check for stability. The relations are described by Fig. 10, which shows the behavior of the CPU utilization when users are added to the system, and by Fig. 11, which shows how the utilization drops when we stop redirecting requests to the system.

A utilization controller converts ϕK_s to number of connected users (χ_s) as follows:

$$\chi_s = f(\phi K_s), \text{ where } f \text{ is given by Equation 6}$$

Let Δ_s be the difference between the normalized target utilization rate and the normalized current utilization rate as follows:

$$\Delta_s = \phi T_s - \phi K_s$$

The error in number of users (ϵ_s) is calculated using the absolute value of Δ_s as follows:

$$\epsilon_s = f(|\Delta_s|), \text{ where } f \text{ is given by Equation 6}$$

A Δ_s greater than zero means ϵ_s users must be added to the server s, to make it reach the target utilization rate. Otherwise, ϵ_s users must be removed from the server. Therefore, the number of expected users to the next iteration (η_s) on the s server is given by Eq. 9.

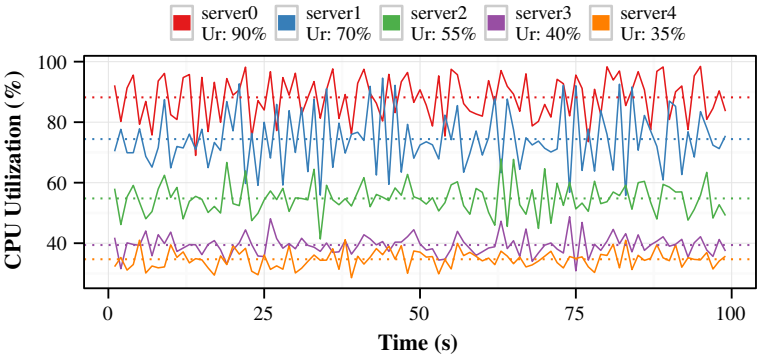

Fig. 14 Evaluation of the utilization controller mechanism for a fixed load. We see that the controller is able to place the servers at the target utilization

$$\eta_s = \begin{cases} \chi_s - \epsilon_s & \text{if } \Delta_s < 0 \\ \chi_s + \epsilon_s & \text{if } \Delta_s \geq 0 \end{cases} \tag{9}$$

The HAProxy allows weights from 0 to 254; thus, the UCA converts each η_s to a HAProxy weight w_s as follows:

$$w_s = \text{round}\left(\frac{\eta_s}{\max(\eta)} \cdot 254\right)$$

In order to evaluate the utilization control mechanism, we set up a small cluster composed of five servers. We configured the benchmark to generate a fixed input load that would be sufficient to keep the servers' utilization at their respective utilization rate targets. Our problem is to distribute the load across the servers by adjusting the load balancer weights to keep the servers' utilization rate at their respective targets. The controller will adjust the HAProxy weights to keep the utilization rates at their targets. Figure 14 shows the behavior of the utilization along 100 s for a fixed load.

The controller exhibits some variation around the target utilization rate because UCA is based on the number of users, but an user can do different types of operations. For example, an operation of adding a person uses more CPU than a logout operation. However, on the average the utilization rates converge to the targets as shown in the dotted lines.

7 Experimental results

This section shows the evaluation of our power management mechanism in two different scenarios: Constant number of users and variable number of users, where we configured the benchmark to follow the trend line described in Fig. 4.

We compared our mechanism to policies usually deployed in SeaMicro production systems, that is, the Linux *performance* governor, where the P-state is kept at maximum frequency when the CPU is executing, and to the Linux *ondemand* governor, where the operating system changes the frequency automatically.

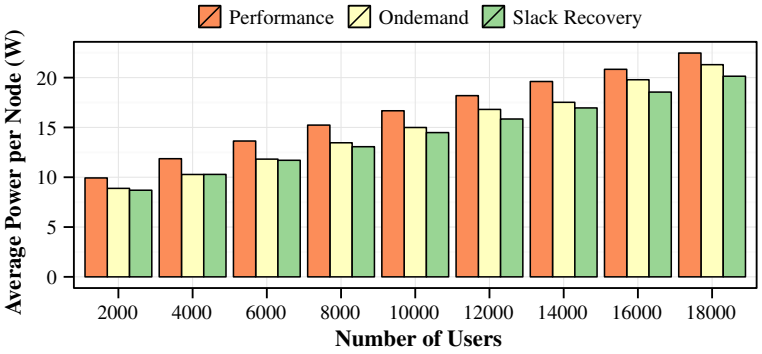

Fig. 15 Average power per node for different number of Olio benchmark users. The *Slack Recovery* exhibits higher power savings when the number of users is higher

We also evaluated a theoretical lower bound for our *Slack Recovery* heuristic, by considering that at idle the CPU would not consume any energy. This limit is the minimum power consumption on a hypothetical environment where the power in idle (i.e., at 0 % of utilization rate) is zero and where we can turn on/off the idle CPUs instantly (i.e., in zero time). This is a hypothetical environment because, in practice, the idle power is 7.3 W and turning on/off machines might have detrimental effects on the QoS due to the time necessary to switch the node back on.

7.1 Constant number of users

The first set of experiments was to evaluate the *Slack Recovery* implementation under constant number of users. Our 25 Web server node cluster supports up to 20,000 Olio users, when the front-end (PHP + nginx) and the back-end (MySQL) are running on the same machine. We ran the benchmark nine times changing the number of users on each execution. The different number of users impacts the CPU cluster utilization rates and the power and performance optimization space. Figure 15 shows the average power per server for different number of users.

Our results show that *Slack Recovery* can reduce the power consumption by up to 16 % when compared with the *performance* Linux governor, and 6.67 % when compared with the Linux *ondemand* governor. The *Slack Recovery* increased the response times. However, it was still able to meet the benchmark SLA constraints as shown in Fig. 16.

We observed that the higher power savings are concentrated when the load is higher. *Slack Recovery* trades power for performance keeping a minimum performance threshold. Therefore, it increases the response time for the benchmark (although still meeting the SLA requirement) and reduces the power consumption. When the load is low, there is much idleness on the system, and the idle power dominates the total power consumption.

Moreover, the SLA is indirectly taken into account when we set a performance threshold for the *Slack Recovery*. The algorithm globally configures the cluster to

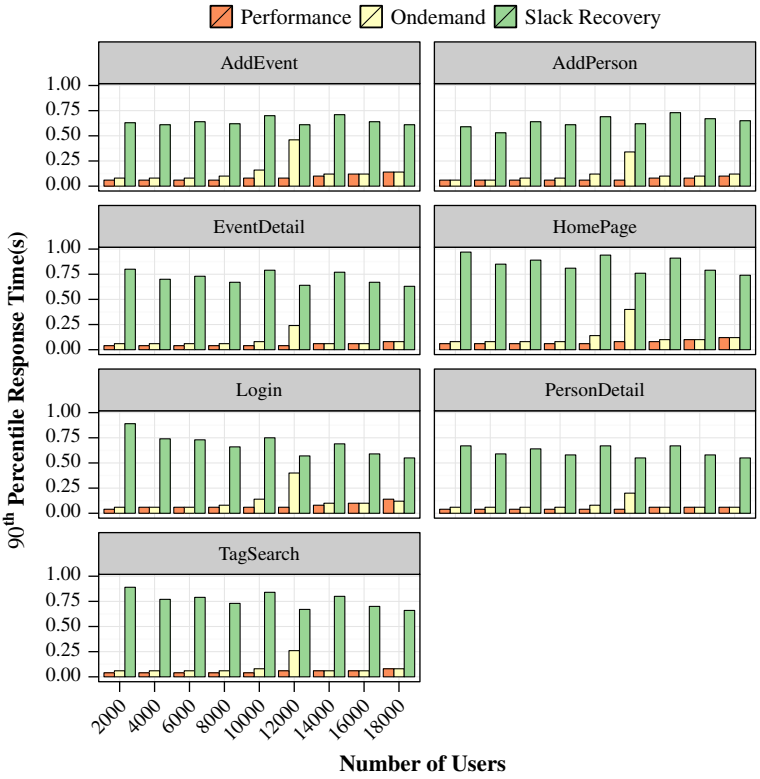

Fig. 16 SLAs for the Olio benchmark operations. The SLA requirements in 90th percentile response time must be 1 s for HomePage and Login; 2 s for EventDetail, PersonDetail, and TagSearch; 3 s for AddPerson; and 4 s AddEvent. Note that, for all cases, *Slack Recovery* could meet these requirements

sustain such performance requirement. In this way, it does not try to optimize SLAs, and it is possible that some requests get assigned to a CPU in a lower frequency, but despite that, the system was able to keep up with the benchmark requirements. If necessary to improve the SLA, we would need to make the performance requirements tighter, which might impact on the energy as well.

7.2 Variable number of users

The next experiment evaluated the behavior of the algorithm under a variable number of users. The 37-h load curve, taken from a real Internet cluster hosting the chat room, shown in Fig. 4 was shrunk to 5 h by calculating the average number of users over a 7-h time window to accelerate the experiments. The maximum number of users was set to 18,000 (about 90 % of the maximum capacity). The maximum follows the provision standards that reserve some processing capacity to handle any utilization spikes.

Figure 17 illustrates the power consumption of the cluster along the benchmark execution in this scenario. The results agree with those from the constant number

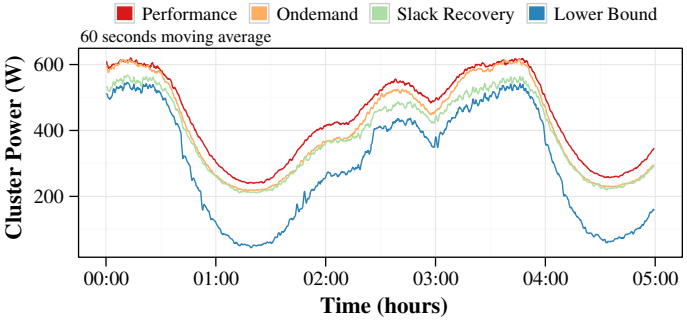

Fig. 17 Cluster power variation along 5 h of benchmark execution. The *Lower Bound* is the minimum power consumption that would result by, *Slack Recovery*, if the CPU power is zero watts when the utilization rate is 0 %

of users, where the higher power savings are in the region of higher loads. *Slack Recovery* was able to save 13.1 % of the power on average when compared with the Linux *performance* governor, and 5.6 % when compared with the Linux *ondemand* governor.

We also want to investigate the power behavior for each node. Figure 18 shows the power consumption for the individual nodes along the benchmark execution for the variable number of users.

The first observation is that three nodes (servers 22, 23, and 24) out of 25 are always idle, as we can see in Fig. 18. This is related to the assumption that the maximum load for the trend line curve corresponds to about 90 % of the maximum processing power capacity of the cluster. The algorithm concentrates the processing power to some nodes while others are placed in idle mode. This fact raises a question about the potential of adding to the algorithm the capacity of powering on/off nodes. We extrapolate our data to determine this value.

The extrapolation is done by setting power to zero instead of 7.3 W when the utilization rate of a node is at 0 % in a given time window. This is the lower bound showed in Fig. 18, because all idle nodes are considered off.

The lower bound corresponds to a reduction in power consumption of 39 % when compared to the Linux *performance* governor, to 30 % when compared to the Linux *ondemand* governor, and to 23 % to *Slack Recovery*.

On our SM15k cluster, a node takes about 5 min to be powered on. For this reason, we did not consider the possibility to power on/off nodes at first sight. However, this possibility is promising and could be feasible if we elaborate our demand predictor by taking into account the time overhead for power on/off the nodes. We leave the evaluation of this approach as a future work.

8 Conclusion

This article presents an adaptive power management system for a Web server cluster running on a real system. The cluster is composed by state-of-the art high density and power efficient architecture nodes, the AMD SeaMicro SM15k.

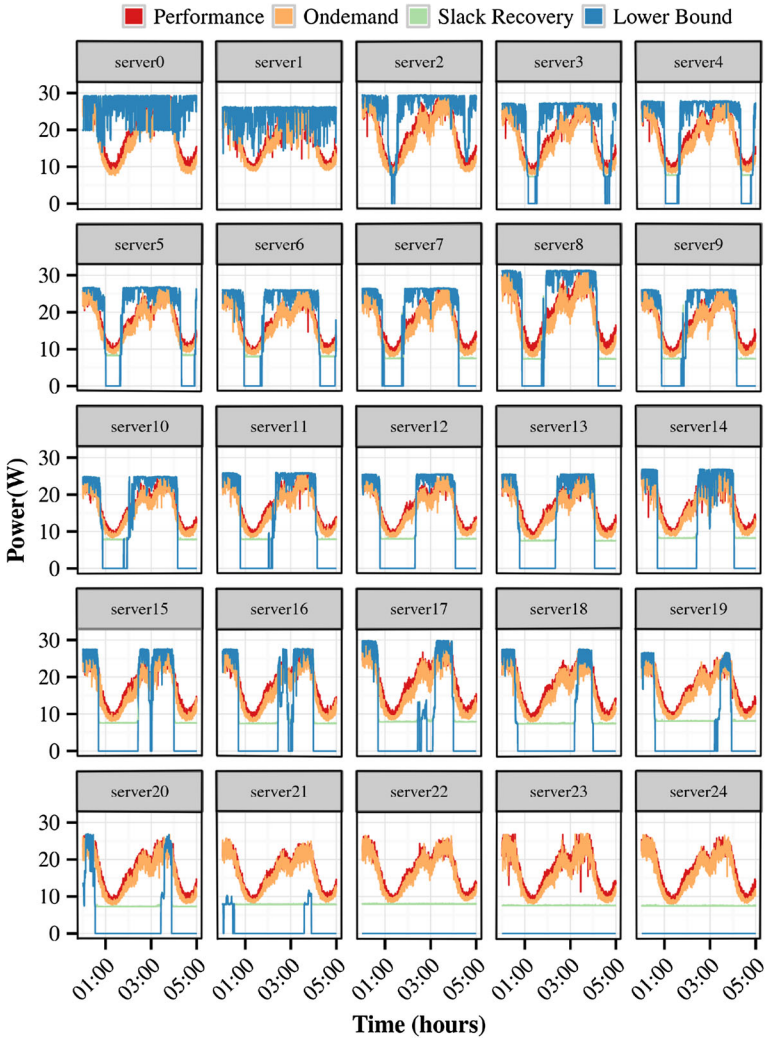

Fig. 18 Cluster power variation along 5 h of benchmark execution for each server. Observe that the last three servers are always idle

Our system is based on the *Slack Recovery* heuristic, which relies on the theoretical concept of VPSs, and was previously evaluated on a simulation environment. In order to bring it to a production cluster, we needed to show how VPSs could be implemented on practice. We did so by presenting new techniques to predict future demanded performance and a UCA. Our experimental evaluation showed that the UCA was capable of maintaining the cluster at the desired utilization rates.

Our power management system was compared to policies usually deployed in SeaMicro production systems: Linux *performance* governors and Linux *ondemand* governors. The experiments were conducted using Olio, a Web 2.0 Web-based social

calendar extracted from the CloudSuite Web server benchmark [14]. We showed that our *Slack Recovery*-based system could save up to 16 % of the power consumed in the cluster when compared with the Linux *performance* governor, and up to 6.67 % when compared with Linux *ondemand* governor. Finally, we evaluated the potential for power savings that could be brought by powering on/off cluster nodes, an alternative that may be promising. However, we decided not to include in this version of our system due to the penalty of turning on/off SM15k nodes. We plan to include this feature when evaluating our power management system in future works.

Acknowledgments Financial support for this study was provided by the Grant 2010/05389-5 from Sao Paulo Research Foundation (FAPESP) and AMD Research.

References

1. libpfm4 documentation. online, 2013. http://perfmon2.sourceforge.net/docs_v4.html. Accessed on 04th July 2013
2. Abbasi Z, Varsamopoulos G, Gupta SKS (2012) Tacoma: server and workload management in internet data centers considering cooling-computing power trade-off and energy proportionality. ACM Trans Archit Code Optim 9:2
3. AMD (2012) SeaMicro SM15000 Fabric Compute Systems. Sunnyvale, CA, USA
4. Bergamaschi RA, Piga L, Rigo S, Azevedo R, Araujo G (2012) Data center power and performance optimization through global selection of p-states and utilization rates. Sustain Computi Inform Syst
5. Bertini L, Leite JCB, Mossé D (2010) Power optimization for dynamic configuration in heterogeneous web server clusters. J Syst Softw 83:4
6. Bianchini R, Rajamony R (2004) Power and energy management for server systems. Computer
7. Brodowski D (2013) CPU frequency and voltage scaling code in the Linux(TM) kernel. Tech. rep., kernel.org
8. Chase JS, Anderson DC, Thakar PN, Vahdat AM, Doyle RP (2001) Managing energy and server resources in hosting centers. In: Proceedings of the eighteenth ACM symposium on operating systems principles, SOSP '01
9. Chen Y, Das A, Qin W, Sivasubramaniam A, Wang Q, Gautam N (2005) Managing server energy and operational costs in hosting centers. In: Proceedings of the 2005 ACM SIGMETRICS international conference on measurement and modeling of computer systems, SIGMETRICS '05, pp 303–314
10. Cochran R, Hankendi C, Coskun A, Reda S (2011) Pack & cap: adaptive dvfs and thread packing under power caps. In: 44th annual IEEE/ACM international symposium on microarchitecture
11. Economou D, Rivoire S, Kozyrakis C (2006) Full-system power analysis and modeling for server environments. In: Workshop on modeling benchmarking and simulation (MOBS)
12. Elnozahy EN, Kistler M, Rajamony R (2003) Energy-efficient server clusters. In: Proceedings of the 2nd international conference on power-aware computer systems, PACS'02
13. Elnozahy M, Kistler M, Rajamony R (2003) Energy conservation policies for web servers. In: Proceedings of the 4th conference on USENIX symposium on internet technologies and systems, vol 4, USITS'03
14. Ferdman M, Adileh A, Koçberber YO, Volos S, Alisafaee M, Jevdjic D, Kaynak IC, Popescu AD, Ailamaki A, Falsafi B (2012) Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In: Seventeenth international conference on architectural support for programming languages and operating systems (ASPLOS'12), pp 37–48
15. Filani D, He J, Gao S, Rajappa M, Kumar A, Shah P, Nagappan R (2008) Dynamic data center power management trends, issues, and solutions. Intel Technol J
16. Hackenberg D, Ilsche T, Schone R, Molka D, Schmidt M, Nagel W (2013) Power measurement techniques on standard compute nodes: a quantitative comparison. In: IEEE International symposium on performance analysis of systems and software (ISPASS), pp 194–204
17. Intel (2013) Intel 64 and IA-32 architectures software developer's manual vol 3B. System Programming Guide, Part 2. Santa Clara, CA, USA

18. Isci C, Buyuktosunoglu A, Cher C, Bose P, Martonosi M (2006) An analysis of efficient multi-core global power management policies: maximizing performance for a given power budget. In: 39th annual IEEE/ACM international symposium on microarchitecture (MICRO-39 2006)
19. Kant K, Murugan M, Du DHC (2012) Enhancing data center sustainability through energy-adaptive computing. J Emerg Technol Comput Syst 8:4
20. Koomey JG (2007) Estimating total power consumption by servers in the U.S. and the world. Tech. rep., Stanford University
21. Koomey JG (2011) Growth in data center electricity use 2005 to 2010. Stanford University, Tech. rep
22. Kusic D, Kephart JO, Hanson JE, Kandasamy N, Jiang G (2008) Power and performance management of virtualized computing environments via lookahead control. In: Proceedings of the 2008 international conference on autonomic computing, ICAC '08
23. Leverich J, Monchiero M, Talwar V, Ranganathan P, Kozyrakis C (2009) Power management of data-center workloads using per-core power gating. IEEE Comput Archit Lett 8(2):48–51
24. Malone C, Belady C (2006) EAC & PUE: metrics to characterize IT equipment & data center energy use. In: Digital power forum
25. Meisner D, Sadler CM, Barroso LA, Weber W-D, Wenisch TF (2011) Power management of online data-intensive services. In: Proceedings of the 38th annual international symposium on computer architecture, ISCA '11
26. Pallipadi V, Starikovskiy A (2006) The ondemand governor: past, present and future. In: Proceedings of Linux symposium
27. Piga L, Bergamaschi R, Klein F, Azevedo R, Rigo S (2011) Empirical web server power modeling and characterization. In: IEEE international symposium on workload characterization (IISWC), 2011, p 75
28. Rajamani K, Lefurgy C (2003) On evaluating request-distribution schemes for saving energy in server clusters. In: Proceedings of the 2003 IEEE international symposium on performance analysis of systems and software, ISPASS '03, pp 111–122
29. Rotem E, Naveh A, Rajwan D, Ananthakrishnan A, Weissmann E (2012) Power-management architecture of the intel microarchitecture code-named sandy bridge. IEEE Micro 32(2):20–27
30. Schneider D (2011) Under the hood at google and facebook. online, 2011. http://spectrum.ieee.org/telecom/internet/under-the-hood-at-google-and-facebook. Accessed on 20th Aug 2013
31. Schulz G (2009) The green and virtual data center, 1st edn. Auerbach Publications, Boston
32. Shen K, Shriraman A, Dwarkadas S, Zhang X (2012) Power and energy containers for multicore servers. In: Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on measurement and modeling of computer systems, SIGMETRICS '12
33. Tarreau W (2013) HAProxy configuration manual version 1.5. Tech. rep., HAProxy
34. Vogelsang T (2010) Understanding the energy consumption of dynamic random access memories. In: Proceedings of the 2010 43rd annual IEEE/ACM international symposium on microarchitecture, MICRO '43, pp 363–374
35. Winter JA, Albonesi DH, Shoemaker CA (2010) Scalable thread scheduling and global power management for heterogeneous many-core architectures. In: Proceedings of the 19th international conference on parallel architectures and compilation techniques, PACT '10